TEXTS IN COMPUTER SCIENCE

An Introduction to Kolmogorov Complexity and Its Applications



Ming Li
Paul Vitányi

THIRD EDITION

Springer

TEXTS IN COMPUTER SCIENCE

Editors

David Gries, Cornell University

Fred B. Schneider, Cornell University

TEXTS IN COMPUTER SCIENCE

Apt and Olderog, Verification of Sequential and Concurrent Programs, Second Edition

Alagar and Periyasamy, Specification of Software Systems

Back and von Wright, Refinement Calculus: A Systematic Introduction

Beidler, Data Structures and Algorithms: An Object-Oriented Approach Using Ada 95

Bergin, Data Structures Programming: With the Standard Template Library in C++

Brooks, C Programming: The Essentials for Engineers and Scientists

Brooks, Problem Solving with Fortran 90: For Scientists and Engineers

Burger and Burge, Digital Image Processing: An Algorithmic Introduction Using Java

Dandamudi, Fundamentals of Computer Organization and Design

Dandamudi, Introduction to Assembly Language Programming: For Pentium and RISC Processors, Second Edition

Dandamudi, Introduction to Assembly Language Programming: From 8086 to Pentium Processors

Fitting, First-Order Logic and Automated Theorem Proving, Second Edition

Grillmeyer, Exploring Computer Science with Scheme

Homer and Selman, Computability and Complexity Theory

Immerman, Descriptive Complexity

Jalote, An Integrated Approach to Software Engineering, Third Edition

(continued after index)

Ming Li
Paul Vitányi

An Introduction to Kolmogorov Complexity and Its Applications

Third Edition

with 49 figures

Springer

Ming Li
Cheriton School of Computer Science
University of Waterloo
Waterloo, ON N2L 3G1
Canada
mli@uwaterloo.ca

Paul Vitányi
Centrum voor Wiskunde en Informatica
Kruislaan 413
1098 SJ Amsterdam
The Netherlands
paul.vitanyi@cwi.nl

ISBN: 978-0-387-33998-6 e-ISBN: 978-0-387-49820-1
DOI: 10.1007/978-0-387-49820-1

Library of Congress Control Number: 2008923041

© 2008 Ming Li and Paul Vitányi; © 1997, 1993 Springer Science+Business Media, LLC
All rights reserved. This work may not be translated or copied in whole or in part without the written permission of the publisher (Springer Science+Business Media, LLC, 233 Spring Street, New York, NY 10013, USA), except for brief excerpts in connection with reviews or scholarly analysis. Use in connection with any form of information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed is forbidden.
The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

Cover image by Blair Somerville

Printed on acid-free paper

springer.com

To our wives, Wenhui and Pauline

Preface to the First Edition

"We are to admit no more causes of natural things" (as we are told by Newton) than "such as are both true and sufficient to explain their appearances." This central theme is basic to the pursuit of science, and goes back to the principle known as Occam's razor: "if presented with a choice between indifferent alternatives, then one ought to select the simplest one." Unconsciously or explicitly, informal applications of this principle in science and mathematics abound.

The conglomerate of different research threads drawing on an objective and absolute form of this approach appears to be part of a single emerging discipline, which will become a major applied science like information theory or probability theory. We aim at providing a unified and comprehensive introduction to the central ideas and applications of this discipline.

Intuitively, the amount of information in a finite string is the size (number of binary digits, or *bits*) of the shortest program that without additional data, computes the string and terminates. A similar definition can be given for infinite strings, but in this case the program produces element after element forever. Thus, a long sequence of 1's such as

$$\underbrace{11111\ldots1}_{\text{10,000 times}}$$

contains little information because a program of size about $\log 10{,}000$ bits outputs it:

```
for i := 1 to 10,000
print 1
```

Likewise, the transcendental number $\pi = 3.1415\ldots$, an infinite sequence of seemingly random decimal digits, contains but a few bits of information. (There is a short program that produces the consecutive digits of π forever.) Such a definition would appear to make the amount of information in a string (or other object) depend on the particular programming language used.

Fortunately, it can be shown that all reasonable choices of programming languages lead to quantification of the amount of absolute information in individual objects that is invariant up to an additive constant. We call this quantity the 'Kolmogorov complexity' of the object. If an object contains regularities, then it has a shorter description than itself. We call such an object 'compressible.'

The application of Kolmogorov complexity takes a variety of forms, for example, using the fact that some strings are extremely compressible; using the compressibility of strings as a selection criterion; using the fact that many strings are not compressible at all; and using the fact that

some strings may be compressed in principle, but that it takes a lot of effort to do so.

The theory dealing with the quantity of information in individual objects goes by names such as 'algorithmic information theory,' 'Kolmogorov complexity,' 'K-complexity,' 'Kolmogorov–Chaitin randomness,' 'algorithmic complexity,' 'stochastic complexity,' 'descriptional complexity,' 'minimum description length,' 'program-size complexity,' and others. Each such name may represent a variation of the basic underlying idea or a different point of departure. The mathematical formulation in each case tends to reflect the particular traditions of the field that gave birth to it, be it probability theory, information theory, theory of computing, statistics, or artificial intelligence.

This raises the question about the proper name for the area. Although there is a good case to be made for each of the alternatives listed above, and a name like 'Solomonoff–Kolmogorov–Chaitin complexity' would give proper credit to the inventors, we regard 'Kolmogorov complexity' as well entrenched and commonly understood, and we shall use it hereafter.

The mathematical theory of Kolmogorov complexity contains deep and sophisticated mathematics. Yet one needs to know only a small amount of this mathematics to apply the notions fruitfully in widely divergent areas, from sorting algorithms to combinatorial theory, and from inductive reasoning and machine learning to dissipationless computing.

Formal knowledge of basic principles does not necessarily imply the wherewithal to apply it, perhaps especially so in the case of Kolmogorov complexity. It is our purpose to develop the theory in detail and outline a wide range of illustrative applications. In fact, while the pure theory of the subject will have its appeal to the select few, the surprisingly large field of its applications will, we hope, delight the multitude.

The mathematical theory of Kolmogorov complexity is treated in Chapters 2, 3, and 4; the applications are treated in Chapters 5 through 8. Chapter 1 can be skipped by the reader who wants to proceed immediately to the technicalities. Section 1.1 is meant as a leisurely, informal introduction and peek at the contents of the book. The remainder of Chapter 1 is a compilation of material on diverse notations and disciplines drawn upon.

We define mathematical notions and establish uniform notation to be used throughout. In some cases we choose nonstandard notation since the standard one is homonymous. For instance, the notions 'absolute value,' 'cardinality of a set,' and 'length of a string' are commonly denoted in the same way as $|\cdot|$. We choose distinguishing notations $|\cdot|$, $d(\cdot)$, and $l(\cdot)$, respectively.

Briefly, we review the basic elements of computability theory and probability theory that are required. Finally, in order to place the subject in the appropriate historical and conceptual context we trace the main roots of Kolmogorov complexity.

This way the stage is set for Chapters 2 and 3, where we introduce the notion of optimal effective descriptions of objects. The length of such a description (or the number of bits of information in it) is its Kolmogorov complexity. We treat all aspects of the elementary mathematical theory of Kolmogorov complexity. This body of knowledge may be called *algorithmic complexity theory*. The theory of Martin-Löf tests for randomness of finite objects and infinite sequences is inextricably intertwined with the theory of Kolmogorov complexity and is completely treated. We also investigate the statistical properties of finite strings with high Kolmogorov complexity. Both of these topics are eminently useful in the applications part of the book. We also investigate the recursion-theoretic properties of Kolmogorov complexity (relations with Gödel's incompleteness result), and the Kolmogorov complexity version of information theory, which we may call 'algorithmic information theory' or 'absolute information theory.'

The treatment of *algorithmic probability theory* in Chapter 4 presupposes Sections 1.6, 1.11.2, and Chapter 3 (at least Sections 3.1 through 3.4). Just as Chapters 2 and 3 deal with the optimal effective description length of objects, we now turn to optimal (greatest) effective probability of objects. We treat the elementary mathematical theory in detail. Subsequently, we develop the theory of effective randomness tests under arbitrary recursive distributions for both finite and infinite sequences. This leads to several classes of randomness tests, each of which has a universal randomness test. This is the basis for the treatment of a mathematical theory of inductive reasoning in Chapter 5 and the theory of algorithmic entropy in Chapter 8.

Chapter 5 develops a general theory of inductive reasoning and applies the developed notions to particular problems of inductive inference, prediction, mistake bounds, computational learning theory, and minimum description length induction in statistics. This development can be viewed both as a resolution of certain problems in philosophy about the concept and feasibility of induction (and the ambiguous notion of 'Occam's razor'), as well as a mathematical theory underlying computational machine learning and statistical reasoning.

Chapter 6 introduces the incompressibility method. Its utility is demonstrated in a plethora of examples of proving mathematical and computational results. Examples include combinatorial properties, the time complexity of computations, the average-case analysis of algorithms such as Heapsort, language recognition, string matching, pumping lemmas in

formal language theory, lower bounds in parallel computation, and Turing machine complexity. Chapter 6 assumes only the most basic notions and facts of Sections 2.1, 2.2, 3.1, 3.3.

Some parts of the treatment of resource-bounded Kolmogorov complexity and its many applications in computational complexity theory in Chapter 7 presuppose familiarity with a first-year graduate theory course in computer science or basic understanding of the material in Section 1.7.4. Sections 7.5 and 7.7 on universal optimal search and logical depth only require material covered in this book. The section on logical depth is technical and can be viewed as a mathematical basis with which to study the emergence of life-like phenomena—thus forming a bridge to Chapter 8, which deals with applications of Kolmogorov complexity to relations between physics and computation.

Chapter 8 presupposes parts of Chapters 2, 3, 4, the basics of information theory as given in Section 1.11, and some familiarity with college physics. It treats physical theories like dissipationless reversible computing, information distance and picture similarity, thermodynamics of computation, statistical thermodynamics, entropy, and chaos from a Kolmogorov complexity point of view. At the end of the book there is a comprehensive listing of the literature on theory and applications of Kolmogorov complexity and a detailed index.

Acknowledgments

We thank Greg Chaitin, Péter Gács, Leonid Levin, and Ray Solomonoff for taking the time to tell us about the early history of our subject and for introducing us to many of its applications. Juris Hartmanis and Joel Seiferas initiated us into Kolmogorov complexity in various ways.

Many people gave substantial suggestions for examples and exercises, or pointed out errors in a draft version. Apart from the people already mentioned, these are, in alphabetical order, Eric Allender, Charles Bennett, Piotr Berman, Robert Black, Ron Book, Dany Breslauer, Harry Buhrman, Peter van Emde Boas, William Gasarch, Joe Halpern, Jan Heering, G. Hotz, Tao Jiang, Max Kanovich, Danny Krizanc, Evangelos Kranakis, Michiel van Lambalgen, Luc Longpré, Donald Loveland, Albert Meyer, Lambert Meertens, Ian Munro, Pekka Orponen, Ramamohan Paturi, Jorma Rissanen, Jeff Shallit, A.Kh. Shen', J. Laurie Snell, Th. Tsantilas, John Tromp, Vladimir Uspensky, N.K. Vereshchagin, Osamu Watanabe, and Yaacov Yesha. Apart from them, we thank the many students and colleagues who contributed to this book.

We especially thank Péter Gács for the extraordinary kindness of reading and commenting in detail on the entire manuscript, including the exercises. His expert advice and deep insight saved us from many pitfalls and misunderstandings. Piergiorgio Odifreddi carefully checked and commented on the first three chapters. Parts of the book have been

tested in one-semester courses and seminars at the University of Amsterdam in 1988 and 1989, the University of Waterloo in 1989, Dartmouth College in 1990, the Universitat Polytècnica de Catalunya in Barcelona in 1991/1992, the University of California at Santa Barbara, Johns Hopkins University, and Boston University in 1992/1993.

This document has been prepared using the LaTeX system. We thank Donald Knuth for TeX, Leslie Lamport for LaTeX, and Jan van der Steen at CWI for online help. Some figures were prepared by John Tromp using the xpic program.

The London Mathematical Society kindly gave permission to reproduce a long extract by A.M. Turing. The Indian Statistical Institute, through the editor of *Sankhyā*, kindly gave permission to quote A.N. Kolmogorov.

We gratefully acknowledge the financial support by NSF Grant DCR-8606366, ONR Grant N00014-85-k-0445, ARO Grant DAAL03-86-K-0171, the Natural Sciences and Engineering Research Council of Canada through operating grants OGP-0036747, OGP-046506, and International Scientific Exchange Awards ISE0046203, ISE0125663, and NWO Grant NF 62-376. The book was conceived in late Spring 1986 in the Valley of the Moon in Sonoma County, California. The actual writing lasted on and off from autumn 1987 until summer 1993.

One of us [PV] gives very special thanks to his lovely wife Pauline for insisting from the outset on the significance of this enterprise. The Aiken Computation Laboratory of Harvard University, Cambridge, Massachusetts, USA; the Computer Science Department of York University, Ontario, Canada; the Computer Science Department of the University of Waterloo, Ontario, Canada; and CWI, Amsterdam, the Netherlands provided the working environments in which this book could be written.

Preface to the Second Edition

When this book was conceived ten years ago, few scientists realized the width of scope and the power for applicability of the central ideas. Partially because of the enthusiastic reception of the first edition, open problems have been solved and new applications have been developed. We have added new material on the relation between data compression and minimum description length induction, computational learning, and universal prediction; circuit theory; distributed algorithmics; instance complexity; CD compression; computational complexity; Kolmogorov random graphs; shortest encoding of routing tables in communication networks; resource-bounded computable universal distributions; average case properties; the equality of statistical entropy and expected Kolmogorov complexity; and so on. Apart from being used by researchers and as a reference work, the book is now commonly used for graduate courses and seminars. In recognition of this fact, the second edition has

been produced in textbook style. We have preserved as much as possible the ordering of the material as it was in the first edition. The many exercises bunched together at the ends of some chapters have been moved to the appropriate sections. The comprehensive bibliography on Kolmogorov complexity at the end of the book has been updated, as have the 'History and References' sections of the chapters. Many readers were kind enough to express their appreciation for the first edition and to send notification of typos, errors, and comments. Their number is too large to thank them individually, so we thank them all collectively.

Preface to the Third Edition

The general area of reasoning based on shortest description length continues to coalesce. Simultaneously, the emphasis in handling of information in computers and communication networks continues to move from being random-variable based to being individual-outcome based. Practically speaking, this has resulted in a number of spectacular real-life applications of Kolmogorov complexity, where the latter is replaced by compression programs. The general area has branched out into subareas, each with its own specialized books or treatments. This work, through its subsequent editions, has been both a catalyst and an outcome of these trends. The third edition endeavors to capture the essence of the state of the art at the end of the first decade of the new millennium. It is a corrected and greatly expanded version of the earlier editions. Many people contributed, and we thank them all collectively.

How to Use This Book

The technical content of this book consists of four layers. The main text is the first layer. The second layer consists of examples in the main text. These elaborate the theory developed from the main theorems. The third layer consists of nonindented, smaller-font paragraphs interspersed with the main text. The purpose of such paragraphs is to have an explanatory aside, to raise some technical issues that are important but would distract attention from the main narrative, or to point to alternative or related technical issues. Much of the technical content of the literature on Kolmogorov complexity and related issues appears in the fourth layer, the exercises. When the idea behind a nontrivial exercise is not our own, we have tried to give credit to the person who originated the idea. Corresponding references to the literature are usually given in comments to an exercise or in the historical section of that chapter.

Starred sections are not really required for the understanding of the sequel and can be omitted at first reading. The application sections are not starred. The exercises are grouped together at the end of main sections. Each group relates to the material in between it and the previous group. Each chapter is concluded by an extensive historical section with full

references. For convenience, all references in the text to the Kolmogorov complexity literature and other relevant literature are given in full where they occur. The book concludes with a References section intended as a separate exhaustive listing of the literature restricted to the theory and the direct applications of Kolmogorov complexity. There are reference items that do not occur in the text and text references that do not occur in the References. We added a very detailed Index combining the index to notation, the name index, and the concept index. The page number where a notion is defined first is printed in boldface. The initial part of the Index is an index to notation. Names such as 'J. von Neumann' are indexed European style 'Neumann, J. von.'

The exercises are sometimes trivial, sometimes genuine exercises, but more often compilations of entire research papers or even well-known open problems. There are good arguments to include both: the easy and real exercises to let the student exercise his comprehension of the material in the main text; the contents of research papers to have a comprehensive coverage of the field in a small number of pages; and research problems to show where the field is (or could be) heading. To save the reader the problem of having to determine which is which: "I found this simple exercise in number theory that looked like Pythagoras's Theorem. Seems difficult." "Oh, that is Fermat's Last Theorem; it took three hundred and fifty years to solve it . . . ," we have adopted the system of *rating numbers* used by D.E. Knuth [*The Art of Computer Programming, Volume 1: Fundamental Algorithms*, Addison-Wesley, 1973. Second Edition, pp. xvii–xix]. The interpretation is as follows:

00 A very easy exercise that can be answered immediately, from the top of your head, if the material in the text is understood.

10 A simple problem to exercise understanding of the text. Use fifteen minutes to think, and possibly pencil and paper.

20 An average problem to test basic understanding of the text and may take one or two hours to answer completely.

30 A moderately difficult or complex problem taking perhaps several hours to a day to solve satisfactorily.

40 A quite difficult or lengthy problem, suitable for a term project, often a significant result in the research literature. We would expect a very bright student or researcher to be able to solve the problem in a reasonable amount of time, but the solution is not trivial.

50 A research problem that, to the authors' knowledge, is open at the time of writing. If the reader has found a solution, he is urged to write it up for publication; furthermore, the authors of this book would appreciate hearing about the solution as soon as possible.

This scale is logarithmic: a problem of rating 17 is a bit simpler than average. Problems with rating 50, subsequently solved, will appear in a next edition of this book with rating about 45. Rates are sometimes based on the use of solutions to earlier problems. The rating of an exercise is based on that of its most difficult item, but not on the number of items. Assigning accurate rating numbers is impossible—one man's meat is another man's poison—and our rating will differ from ratings by others.

An orthogonal rating M implies that the problem involves more mathematical concepts and motivation than is necessary for someone who is primarily interested in Kolmogorov complexity and applications. Exercises marked HM require the use of calculus or other higher mathematics not developed in this book. Some exercises are marked with a •; and these are especially instructive or useful. Exercises marked O are problems that are, to our knowledge, unsolved at the time of writing. The rating of such exercises is based on our estimate of the difficulty of solving them. Obviously, such an estimate may be totally wrong.

Solutions to exercises, or references to the literature where such solutions can be found, appear in the Comments paragraph at the end of each exercise. Nobody is expected to be able to solve all exercises.

The material presented in this book draws on work that until now was available only in the form of advanced research publications, possibly not translated into English, or was unpublished. A large portion of the material is new. The book is appropriate for either a one- or a two-semester introductory course in departments of mathematics, computer science, physics, probability theory and statistics, artificial intelligence, cognitive science, and philosophy. Outlines of possible one-semester courses that can be taught using this book are presented below.

Fortunately, the field of descriptional complexity is fairly young and the basics can still be comprehensively covered. We have tried to the best of our abilities to read, digest, and verify the literature on the topics covered in this book. We have taken pains to establish correctly the history of the main ideas involved. We apologize to those who have been unintentionally slighted in the historical sections. Many people have generously and selflessly contributed to verify and correct drafts of the various editions of this book. We thank them below and apologize to those we forgot. In a work of this scope and size there are bound to remain factual errors and incorrect attributions. We greatly appreciate notification of errors or any other comments the reader may have, preferably by email, in order that future editions may be corrected.

Outlines of One-Semester Courses

We have mapped out a number of one-semester courses on a variety of topics. These topics range from basic courses in theory and applications to special-interest courses in learning theory, randomness, or information theory using the Kolmogorov complexity approach.

PREREQUISITES: Sections 1.1, 1.2, 1.7 (except Section 1.7.4).

I. Course on Basic Algorithmic Complexity and Applications

TYPE OF COMPLEXITY	THEORY	APPLICATIONS
plain complexity	2.1, 2.2, 2.3	4.4, Chapter 6
prefix complexity	1.11.2, 3.1 3.3, 3.4	5.1, 5.1.3, 5.2, 5.4 8.2, 8.3, 8.4
resource-bounded complexity	7.1, 7.5, 7.7	7.2, 7.3, 7.6, 7.7

II. Course on Algorithmic Complexity

TYPE OF COMPLEXITY	BASICS	RANDOMNESS	ALGORITHMIC PROPERTIES
state × symbol	1.12		
plain complexity	2.1, 2.2, 2.3	2.4	2.7
prefix complexity	1.11.2, 3.1 3.3, 3.4	3.5	3.7, 3.8
monotone complexity	4.5 (intro)	4.5.4	

III. Course on Algorithmic Randomness

RANDOMNESS TESTS ACCORDING TO	COMPLEXITY USED	FINITE STRINGS	INFINITE SEQUENCES
von Mises			1.9
Martin-Löf	2.1, 2.2	2.4	2.5
prefix complexity	1.11.2, 3.1, 3.3, 3.4	3.5	3.6, 4.5.6
general discrete	1.6 (intro), 4.3.1	4.3	
general continuous	1.6 (intro), 4.5 (intro), 4.5.1		4.5

IV. Course on Algorithmic Information Theory and Applications

TYPE OF COMPLEXITY USED	BASICS	ENTROPY	SYMMETRY OF INFORMATION
classical information theory	1.11	1.11	1.11
plain complexity	2.1, 2.2	2.8	2.8
prefix complexity	3.1, 3.3, 3.4	8.1	3.8, 3.9.1
resource-bounded	7.1		Exercises 7.1.12 7.1.13
applications	8.3 8.4	8.1.1, 8.5, 8.6	Theorem 7.2.6 Exercise 6.10.15

V. Course on Algorithmic Probability Theory, Learning, Inference, and Prediction

Theory	Basics	Universal Distribution	Applications to Inference
classical probability	1.6, 1.11.2		1.6
algorithmic complexity	2.1, 2.2, 2.3 3.1, 3.3, 3.4		8
algorithmic discrete probability	4.2, 4.1 4.3 (intro)	4.3.1, 4.3.2 4.3.3, 4.3.4, 4.3.6	
algorithmic contin. probability	4.5 (intro)	4.5.1, 4.5.2 4.5.4, 4.5.8	5.2
Solomonoff's inductive inference	5.1, 5.1.3, 5.2 5.3, 8	5.2.5, 5.3.3, 5.4 5.4.5	5.1.3
MDL and nonprobabilistic statistics	5.4		5.4, 5.5

VI. Course on the Incompressibility Method

Chapter 2 (Sections 2.1, 2.2, 2.4, 1.11.5, 2.8), Chapter 3 (mainly Sections 3.1, 3.3), Section 4.4, and Chapters 6 and 7. The course covers the basics of the theory with many applications in proving upper and lower bounds on the running time and space use of algorithms.

VII. Course on Randomness, Information, and Physics

Course III and Chapter 8. In physics the applications of Kolmogorov complexity include theoretical illuminations of foundational issues. For example, the approximate equality of statistical entropy and expected Kolmogorov complexity, the nature of entropy, a fundamental resolution of the Maxwell's Demon paradox. However, also more concrete applications such as information distance, normalized information distance and its applications to phylogeny, clustering, classification, and relative semantics of words and phrases, as well as thermodynamics of computation are covered.

Contents

Preface to the First Edition vii
Preface to the Second Edition xi
Preface to the Third Edition xii
How to Use This Book xii
Outlines of One-Semester Courses xv
Contents xvii
List of Figures xxi

1 Preliminaries **1**
1.1 A Brief Introduction 1
1.2 Prerequisites and Notation 7
1.3 Numbers and Combinatorics 8
1.4 Binary Strings 12
1.5 Asymptotic Notation 15
1.6 Basics of Probability Theory 18
1.7 Basics of Computability Theory 24
1.8 The Roots of Kolmogorov Complexity 47
1.9 Randomness 49
1.10 Prediction and Probability 59

1.11 Information Theory and Coding 65
1.12 State × Symbol Complexity 90
1.13 History and References 92

2 Algorithmic Complexity 101
2.1 The Invariance Theorem 104
2.2 Incompressibility 116
2.3 C as an Integer Function 126
2.4 Random Finite Sequences 133
2.5 *Random Infinite Sequences 143
2.6 Statistical Properties of Finite Sequences 165
2.7 Algorithmic Properties of C 174
2.8 Algorithmic Information Theory 186
2.9 History and References 192

3 Algorithmic Prefix Complexity 197
3.1 The Invariance Theorem 200
3.2 *Sizes of the Constants 206
3.3 Incompressibility 211
3.4 K as an Integer Function 216
3.5 Random Finite Sequences 218
3.6 *Random Infinite Sequences 220
3.7 Algorithmic Properties of K 239
3.8 *Complexity of Complexity 241
3.9 *Symmetry of Algorithmic Information 244
3.10 History and References 255

4 Algorithmic Probability 259
4.1 Semicomputable Functions Revisited 260
4.2 Measure Theory 262
4.3 Discrete Sample Space 265
4.4 Universal Average-Case Complexity 290
4.5 Continuous Sample Space 294
4.6 Universal Average-Case Complexity, Continued 330
4.7 History and References 331

5 Inductive Reasoning **339**

5.1 Introduction 339
5.2 Solomonoff's Theory of Prediction 348
5.3 Simple Pac-Learning 370
5.4 Hypothesis Identification by MDL 382
5.5 Nonprobabilistic Statistics 401
5.6 History and References 431

6 The Incompressibility Method **441**

6.1 Three Examples 442
6.2 High-Probability Properties 448
6.3 Combinatorics 451
6.4 Kolmogorov Random Graphs 461
6.5 Compact Routing 469
6.6 Average-Case Analysis of Sorting 476
6.7 Longest Common Subsequence 486
6.8 Formal Language Theory 490
6.9 Online CFL Recognition 497
6.10 Turing Machine Time Complexity 502
6.11 Communication Complexity 516
6.12 Circuit Complexity 521
6.13 History and References 524

7 Resource-Bounded Complexity **531**

7.1 Mathematical Theory 532
7.2 Language Compression 550
7.3 Computational Complexity 562
7.4 Instance Complexity 571
7.5 Kt and Universal Search 577
7.6 Time-Limited Universal Distributions 582
7.7 Logical Depth 589
7.8 History and References 596

8 Physics, Information, and Computation 601

8.1 Information Theory 602

8.2 Reversible Computation 629

8.3 Information Distance 641

8.4 Normalized Information Distance 660

8.5 Thermodynamics 674

8.6 Entropy Revisited 686

8.7 Quantum Kolmogorov Complexity 696

8.8 Compression in Nature 711

8.9 History and References 714

References 723

Index 765

List of Figures

1.1 Turing machine 28

1.2 Inferred probability for increasing n 61

1.3 Binary tree for $E(1) = 0$, $E(2) = 10$, $E(3) = 110$, $E(4) = 111$ 74

1.4 Binary tree for $E(1) = 0$, $E(2) = 01$, $E(3) = 011$, $E(4) = 0111$ 75

2.1 The graph of the integer function $C(x)$ 127

2.2 The graph of the integer function $C(x|l(x))$ 129

2.3 Test of Example 2.4.1 135

2.4 Complexity oscillations of initial segments of high–complexity infinite sequences 146

2.5 Three notions of 'chaotic' infinite sequences 155

3.1 The 425-bit universal combinator U' in pixels 210

3.2 The graphs of $K(x)$ and $K(x|l(x))$ 217

3.3 Complexity oscillations of a typical random sequence ω . 224

3.4 K-complexity criteria for randomness of infinite sequences 224

3.5 Complexity oscillations of Ω 226

4.1 Graph of $\mathbf{m}(x)$ with lower bound $1/(x \cdot \log x \cdot \log\log x \cdots)$ 269

4.2 Relations between five complexities with $l(x) = n$ 308

5.1 Trivial consistent automaton 341
5.2 Smallest consistent automaton 341
5.3 Table of sample data set 393
5.4 Imperfect decision tree 394
5.5 Perfect decision tree 395
5.6 Kolmogorov's structure function (ML estimator) 406
5.7 Relations among the structure functions 409
5.8 Graph of $h_x(i)$ in strip around $h(i)$ 411
5.9 Some shapes of the structure function 412
5.10 Positive randomness and negative randomness 438

6.1 Single-tape Turing machine 443
6.2 The two possible nnis on (u, v): swap $B \leftrightarrow C$ or $B \leftrightarrow D$. 483
6.3 The nni distance between (i) and (ii) is two 484
6.4 Multitape Turing machine 497

8.1 A rate-distortion function for Hamming distortion 616
8.2 Denoising of the noisy cross 621
8.3 Reversible Boolean gates 631
8.4 Implementing reversible AND gate and NOT gate 632
8.5 Controlling billiard-ball movements 633
8.6 A billiard-ball computer 634
8.7 Combining irreversible computations of y from x and x from y to achieve a reversible computation of y from x . 649
8.8 Reversible execution of concatenated programs for $(y|x)$ and $(z|y)$ to transform x into z 650
8.9 The evolutionary tree built from complete mammalian mtDNA sequences 665
8.10 Clustering of heterogeneous file types 667
8.11 Carnot cycle 676
8.12 Heat engine 676

8.13 State space 679
8.14 Atomic spin in CuO_2 at low temperature 683
8.15 Regular 'up' and 'down' spins 683
8.16 Adiabatic demagnetization to achieve low temperature . 685
8.17 Algorithmic entropy: left a random micro state, right a regular micro state 688
8.18 Szilard engine 689
8.19 The maze: a binary tree constructed from matches 712
8.20 Time required for *Formica sanguinea* scouts to transmit information about the direction to the syrup to the forager ants 713

1

Preliminaries

1.1 A Brief Introduction

Suppose we want to describe a given object by a finite binary string. We do not care whether the object has many descriptions; however, each description should describe but one object. From among all descriptions of an object we can take the length of the shortest description as a measure of the object's complexity. It is natural to call an object 'simple' if it has at least one short description, and to call it 'complex' if all of its descriptions are long.

But now we are in danger of falling into the trap so eloquently described in the Richard–Berry paradox, where we define a natural number as "the least natural number that cannot be described in fewer than twenty words." If this number does exist, we have just described it in thirteen words, contradicting its definitional statement. If such a number does not exist, then all natural numbers can be described in fewer than twenty words. We need to look very carefully at the notion of 'description.'

Assume that each description describes at most one object. That is, there be a specification method D that associates at most one object x with a description y. This means that D is a function from the set of descriptions, say Y, into the set of objects, say X. It seems also reasonable to require that for each object x in X, there be a description y in Y such that $D(y) = x$. (Each object has a description.) To make descriptions useful we like them to be finite. This means that there are only countably many descriptions. Since there is a description for each object, there are also only countably many describable objects. How do we measure the complexity of descriptions?

M. Li and P.M.B. Vitányi, *An Introduction to Kolmogorov Complexity and Its Applications*,
DOI: 10.1007/978-0-387-49820-1_1, © Springer Science + Business Media, LLC 2008

Taking our cue from the theory of computation, we express descriptions as finite sequences of 0's and 1's. In communication technology, if the specification method D is known to both a sender and a receiver, then a message x can be transmitted from sender to receiver by transmitting the sequence of 0's and 1's of a description y with $D(y) = x$. The cost of this transmission is measured by the number of occurrences of 0's and 1's in y, that is, by the length of y. The least cost of transmission of x is given by the length of a shortest y such that $D(y) = x$. We choose this least cost of transmission as the descriptional complexity of x under specification method D.

Obviously, this descriptional complexity of x depends crucially on D. The general principle involved is that the syntactic framework of the description language determines the succinctness of description.

In order to objectively compare descriptional complexities of objects, to be able to say "x is more complex than z," the descriptional complexity of x should depend on x alone. This complexity can be viewed as related to a universal description method that is a priori assumed by all senders and receivers. This complexity is optimal if no other description method assigns a lower complexity to any object.

We are not really interested in optimality with respect to all description methods. For specifications to be useful at all, it is necessary that the mapping from y to $D(y)$ be executable in an effective manner. That is, it can at least in principle be performed by humans or machines. This notion has been formalized as that of partial recursive functions. According to generally accepted mathematical viewpoints it coincides with the intuitive notion of effective computation.

The set of partial recursive functions contains an optimal function that minimizes description length of every other such function. We denote this function by D_0. Namely, for any other recursive function D, for all objects x, there is a description y of x under D_0 that is shorter than any description z of x under D. (That is, shorter up to an additive constant that is independent of x.) Complexity with respect to D_0 minorizes the complexities with respect to all partial recursive functions.

We identify the length of the description of x with respect to a fixed specification function D_0 with the 'algorithmic (descriptional) complexity' of x. The optimality of D_0 in the sense above means that the complexity of an object x is invariant (up to an additive constant independent of x) under transition from one optimal specification function to another. Its complexity is an objective attribute of the described object alone: it is an intrinsic property of that object, and it does not depend on the description formalism. This complexity can be viewed as absolute information content: the amount of information that needs to be transmitted between all senders and receivers when they communicate the message

in absence of any other a priori knowledge that restricts the domain of the message.

Broadly speaking, this means that all description syntaxes that are powerful enough to express the partial recursive functions are approximately equally succinct. All algorithms can be expressed in each such programming language equally succinctly, up to a fixed additive constant term. The remarkable usefulness and inherent rightness of the theory of Kolmogorov complexity stems from this independence of the description method.

Thus, we have outlined the program for a general theory of algorithmic complexity. The four major innovations are as follows:

1. In restricting ourselves to formally effective descriptions, our definition covers every form of description that is intuitively acceptable as being effective according to general viewpoints in mathematics and logic.

2. The restriction to effective descriptions entails that there is a universal description method that minorizes the description length or complexity with respect to any other effective description method. This would not be the case if we considered, say, all noneffective description methods. Significantly, this implies Item 3.

3. The description length or complexity of an object is an intrinsic attribute of the object independent of the particular description method or formalizations thereof.

4. The disturbing Richard–Berry paradox above does not disappear, but resurfaces in the form of an alternative approach to proving Kurt Gödel's (1906–1978) famous result that not every true mathematical statement is provable in mathematics.

Example 1.1.1 **(Gödel's incompleteness result)** Gödel proved that in every consistent powerful enough theory, there are true but unprovable statements. He constructed such a statement. Here we use the incompressibility argument to show in a very simple manner that there are, in fact, infinitely many such undecidable statements.

A formal system (consisting of definitions, axioms, rules of inference) is *consistent* if no statement that can be expressed in the system can be proved to be both true and false in the system. A formal system is *sound* if only true statements can be proved to be true in the system. (Hence, a sound formal system is consistent.)

Let x be a finite binary string. We write "x is random" if the shortest binary description of x with respect to the optimal specification method

D_0 has length at least that of the literal description of x. A simple counting argument shows that there are random x's of each length.

Fix any sound formal system F in which we can express statements like "x is random." Suppose F can be described in f bits—assume, for example, that this is the number of bits used in the exhaustive description of F in the first chapter of the textbook *Foundations of F*. We claim that for all but finitely many random strings x, the sentence "x is random" is not provable in F. Assume the contrary. Then given F, we can start to search exhaustively for a proof that some string of length $n \gg f$ is random, and print it when we find such a string. This is an x satisfying the "x is random" sentence. This procedure to print x of length n uses only $\log n + f$ bits of data, where log denotes the binary logarithm, which is much less than n. But x is random by the proof, which is a true fact since F is sound, and hence its shortest effective description has binary length at least n. Hence, F is not consistent, which is a contradiction. ◇

This shows that although most strings are random, it is impossible to effectively prove them random. In a way, this explains why the incompressibility method in Chapter 6 is so successful. We can argue about a 'typical' individual element, which is difficult or impossible by other methods.

Example 1.1.2 **(Lower bounds)** The secret of the successful use of descriptional complexity arguments as a proof technique is due to a simple fact: the overwhelming majority of strings have almost no computable regularities. We have called such a string 'random.' There is no shorter description of such a string than the literal description: it is incompressible. Incompressibility is a noneffective property in the sense of Example 1.1.1.

Traditional proofs often involve all instances of a problem in order to conclude that some property holds for at least one instance. The proof would be simpler if only that one instance could have been used in the first place. Unfortunately, that instance is hard or impossible to find, and the proof has to involve all the instances. In contrast, in a proof by the incompressibility method, we first choose a random (that is, incompressible) individual object that is known to exist (even though we cannot construct it). Then we show that if the assumed property did not hold, then this object could be compressed, and hence it would not be random. Let us give a simple example.

A prime number is a natural number that is not divisible by natural numbers other than itself and 1. By the celebrated result of J.S. Hadamard (1865–1963) and C.J.G.N. de la Vallée Poussin (1866–1962) it is known that the number $\pi(n)$ of primes less than or equal to n satisfies $\pi(n) \sim n/\ln n$, where ln denotes the natural logarithm. For more

detailed results about $\pi(n)$ see Exercise 1.5.8 on page 17. We first prove a weak result due to G.J. Chaitin: For infinitely many n, the number of primes $\pi(n)$ satisfies

$$\pi(n) \geq \frac{\log n}{\log \log n} - o(1). \tag{1.1}$$

The proof method is as follows. For each n, we construct a description from which n can be effectively retrieved. This description will involve the primes less than n. For some n this description must be long, which shall give the desired result. Formally, assume that $p_1, p_2, \ldots, p_m$ is the list of all the primes less than n. Then, the integer number $n = p_1^{e_1} p_2^{e_2} \cdots p_m^{e_m}$ can be reconstructed from the vector of the exponents. Each exponent is at most $\log n$ and can be represented by $\log \log n$ bits. The description of n can be given in $m \log \log n$ bits provided we know the value $\log \log n$ enabling us to parse the constituent blocks of exponents. Thus, we prefix the description with a prefix-free code for $\log \log n$ in $(1 + o(1)) \log \log \log n$ bits. (Prefix codes and their lengths are described in Section 1.11.1.) It can be shown that for every integer $l > 0$ there is a natural number n of binary length $l \approx \log n$ that cannot be described in fewer than l bits (n is random), whence Equation 1.1 follows.

Can we do better? This is slightly more complicated. Let $l(x)$ denote the length of the binary representation of x. We shall show that for all n, the number of primes $\pi(n)$ satisfies

$$\pi(n) \geq \frac{1}{\epsilon(n)} \frac{n}{\ln n}, \tag{1.2}$$

where $\epsilon(n) = O((\log \log n)^{1+\epsilon})$ for an arbitrary small $\epsilon > 0$. We argue as follows: Every integer n can be described by the string $E(m)n/p_m$, where the binary string $E(m)$ is a prefix-free encoding of m, which is concatenated with the binary string representation of the integer n/p_m, and p_m is the largest prime dividing n. For random n, the length of this description, $l(E(m)) + \log n - \log p_m$, must exceed $\log n$. Therefore, $\log p_m < l(E(m))$. It is known (and easy) that we can set $l(E(m)) \leq \log m + \log \log m + \log \epsilon(m)$, Section 1.11.1. Hence, $p_m < n_m$ with $n_m = \epsilon(m) m \log m$. Since there are infinitely many primes (Equation 1.1), we have proven that for the special sequence of values of $n_1, n_2, \ldots$ the number of primes $\pi(n_m) \geq n_m/(\epsilon(n_m) \log n_m)$, for every $m \geq 1$. Let us denote $c_m = n_{m+1}/n_m$. For every n with $n_m \leq n \leq n_{m+1}$ we have $n_m \leq n \leq c_m n_m$, with $c_m \to 1$ for $m \to \infty$. Therefore, Equation 1.2 holds for all n if we absorb both c_m and $\log e$ in $\epsilon(n)$. (Note that $\log n = (\log e) \ln n$.) The idea of connecting primality and prefix code-word length is due to P. Berman, and the present proof is partially due to J.T. Tromp.

Chapter 6 introduces the incompressibility method. Its utility is demonstrated in a variety of examples of proving mathematical and computational results. These include questions concerning the average-case analysis of algorithms (such as Shellsort, Heapsort and routing), sequence analysis, formal languages, combinatorics, graphs, time and space complexity of machine models, language recognition, communication complexity, circuit complexity, and string matching. Other topics such as the use of resource-bounded Kolmogorov complexity in the analysis of computational complexity classes, the universal optimal search algorithm, and logical depth are treated in Chapter 7. ◇

Example 1.1.3 **(Prediction)** We are given an initial segment of an infinite sequence of zeros and ones. Our task is to predict the next element in the sequence: zero or one? The set of possible sequences we are dealing with constitutes the sample space, in this case, the set of one-way infinite binary sequences. We assume some orobability distribution μ over the sample space, where $\mu(x)$ is the probability of the initial segment of a sequence being x. Then the probability of the next bit being 0, after an initial segment x, is clearly $\mu(0|x) = \mu(x0)/\mu(x)$. This problem constitutes, perhaps, the central task of inductive reasoning and artificial intelligence. However, the problem of induction is that in general we do not know the distribution μ, preventing us from assessing the actual probability. Hence, we have to use an estimate.

Now assume that μ is computable. (This is not very restrictive, since any distribution used in statistics is computable, provided the parameters are computable.) We can use Kolmogorov complexity to give a very good estimate of μ. This involves the so-called universal distribution $\mathbf{M}$. Roughly speaking, $\mathbf{M}(x)$ is close to 2^{-l}, where l is the length in bits of the shortest effective description of x. The distribution $\mathbf{M}$ has the property that it assigns at least as high a probability to x as any computable μ (up to a multiplicative constant factor depending on μ but not on x). What is particularly important to prediction is the following:

Let S_n denote the μ-expectation of a particular form of the error we make in estimating the probability of the nth symbol by $\mathbf{M}$. Then it can be shown that the sum $\sum_n S_n$ is bounded by a constant. In other words, if S_n is smooth then it converges to zero faster than $1/n$. Consequently, any actual (computable) distribution can be estimated and predicted with great accuracy using only the single universal distribution.

Among other things, Chapter 5 develops a general theory of inductive reasoning and applies the notions introduced to particular problems of inductive inference, prediction, mistake bounds, computational learning theory, and minimum description length induction methods in statistics. In particular, it is demonstrated that data compression improves generalization and prediction performance. ◇

The purpose of the remainder of this chapter is to define several concepts we require, if not by way of introduction, then at least to establish notation.

1.2 Prerequisites and Notation

We usually deal with nonnegative integers, sets of nonnegative integers, and mappings from nonnegative integers to nonnegative integers. A, B, $C, \ldots$ denote sets. $\mathcal{N}$, $\mathcal{Z}$, $\mathcal{Q}$, $\mathcal{R}$ denote the sets of nonnegative integers (natural numbers including zero), integers, rational numbers, and real numbers, respectively. For each such set A, by A^+ we denote the subset of A consisting of positive numbers.

We use the following set-theoretic notation: $x \in A$ means that x is a member of A. In $\{x : x \in A\}$, the symbol : denotes set formation. $A \bigcup B$ is the *union* of A and B, $A \bigcap B$ is the *intersection* of A and B, and $\bar{A}$ is the *complement* of A when the universe $A \bigcup \bar{A}$ is understood. $A \subseteq B$ means A is a *subset* of B; $A = B$ means A and B are *identical* as sets (have the same members).

The *cardinality* (or diameter) of a finite set A is the number of elements it contains and is denoted by $d(A)$. If $A = \{a_1, \ldots, a_n\}$, then $d(A) = n$. The *empty* set $\{\}$, with no elements in it, is denoted by $\varnothing$. In particular, $d(\varnothing) = 0$.

Given x and y, the ordered pair (x, y) consists of x and y in that order. $A \times B$ is the *Cartesian product* of A and B, the set $\{(x, y) : x \in A \text{ and } y \in B\}$. The n-fold Cartesian product of A with itself is denoted by A^n. If $R \subseteq A^2$, then R is called a *binary relation*. The same definitions can be given for n-tuples, $n > 2$, and the corresponding relations are *n-ary*. We say that an n-ary relation R is *single-valued* if for every $(x_1, \ldots, x_{n-1})$ there is at most one y such that $(x_1, \ldots, x_{n-1}, y) \in R$. Consider the *domain* $\{(x_1, \ldots, x_{n-1}) : \text{there is a } y \text{ such that } (x_1, \ldots, x_{n-1}, y) \in R\}$ of a single-valued relation R. Clearly, a single-valued relation $R \subseteq A^{n-1} \times B$ can be considered as a mapping from its domain into B. Therefore, we also call a single-valued n-ary relation a *partial function* of $n - 1$ variables ('partial' because the domain of R may not comprise all of A^{n-1}). We denote functions by $\phi, \psi, \ldots$ or $f, g, h, \ldots$. Functions defined on the n-fold Cartesian product A^n are denoted with possibly a superscript denoting the number of variables, like $\phi^{(n)} = \phi^{(n)}(x_1, \ldots, x_n)$.

We use the notation $\langle \cdot \rangle$ for some standard one-to-one encoding of $\mathcal{N}^n$ into $\mathcal{N}$. We will use $\langle \cdot \rangle$ especially as a *pairing function* over $\mathcal{N}$ to associate a unique natural number $\langle x, y \rangle$ with each pair (x, y) of natural numbers. An example is $\langle x, y \rangle$ defined by $y + (x + y + 1)(x + y)/2$. This mapping can be used recursively: $\langle x, y, z \rangle = \langle x, \langle y, z \rangle \rangle$.

If ϕ is a *partial function* from A to B, then for each $x \in A$ either $\phi(x) \in B$ or $\phi(x)$ is undefined. If x is a member of the domain of ϕ,

then $\phi(x)$ is called a *value* of ϕ, and we write $\phi(x) < \infty$ and ϕ is called *convergent* or *defined* at x; otherwise we write $\phi(x) = \infty$ and we call ϕ *divergent* or *undefined* at x. The set of values of ϕ is called the *range* of ϕ. If ϕ converges at every member of A, it is a *total function*, otherwise a *strictly partial function*. If each member of a set B is also a value of ϕ, then ϕ is said to *map onto* B, otherwise to *map into* B. If for each pair x and y, $x \neq y$, for which ϕ converges $\phi(x) \neq \phi(y)$ holds, then ϕ is a *one-to-one mapping*, otherwise a *many-to-one mapping*. The function $f : A \to \{0, 1\}$ defined by $f(x) = 1$ if $\phi(x)$ converges, and $f(x) = 0$ otherwise, is called the *characteristic function* of the domain of ϕ.

If ϕ and ψ are two partial functions, then $\psi\phi$ (equivalently, $\psi(\phi(x))$) denotes their *composition*, the function defined by $\{(x, y) :$ there is a z such that $\phi(x) = z$ and $\psi(z) = y\}$. The *inverse* ϕ^{-1} of a one-to-one partial function ϕ is defined by $\phi^{-1}(y) = x$ iff $\phi(x) = y$.

A set A is called *countable* if it is either empty or there is a total one-to-one mapping from A to the natural numbers $\mathcal{N}$. We say A is *countably infinite* if it is both countable and infinite. By 2^A we denote the *set of all subsets* of A. The set $2^{\mathcal{N}}$ has the cardinality of the *continuum* and is therefore uncountably infinite.

For binary relations, we use the terms *reflexive*, *transitive*, *symmetric, equivalence, partial order,* and *linear* (or *total*) *order* in the usual meaning. Partial orders can be *strict* ($<$) or *nonstrict* ($\leq$).

If we use the logarithm notation $\log x$ without subscript, then we shall always mean base 2. By $\ln x$ we mean the *natural logarithm* $\log_e x$, where $e = 2.71\ldots$.

We use the quantifiers $\exists$ (there exists), $\forall$ (for all), $\exists^\infty$ (there exist infinitely many), and the awkward $\forall^\infty$ (for all but finitely many). In this way, $\forall^\infty x[\phi(x)]$ iff $\neg\exists^\infty x[\neg\phi(x)]$.

1.3 Numbers and Combinatorics

The *absolute value* of a real number r is denoted by $|r|$ and is defined as $|r| = -r$ if $r < 0$ and r otherwise. The *floor* of a real number r, denoted by $\lfloor r \rfloor$, is the greatest integer n such that $n \leq r$. Analogously, the *ceiling* of a real number r, denoted by $\lceil r \rceil$, is the least integer n such that $n \geq r$.

Example 1.3.1 $|-1| = |1| = 1$. $\lfloor 0.5 \rfloor = 0$ and $\lceil 0.5 \rceil = 1$. Analogously, $\lfloor -0.5 \rfloor = -1$ and $\lceil -0.5 \rceil = 0$. But $\lfloor 2 \rfloor = \lceil 2 \rceil = 2$ and $\lfloor -2 \rfloor = \lceil -2 \rceil = -2$. ◇

A *permutation* of n objects is an arrangement of n distinct objects in an ordered sequence. For example, the six different permutations of objects a, b, c are

$$abc, acb, bac, bca, cab, cba.$$

The number of permutations of n objects is found most easily by imagining a sequential process to choose a permutation. There are n choices of which object to place in the first position; after filling the first position there remain $n-1$ objects and therefore $n-1$ choices of which object to place in the second position, and so on. Therefore, the number of permutations of n objects is $n \times (n-1) \times \cdots \times 2 \times 1$, denoted by $n!$ and referred to as n *factorial*. In particular, $0! = 1$.

A *variation* of k out of n objects is an arrangement consisting of the first k elements of a permutation of n objects. For example, the twelve variations of two out of four objects a, b, c, d are

$$ab, ac, ad, ba, bc, bd, ca, cb, cd, da, db, dc.$$

The number of variations of k out of n is $n!/(n-k)!$, as follows by the previous argument. While there is no accepted standard notation, we denote the number of variations by $(n)_k$. In particular, $(n)_0 = 1$.

The *combinations* of n objects taken k at a time (n choose k) are the possible choices of k different elements from a collection of n objects. The six different combinations of two out of four objects a, b, c, d are

$$\{a, b\}, \{a, c\}, \{a, d\}, \{b, c\}, \{b, d\}, \{c, d\}.$$

We can consider a combination as a variation in which the order does not count. We have seen that there are $n(n-1)\cdots(n-k+1)$ ways to choose the first k elements of a permutation. Every k-combination appears precisely $k!$ times in these arrangements, since each combination occurs in all its permutations. Therefore, the number of combinations, denoted by $\binom{n}{k}$, is

$$\binom{n}{k} = \frac{n(n-1)\cdots(n-k+1)}{k(k-1)\cdots(1)}.$$

In particular, $\binom{n}{0} = 1$. The quantity $\binom{n}{k}$ is also called a *binomial coefficient*. It has an extraordinary number of applications. Perhaps the foremost relation associated with it is the binomial theorem, discovered in 1676 by Isaac Newton:

$$(x+y)^n = \sum_k \binom{n}{k} x^k y^{n-k},$$

with n a positive integer. Note that in the summation k need not be restricted to $0 \le k \le n$, but can range over $-\infty < k < +\infty$, since for $k < 0$ or $k > n$ the terms are all zero.

Example 1.3.2 An important relation following from the binomial theorem is found by substituting $y = 1$:

$$(x+1)^n = \sum_k \binom{n}{k} x^k.$$

Substituting also $x = 1$, we obtain

$$2^n = \sum_k \binom{n}{k}.$$

◇

Exercises

1.3.1. [12] Consider a random distribution of k distinguishable balls in n cells, that is, each of the n^k possible arrangements has probability n^{-k}. Show that the probability P_i that a specified cell contains exactly i balls ($0 \le i \le k$) is given by $P_i = \binom{k}{i}(1/n)^i(1-1/n)^{k-i}$.

Comments. Source: W. Feller, *An Introduction to Probability Theory and Its Applications*, Vol. 1, Wiley, 1968.

1.3.2. [08] Show that $\binom{n}{k} = \frac{(n)_k}{k!}$ and $\binom{n}{k} = \binom{n}{n-k}$.

1.3.3. [M34] Prove the following identity, which is very useful in the sequel of this book: Up to a fixed additive constant we have

$$\log \binom{n}{k} = k \log \frac{n}{k} + (n-k) \log \frac{n}{n-k} + \frac{1}{2} \log \frac{n}{k(n-k)}.$$

1.3.4. [15] (a) Prove that the number of ways n distinguishable balls can be placed in k numbered cells such that the first cell contains n_1 balls, the second cell n_2 balls, up to the kth cell contains n_k balls with $n_1 + \cdots + n_k = n$ is

$$\binom{n}{n_1, \ldots, n_k} = \frac{n!}{n_1! \cdots n_k!}.$$

This number is called a *multinomial coefficient*. Note that the order of the cells is essential in that the partitions $(n_1 = 1, n_2 = 2)$ and $(n_1 = 2, n_2 = 1)$ are different. The order of the elements within a cell is irrelevant.

(b) Show that

$$(x_1 + \cdots + x_k)^n = \sum \binom{n}{n_1, \ldots, n_k} x_1^{n_1} \cdots x_k^{n_k},$$

with the sum taken for all $n_1 + \cdots + n_k = n$.

(c) The *number of ordered different partitions* of n in r nonnegative integral summands is denoted by $A_{n,r}$. Compute $A_{n,r}$ in the form of a binomial coefficient.

Comments. $(1, 0)$ and $(0, 1)$ are different partitions, so $A_{1,2} = 2$. Source: W. Feller, *Ibid.*

1.3.5. [14] Define the *occupancy numbers* for n balls distributed over k cells as a k-tuple of integers $(n_1, n_2, \ldots, n_k)$ satisfying $n_1+n_2+\cdots+n_k = n$ with $n_i \geq 0$ $(1 \leq i \leq k)$. That is, the first cell contains n_1 balls, the second cell n_2 balls, and so on.

(a) Show that there are $\binom{n}{n_1,\ldots,n_k}$ placements of n balls in k cells resulting in the numbers $(n_1, \ldots, n_k)$.

(b) There are k^n possible placements of n balls in k cells altogether. Compute the fraction that results in the given occupancy numbers.

(c) Assume that all k^n possible placements of n balls in k cells are equally probable. Conclude that the probability of obtaining the given occupancy numbers is

$$\frac{n!}{n_1! \cdots n_k!} k^{-n}.$$

Comments. In physics this is known as the *Maxwell–Boltzmann statistics* (here 'statistics' is used as a synonym for 'distribution'). Source: W. Feller, *Ibid.*

1.3.6. [15] We continue with the previous exercise. In physical situations the assumption of equiprobability of possible placements seems unavoidable, for example, molecules in a volume of gas divided into (hypothetical) cells of equal volume. Numerous attempts have been made to prove that physical particles behave in accordance with the Maxwell–Boltzmann distribution. However, it has been shown conclusively that *no known particles* behave according to this distribution.

(a) In the *Bose–Einstein* distribution we count only *distinguishable* distributions of n balls over k cells without regard for the identities of the balls. We are interested only in the number of solutions of $n_1+n_2+\cdots+n_k = n$. Show that this number is $\binom{k+n-1}{n} = \binom{k+n-1}{k-1}$. Conclude that the probability of obtaining each given occupancy number is equally $1/\binom{k+n-1}{k-1}$. (Illustration: the distinguishable distributions of two balls over two cells are |**, *|*, and **|. According to Bose–Einstein statistics there are only three possible outcomes for two coin flips: head–head, head–tail, and tail–tail, and each outcome has equal probability $\frac{1}{3}$.)

(b) In the *Fermi–Dirac* distribution, (1) two or more particles cannot occupy the same cell and (2) all distinguishable arrangements satisfying (1) have the same probability. Note that (1) requires $n \leq k$. Prove that in the Fermi–Dirac distribution there are in total $\binom{k}{n}$ possible arrangements. Conclude that the probability for each possible occupancy number is equally $1/\binom{k}{n}$.

Comments. According to modern physics, photons, nuclei, and atoms containing an even number of elementary particles behave according to

model (a), and electrons, neutrons, and protons behave according to model (b). This shows that nature does not necessarily satisfy our a priori assumptions, however plausible they may be. Source: W. Feller, *Ibid.*

1.4 Binary Strings

We are concerned with *strings* over a nonempty set $\mathcal{B}$ of *basic elements*. Unless otherwise noted, we use $\mathcal{B} = \{0, 1\}$. Instead of 'string' we also use 'word' and 'sequence' synonymously. The way we use it, 'strings' and 'words' are usually finite, while 'sequences' are usually infinite. The set of all finite strings over $\mathcal{B}$ is denoted by $\mathcal{B}^*$, defined as

$$\mathcal{B}^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, \ldots\},$$

with ϵ denoting the *empty* string, with no letters. *Concatenation* is a binary operation on the elements of $\mathcal{B}^*$ that associates xy with each ordered pair of elements (x, y) in the Cartesian product $\mathcal{B}^* \times \mathcal{B}^*$. Clearly,

1. $\mathcal{B}^*$ is closed under the operation of concatenation; that is, if x and y are elements of $\mathcal{B}^*$, then so is xy;
2. concatenation is an associative operation on $\mathcal{B}^*$; that is, $(xy)z = x(yz) = xyz$; and
3. concatenation on $\mathcal{B}^*$ has the *unit* element ϵ; that is, $\epsilon x = x\epsilon = x$.

We now consider a *correspondence* of finite binary strings and natural numbers. The standard binary representation has the disadvantage that either some strings do not represent a natural number, or each natural number is represented by more than one string. For example, either 010 does not represent 2, or both 010 and 10 represent 2. We can map $\mathcal{B}^*$ one-to-one onto the natural numbers by associating each string with its index in the length-increasing lexicographic ordering

$$(\epsilon, 0), (0, 1), (1, 2), (00, 3), (01, 4), (10, 5), (11, 6), \ldots. \tag{1.3}$$

In this way we represent $x = 2^{n+1} - 1 + \sum_{i=0}^{n} a_i 2^i$ by $a_n \ldots a_1 a_0$. This is equivalent to $x = \sum_{i=0}^{n} b_i 2^i$ with $b_i \in \{1, 2\}$ and $b_i = a_i + 1$ for $0 \le i \le n$.

The binary representation for the natural numbers given in Equation 1.3 is different from the standard binary representation. It is convenient not to distinguish between the first and second elements of the same pair, and call them 'string' or 'number' arbitrarily. That is, we consider both the string 01 and the natural number 4 as the same object. For example, we may write $01 = 4$. We denote these objects in general with lowercase roman letters. A string consisting of n zeros is denoted by 0^n.

If x is a string of n 0's and 1's, then x_i denotes the ith *bit* (binary digit) of x for all i, $1 \leq i \leq n$, and $x_{i:j}$ denotes the $(j-i+1)$-bit segment $x_i x_{i+1} \ldots x_j$. For $x = 1010$ we have $x_1 = x_3 = 1$ and $x_2 = x_4 = 0$; for $x = x_1 x_2 \ldots x_n$ we have $x_{1:i} = x_1 x_2 \ldots x_i$. The *reverse,* x^R, of a string $x = x_1 x_2 \ldots x_n$ is $x_n x_{n-1} \ldots x_1$.

The *length* of a finite binary string x is the number of bits it contains and is denoted by $l(x)$. If $x = x_1 x_2 \ldots x_n$, then $l(x) = n$. In particular, $l(\epsilon) = 0$.

Thus, $l(xy) = l(x) + l(y)$, and $l(x^R) = l(x)$. Recall that we use the above pairing of binary strings and natural numbers. Thus, $l(4) = 2$ and $01 = 4$. The *number of elements* (*cardinality*) in a finite set A is denoted by $d(A)$. Therefore, $d(\{u : l(u) = n\}) = 2^n$ and $d(\{u : l(u) \leq n\}) = 2^{n+1} - 1$.

Let D be any function $D : \{0,1\}^* \to \mathcal{N}$. Considering the domain of D as the set of *code words*, and the range of D as the set of *source words*, $D(y) = x$ is interpreted as 'y is a code word for the source word x, and D is the *decoding* function.' (In the introduction we called D a specification method.) The set of all code words for source word x is the set $D^{-1}(x) = \{y : D(y) = x\}$. Hence, $E = D^{-1}$ can be called the *encoding* substitution (E is not necessarily a function). Let $x, y \in \{0,1\}^*$. We call x a *prefix* of y if there is a z such that $y = xz$. A set $A \subseteq \{0,1\}^*$ is *prefix-free* if no element in A is the prefix of another element in A. A function $D : \{0,1\}^* \to \mathcal{N}$ defines a *prefix-code* if its domain is prefix-free. (Coding theory is treated in Section 1.11.1.) A simple prefix-code we use throughout is obtained by reserving one symbol, say 0, as a stop sign and encoding $x \in \mathcal{N}$ as $1^x 0$. We can prefix an object with its length and iterate this idea to obtain ever shorter codes:

$$E_i(x) = \begin{cases} 1^x 0 & \text{for } i = 0, \\ E_{i-1}(l(x))x & \text{for } i > 0. \end{cases} \tag{1.4}$$

Thus, $E_1(x) = 1^{l(x)}0x$ and has length $l(E_1(x)) = 2l(x)+1$. This encoding is sufficiently important to have a simpler notation:

$$\bar{x} = 1^{l(x)}0x,$$
$$l(\bar{x}) = 2l(x) + 1.$$

Sometimes we need the shorter prefix-code $E_2(x)$,

$$E_2(x) = \overline{l(x)}x,$$
$$l(E_2(x)) = l(x) + 2l(l(x)) + 1.$$

We call $\bar{x}$ the *self-delimiting* version of the binary string x. Now we can effectively recover both x and y unambiguously from the binary string $\bar{x}y$. If $\bar{x}y = 111011011$, then $x = 110$ and $y = 11$. If $\bar{x}\bar{y} = 1110110101$, then $x = 110$ and $y = 1$.

Example 1.4.1 It is convenient to consider also the set of *one-way infinite* sequences $\mathcal{B}^\infty$. If ω is an element of $\mathcal{B}^\infty$, then $\omega = \omega_1\omega_2\ldots$ and $\omega_{1:n} = \omega_1\omega_2\ldots\omega_n$. The set of infinite sequences of elements from a finite, nonempty basic set $\mathcal{B}$ corresponds with the set $\mathcal{R}$ of real numbers in the following way: Let $\mathcal{B} = \{0, 1, \ldots, k-1\}$ with $k \geq 2$. If r is a real number $0 < r < 1$ then there is a sequence $\omega_1\omega_2\ldots$ of elements $\omega_n \in \mathcal{B}$ such that

$$r = \sum_n \frac{\omega_n}{k^n},$$

and that sequence is unique except when r is of the form q/k^n, in which case there are exactly two such sequences, one of which has infinitely many 0's. Conversely, if $\omega_1\omega_2\ldots$ is an infinite sequence of integers with $0 \leq \omega_n < k$, then the series

$$\sum_n \frac{\omega_n}{k^n}$$

converges to a real number r with $0 \leq r \leq 1$. This sequence is called the *k-ary* expansion of r. In the following we identify a real number r with its k-ary expansion (if there are two k-ary expansions, then we identify r with the expansion with infinitely many 0's).

Define the set $S \subseteq \mathcal{B}^\infty$ as the set of sequences that do not end with infinitely many digits '$k-1$.' Then, S is in one-to-one correspondence with the set of real numbers in the interval $[0, 1)$.

Let x be a finite string over $\mathcal{B}$. The set of all one-way infinite sequences starting with x is called a *cylinder* and is denoted by Γ_x and is defined by $\Gamma_x = \{x\omega : \omega \in \mathcal{B}^\infty\}$ with $x \in \mathcal{B}^*$. Geometrically speaking, the cylinder Γ_x can be identified with the *half-open interval* $[0.x, 0.x + k^{-l(x)})$ in the real interval $[0, 1)$. Observe that the usual geometric length of interval Γ_x equals $k^{-l(x)}$. Furthermore, $\Gamma_y \subseteq \Gamma_x$ iff x is a prefix of y. The prefix relation induces a partial order on the cylinders of $\mathcal{B}^\infty$. ◇

Exercises

1.4.1. [03] If $\bar{x}\bar{y}z = 10010111$, what are x, y, z in decimal numbers?

Comments. 1, 2, 6.

1.4.2. [07] (a) Show that for $x \in \mathcal{N}$ we have $l(x) = \lfloor \log(x+1) \rfloor$.

(b) Give another code $c(x)$ for the natural numbers $x = 1, 2, \ldots$ such that $l(c(x)) = \lfloor \log x \rfloor$.

Comments. Hint for Item (b): use the correspondence

$$(\epsilon, 1), (0, 2), (1, 3), (00, 4), (01, 5), (10, 6), (11, 7), \ldots.$$

1.4.3. [10] Let $E : \mathcal{N} \to \{0,1\}^*$ be a total one-to-one function whose range is prefix-free. E defines a prefix-code. Define the mapping $\langle\cdot\rangle : \mathcal{N} \times \mathcal{N} \to \mathcal{N}$ by $\langle x, y\rangle = E(x)y$.

(a) Show that $\langle\cdot\rangle$ is total and one-to-one.

(b) Show that we can extend this scheme to k-tuples $(n_1, n_2, \ldots, n_k)$ of natural numbers to obtain a total one-to-one mapping from $\mathcal{N} \times \mathcal{N} \times \cdots \times \mathcal{N}$ into $\mathcal{N}$.

Comments. Define the mapping for (x, y, z) as $\langle x, \langle y, z\rangle\rangle$, and iterate this construction.

1.4.4. [10] Let E be as above. Define the mapping $\langle\cdot\rangle : \mathcal{N} \times \mathcal{N} \to \mathcal{N}$ by $\langle x, y\rangle = E(x)E(y)$.

(a) Show that $\langle\cdot\rangle$ is a total one-to-one mapping and a prefix-code.

(b) Show that we can extend this scheme to k-tuples $(n_1, n_2, \ldots, n_k)$ of natural numbers to obtain a total one-to-one mapping from $\mathcal{N} \times \mathcal{N} \times \cdots \times \mathcal{N}$ into $\mathcal{N}$ that is a prefix-code.

Comments. Define the mapping for (x, y, z) as $\langle x, \langle y, z\rangle\rangle$ and iterate this construction. Another way is to map $(x, y, \ldots, z)$ to $E(x)E(y)\ldots E(z)$.

1.4.5. [10] (a) Show that $E(x) = \bar{x}$ is a prefix-code.

(b) Consider a variant of the $\bar{x}$ code such that $x = x_1x_2\ldots x_n$ is encoded as $x_1 1 x_2 1 \ldots 1 x_{n-1} 1 x_n 0$. Show that this is a prefix-code for the binary nonempty strings with $l(\bar{x}) = 2l(x)$.

(c) Consider $x = x_1x_2\ldots x_n$ encoded as $x_1x_1x_2x_2\ldots x_{n-1}x_{n-1}x_n\neg x_n$. Show that this is a prefix-code for the nonempty binary strings.

(d) Give a prefix-code $\tilde{x}$ for the set of all binary strings x including ϵ, such that $l(\tilde{x}) = 2l(x) + 2$.

1.5 Asymptotic Notation

It is often convenient to express approximate equality or inequality of one quantity with another. If f and g are functions of a real variable, then it is customary to denote $\lim_{n\to\infty} f(n)/g(n) = 1$ by $f(n) \sim g(n)$, and we write 'f goes asymptotically to g.'

P. Bachman introduced a convenient notation for dealing with approximations in his book *Analytische Zahlentheorie* in 1892. This big-O notation allows us to write $l(x) = \log x + O(1)$ (no subscript on the logarithm means base 2).

We use the notation $O(f(n))$ whenever we want to denote a quantity that does not exceed $f(n)$ by more than a fixed multiplicative factor. This is useful in case we want to simplify the expression involving this

quantity by suppressing unnecessary detail, but also in case we do not know this quantity explicitly. Bachman's notation is the first of a family of *order of magnitude* symbols: O, o, Ω, and Θ. If f and g are functions on the real numbers, then

1. $f(x) = O(g(x))$ if there are constants $c, x_0 > 0$ such that $|f(x)| \leq c|g(x)|$, for all $x \geq x_0$;
2. $f(x) = o(g(x))$ if $\lim_{x\to\infty} f(x)/g(x) = 0$;
3. $f(x) = \Omega(g(x))$ if $f(x) \neq o(g(x))$; and
4. $f(x) = \Theta(g(x))$ if both $f(x) = O(g(x))$ and $f(x) = \Omega(g(x))$.

It is straightforward to extend these definitions to functions of more variables. For example, $f(x, y) = O(g(x, y))$ if there are positive constants c, x_0, y_0 such that $|f(x, y)| \leq c|g(x, y)|$, for all $x \geq x_0, y \geq y_0$. The definitions are standard, except Item 3, and thereby Item 4, which involves Item 3. This definition of Ω was introduced first by G.H. Hardy and J.E. Littlewood in 1914 and is the one commonly used in mathematics. It has the advantage that Ω is the complement of o. This is not the case with the definition proposed by D.E. Knuth in 1976, which is often referred to in computer science. Namely, Knuth defines $f(x) = \Omega(g(x))$ if there is a constant $c > 0$ such that $|f(x)| \geq c|g(x)|$ from some x onward. We have defined $f(x) = \Omega(g(x))$ if there is a constant $c > 0$ such that $|f(x)| \geq c|g(x)|$ infinitely often. This use of Ω should not be confused with Chaitin's mystery number Ω, which we encounter in Section 3.6.2.

Example 1.5.1 The definition of the big-O notation contains some mysterious absolute value signs. This becomes understandable if we realize that one wants to use a term like $O(f(x))$ to bound the absolute value of an error term, be it positive or negative, for example, as in

$$x^2 + x \sin x = x^2 + O(x).$$

This avoids the clumsy notation $\pm O(f(x))$ one would have been forced to use otherwise. ◇

Example 1.5.2 If $f(n) \sim g(n)$, then $f(n) = \Theta(g(n))$, but the converse implication does not hold. For instance, we have $2x = \Theta(x)$, but $2x \sim x$ does not hold. On the other hand, $-x = \Theta(x)$. ◇

Example 1.5.3 We can use O-notation to speak generically about mth-degree polynomials, for instance, $1+2+\cdots+n = n(n+1)/2$. Then $1+2+\cdots+n = O(n^2)$, but also $1 + 2 + \cdots + n = n^2/2 + O(n)$. The latter approximation is obviously a stronger statement than the first approximation. Similarly, if $p(n)$ is a polynomial of degree m, then $p(n) = O(n^m)$ and $p(n) = \Theta(n^m)$. ◇

Exercises

1.5.1. [07] Show that $x = o(x^2)$; $\sin x = O(1)$; $x^{-1/2} = o(1)$; $x + x^2 \sim x^2$, and $\sum_{k=1}^{n} kn = O(n^3)$.

1.5.2. [10] Show that $f(n) = O(f(n))$; $c \cdot O(f(n)) = O(f(n))$ if c is a constant; $O(f(n) + O(f(n)) = O(f(n))$; $O(O(f(n))) = O(f(n))$; $O(f(n))O(g(n)) = O(f(n)g(n))$; $O(f(n)g(n)) = f(n)O(g(n))$.

1.5.3. [10] Show that $f(n) = o(g(n))$ implies $f(n) = \Omega(g(n))$, but not vice versa.

1.5.4. [M30] It is natural to wonder how large 100! is approximately, without carrying out the multiplications implied by the definition. Prove the approximation $n! \sim \sqrt{2\pi n}(n/e)^n$.

Comments. This celebrated approximation of the factorial function was found by James Stirling [*Methodus Differentialis* (1730), 137].

1.5.5. [15] Denote the Hardy–Littlewood version of Ω we gave in the main text by Ω_H and the Knuth version by Ω_K.

(a) Show that for function f defined by $\log\log f(n) = \lfloor \log\log n \rfloor$, we have $f(n) = \Omega_H(n)$ but for no $\epsilon > 0$ does $f(n) = \Omega_K(n^{1/2+\epsilon})$ hold.

(b) Show that nonetheless, while $f(n) = O(n)$, for no $\epsilon > 0$ do we have $f(n) = O(n^{1-\epsilon})$.

Comments. Source: P.M.B. Vitányi and L.G.L.T. Meertens, *SIGACT News*, 16:4(1985), 56–59.

1.5.6. [20] It is well known that $n^{1/n} \to 1$ for $n \to \infty$.

(a) Show that $n^{1/n} = e^{\ln n/n} = 1 + (\ln n/n) + O((\ln n/n)^2)$.

(b) Use (a) to show that $\lim_{n\to\infty} n(n^{1/n} - 1) = \ln n$.

1.5.7. [15] (a) Show that $f(n) \neq \Omega(g(n))$ iff $f(n) = o(g(n))$.

(b) Show that $f(n) = \Theta(g(n))$ or $f(n) = o(g(n))$ iff $f(n) = O(g(n))$.

(c) Show that $f(n) \neq O(g(n))$ iff $f(n) = \Omega(g(n))$ and $f(n) \neq \Theta(g(n))$.

1.5.8. [HM45] Let $\pi(n)$ denote the number of primes that do not exceed n. Show that

(a) a crude approximation is $\pi(n) \sim n/\ln n$;

(b) a better approximation is

$$\pi(n) = \frac{n}{\ln n} + \frac{n}{(\ln n)^2} + \frac{2!n}{(\ln n)^3} + \frac{3!n}{(\ln n)^4} + O\left(\frac{n}{(\ln n)^5}\right).$$

Comments. The displayed formulas are called *prime number theorems.* Source: R. Graham, D.E. Knuth, O. Patashnik, *Concrete Mathematics*, Addison-Wesley, 1989.

1.6 Basics of Probability Theory

It is useful to recall briefly the basic notions of probability theory. The calculus of probabilities studies mathematical models of situations (experiments, observations) in which the *outcome* is not deterministic but is determined by uncertain circumstances. The set of all possible outcomes is called the *sample space*, usually denoted by S, and an *event* E is a subset of S. The sample space S can be *countable*, which means that it is finite or countably infinite, or *continuous*, which means that it is uncountably infinite.

Example 1.6.1 The throwing of two dice, one white and one black, gives a sample space S consisting of all pairs (i, j) where i is the number on the top face of the white die and j is the number on the top face of the black die. If $A = \{(1,3),(2,2),(3,1)\}$, then A is the event that the sum of i and j is four. If $B = \{(1,1),(1,2),(2,1)\}$, then B is the event that the sum of i and j is less than 4. ◇

Intuitively, the probability p of an event A is the apparent limit of the relative frequency of outcomes in A in the long run in a sequence of independent repetitions of the experiment. For instance, the probability associated with A in the example is $\frac{1}{12}$.

1.6.1 Kolmogorov Axioms

Let S denote the sample space. Following A.N. Kolmogorov's formalization of 1933, it is customary to use the following axioms.

(A1) If A and B are events, then so is the *intersection* $A \bigcap B$, the *union* $A \bigcup B$, and the *difference* $A - B$.

(A2) The *sample space* S is an event. We call S the *certain* event. The *empty set*, denoted by $\varnothing$, is an event. We call $\varnothing$ the *impossible* event.

(A3) To each event E is assigned a nonnegative real number $P(E)$ that we call the *probability* of event E.

(A4) $P(S) = 1$.

(A5) If A and B are disjoint, then $P(A \bigcup B) = P(A) + P(B)$.

(A6) For a decreasing sequence $A_1 \supset A_2 \supset \cdots \supset A_n \supset \cdots$ of events with $\bigcap_n A_n = \varnothing$ we have $\lim_{n\to\infty} P(A_n) = 0$.

For systems with *finitely* many events Axiom A6 clearly follows from Axioms A1 through A5. For systems with *infinitely* many events, however, it is independent of the first five axioms. Therefore, Axiom A6 is essential only for systems with infinitely many events.

A system $\mathcal{F}$ of sets $f \subseteq S$ that is closed under the binary operations union, intersection, difference and contains a 1-element (here S) and a 0-element (here $\varnothing$) is called a (set) *field*. The set of events $\mathcal{F}$ together with the associated set function P, also called a *measure* on $\mathcal{F}$, is called a *probability field* and is denoted by $(\mathcal{F}, P)$. It is easy to show that the axiom system A1 through A6 is *consistent* (free from contradictions). This is ascertained in the usual way by constructing an example that satisfies the axioms. Let S consist of a single element, and let the set of events be S and the empty event $\varnothing$, and set $P(S) = 1$ and $P(\varnothing) = 0$. It is easy to verify that the system defined in this way satisfies all the axioms above. However, the set of axioms is *incomplete*: for different problems in probability theory we have to construct different probability fields.

Example 1.6.2 We call P a *probability distribution* over S. It follows from the axioms that $0 \leq P(E) \leq 1$ for every event E; $P(\varnothing) = 0$; if $A \subseteq B$, then $P(A) \leq P(B)$; if $\bar{A}$ is the complement $S - A$ of A, then $P(\bar{A}) = 1 - P(A)$; and $P(A \bigcup B) = P(A) + P(B) - P(A \bigcap B)$. $\diamond$

1.6.2 Conditional Probability

If A and B are two events, with $P(A) > 0$, then the *conditional probability* that B occurs given that A occurs is defined by

$$P(B|A) = \frac{P(A \bigcap B)}{P(A)}.$$

It follows immediately that $P(A \bigcap B) = P(A)P(B|A)$, and by induction we obtain the *multiplication rule*:

$$P\left(A \bigcap B \bigcap \cdots \bigcap N\right) = P(A)P(B|A) \cdots P\left(N|A \bigcap B \bigcap \cdots \bigcap M\right).$$

Consider the setup of Example 1.6.1 again. Let A be the event that at least one out of two dice shows an even number, and let B be the event that the sum of the numbers shown is even. Then $P(B|A) = P(A \bigcap B)/P(A) = \frac{1}{3}$.

The function $P(\cdot|A)$ is a probability distribution on S and is called the *conditional probability distribution* given A. Clearly, $P(A|A) = 1$.

Example 1.6.3 Rewriting $P(A \bigcap B)$ as $P(B)P(A|B)$ and as $P(A)P(B|A)$, substitution yields the formula that incorporates the essence of Bayes's rule,

$$P(A|B) = \frac{P(A)P(B|A)}{P(B)}.$$

We require the *law of complete probabilities*. Let $A \bigcup B \bigcup \cdots \bigcup N = S$ for disjoint events $A, B, \ldots, N$, and let X be an arbitrary event. Then

$$P(X) = P(A)P(X|A) + P(B)P(X|B) + \cdots + P(N)P(X|N).$$

If $A, B, \ldots, N$ are disjoint and X is arbitrary, it is easy to derive from the previous two displayed formulas the important *Bayes's rule:*

$$P(Y|X) = \frac{P(Y)P(X|Y)}{P(A)P(X|A) + P(B)P(X|B) + \cdots + P(N)P(X|N)},$$

where $Y \in \{A, B, \ldots, N\}$. We often call $A, B, \ldots, N$ *hypotheses* and say that Bayes's rule gives the *posterior* or *inferred* probability $P(Y|X)$ of hypothesis Y after occurrence of event X. It is common to call $P(Y)$ the *prior* probability (or a priori probability) of hypothesis Y before the occurrence of event X. In Sections 1.10 and 5.1.3 we analyze the meaning of Bayes's rule in detail, and in Chapter 5 we apply it extensively in inductive reasoning. ◇

Example 1.6.4 The notion of mutual independence of two or more events lies at the heart of probability theory. From a mathematical viewpoint the given axioms specify just a special application of the general theory of additive sets. However, the special nature of this application is to a large part contained in the way we formalize the intuitive notion of mutual independence of events. In the following we assume that the events have positive probabilities. Events A and B are *mutually independent* iff $P(A|B) = P(A)$ and $P(B|A) = P(B)$, in other words, $P(A \bigcap B) = P(A)P(B)$. And more generally for $n > 2$ events, events $A, B, \ldots, N$ are mutually independent iff for all subsets of pairwise distinct $X, Y, \ldots, Z$ in $A, B, \ldots, N$ we have

$$P(X|Y \bigcap \cdots \bigcap Z) = P(X).$$

In Example 1.6.1, we have $P(A \bigcap B) \neq P(A)P(B)$, and hence the events A and B are not independent.

The classical work on probability from Laplace to von Mises is essentially concerned with the investigation of sequences of independent events. For instance, in a sequence of throws of a fair coin the throws are treated as mutually independent events. (We do not consider that after a run of a hundred 'heads' the chance on throwing 'tails' has increased.) If in newer developments such as so-called Markov processes one often dispenses with complete independence, then still some weaker analogous requirements have to be imposed to obtain meaningful results. ◇

1.6.3 Continuous Sample Spaces

If a field is infinite, and additionally all countable unions $\bigcup A_n$ of disjoint events A_n belong to it, then we call it a *Borel field* or σ-algebra in honor of E. Borel (1871–1956). We denote a Borel field by the Greek letter σ. It follows easily that in a Borel field σ all countable unions of not necessarily disjoint events also belong to σ, and the same holds

for countable intersections. The closure of $\mathcal{F}$, under the field operations and countable union together, gives the unique smallest Borel field that contains $\mathcal{F}$. Suppose $(\mathcal{F}, P)$ is an infinite probability field. It is a fundamental result of measure theory that an extension of both $\mathcal{F}$ and P under countable union and addition, respectively, preserving satisfaction of Axioms A1 through A6, is always possible and, moreover, unique. The result is called the *Borel extension* (σ, P^*) of the probability field. This is best illustrated by the construction of such an extension in a particular case.

Example 1.6.5 We consider infinite binary sequences. The Borel probability field we aim at has as sample space S the real numbers in the half-open interval $[0, 1)$. We start out by identifying a real ω with its infinite binary expansion $0.\omega_1\omega_2\ldots$. In case a real number has two representations, such as $\frac{1}{2}$, which can be represented by $0.100\ldots$ and $0.011\ldots$, we choose the representation with infinitely many zeros. A *cylinder* Γ_x consisting of all real numbers that start with $0.x$, where x is a finite binary string, is an event. The probability field $(\mathcal{F}, P)$ is formed as follows. The set field $\mathcal{F}$ is the closure of all cylinder events under pairwise union, intersection, and difference. It contains the impossible event (by $\Gamma_0 \bigcap \Gamma_1 = \varnothing$) and the certain event Γ_ϵ. The *uniform* distribution, or *Lebesgue measure,* usually denoted by λ, associates with each cylinder Γ_y a probability $\lambda(\Gamma_y) = 2^{-l(y)}$. By Axioms A1 through A5 all unions and intersections of pairs of cylinders are events, including the empty set, and have associated probabilities. We now consider the closure σ of $\mathcal{F}$ under countably infinite union and the field operations. Then σ is the smallest Borel field containing $\mathcal{F}$ as a subfield. Let A be an arbitrary subset of S. Define $P^*(A)$ as the greatest lower bound on $\sum_n P(A_n)$ for all *coverings* $A \subseteq \bigcap_n A_n$ of A by $A_1, A_2, \ldots$, finitely many or countably infinitely elements from $\mathcal{F}$. It can be shown that for elements A in the original field $\mathcal{F}$ we have $P^*(A) = P(A)$. We can also say that the probability distribution $P(A)$ on the sets in $\mathcal{F}$ associates the *measure* $P(A)$ with A. ◇

Example 1.6.6 Sample spaces can be discrete (natural numbers), countable (the rational numbers), or continuous (the real numbers). We will be interested in discrete versus continuous measures. The discrete measures we consider will have as sample space the natural numbers $\mathcal{N}$ or, equivalently, the set of finite binary sequences $\{0, 1\}^*$. The continuous measures will have as sample space the real numbers $\mathcal{R}$ or, equivalently, the set of one-way infinite binary sequences $\{0, 1\}^\infty$. Consider the continuous sample space $S = \{0, 1\}^\infty$ with measure μ. If ϵ is the empty word, then $\mu(\Gamma_\epsilon) = 1$ by Axiom A4, and for all $x \in \{0, 1\}^*$, $\mu(\Gamma_x) = \mu(\Gamma_{x0}) + \mu(\Gamma_{x1})$ by Axiom A5. For convenience, we will in Chapters 4 and 5 write $\mu(x)$ instead

of $\mu(\Gamma_x)$ and consider a measure μ as a function from the finite binary strings into the positive real numbers satisfying Axioms A4 and A5. $\diamond$

Example 1.6.7 A real-valued function on a sample space S is called a *random variable*. We denote random variables by X, Y, Z. A random variable maps an element from the sample space to an aspect of it we are interested in (or want to measure). For instance, S consists of the set of infinite binary sequences, and for each $\omega = \omega_1\omega_2\ldots$ in S the random variable X is defined as $X(\omega) = \omega_2$. We can also talk about a random variable X that is a finite vector $X_1X_2\ldots X_n$ or infinite vector $X_1X_2\ldots$ with $X_i(\omega) = \omega_i$ for all i. If ω is a sequence of outcomes of fair coin tosses, the measure on S is the Lebesgue measure λ, and $\lambda\{\omega : X_i(\omega) = 0\} = \frac{1}{2}$ for all i. Justified by the definitions above, we call the random variables X_i *independent*.

Another example of a random variable is $Y_i(\omega) = k$, where k is the length of the longest uninterrupted subsequence of zeros in $\omega_{1:i}$. Clearly, always either $Y_i(\omega) = Y_{i+1}(\omega)$ or $Y_i(\omega) < Y_{i+1}(\omega)$, where both options occur. The random variables Y_i are *dependent*.

If P is a measure on S, then we customarily denote $P\{\omega : X(\omega) \leq x\}$ by the shorthand $P(X \leq x)$. The function F defined by $F(x) = P(X \leq x)$ is called the *distribution function*. For instance, the random variable X defined as the outcome of a single throw of a fair die has distribution function $P(X \leq i) = i/6$, $i = 1, 2, \ldots, 6$. A random variable is *discrete* if the distribution function F is a step function. The domain of a discrete random variable consists of finitely many or countably infinitely many elements $\{x_1, x_2, \ldots\}$. The function $P(X = x_i)$, $i = 1, 2, \ldots$, is the *probability mass function*. The probability mass function associated with the outcome of a single throw of a fair die is $P(X = i) = \frac{1}{6}$. A random variable is *continuous* if the distribution function F has a continuous derivative f (at most discontinuous in finitely many points). This f is called the *probability density function*. For instance, if the distribution function of a random variable satisfies $F(x) = 1 - e^{-\lambda x}$ for $x > 0$, and $F(x) = 0$ for $x \leq 0$, then the density function is $f(x) = \lambda e^{-\lambda x}$ for $x > 0$, and $f(x) = 0$ for $x \leq 0$. $\diamond$

Exercises

1.6.1. [17] A random sample of size k is taken from a population of n elements. We draw the k elements one after the other and replace each drawn element in the population before drawing the next element. What is the probability of the event that in the sample no element occurs twice, that is, our sample could have been obtained also by sampling without replacement?

Comments. $(n)_k/n^k$. Source: W. Feller, *An Introduction to Probability Theory and Its Applications*, Vol. 1, Wiley, 1968.

1.6.2. [15] Consider the population of digits $\{0, 1, \ldots, 9\}$. Use the formula you have found above to check that the probability that five consecutive random digits are all different is $p = (10)_5/10^5 = 0.3024$.

Comments. Source: W. Feller, *Ibid.*

1.6.3. [15] What is the probability that in a party of 23 people at least 2 people have a common birthday? Assume that birthdays are uniformly distributed over a year of 365 days and that the people at the party constitute a random sample. Use the formula derived above.

Comments. Note that contrary to intuition, it is better to bet that there will be shared birthdays than the other way. For a party of 23, the probability is close to 0.5. However, the analogous probability for a party of only 30 people already exceeds 0.7. Source: W. Feller, *Ibid.*

1.6.4. [10] Show that the probability of obtaining at least one ace (a 6) in four throws with one die is greater than the probability of obtaining at least one double ace in twenty-four throws with two dice.

Comments. This is known as *Chevalier de Méré's paradox.* It was posed by this passionate gambler to Pierre de Fermat, who wrote a solution in a letter to Blaise Pascal in 1654. It was solved earlier by G. Cardano (1501–1576). Source: [*Amer. Math. Monthly* 67(1960), 409–419].

1.6.5. [08] Which probability is greater: to score at least one ace (a 6) in six throws of a die, or to score at least two aces in twelve throws of a die?

Comments. This question was submitted to Isaac Newton by the famous diarist Samuel Pepys in 1693. Newton answered that an easy computation shows that the first event has the greater probability, but failed to convince Pepys. Source: [*The Amer. Statistician,* 14(1960), 27–30].

1.6.6. [M12] The *uniform* distribution over the countable sample space $S = \mathcal{N}$ can be defined as the probability density function

$$L(x) = 2^{-2l(x)-1} \text{ or alternatively, } \frac{6}{\pi^2(l(x)+1)^2}2^{-l(x)}.$$

(a) Show that in both cases $\sum_{x\in S} L(x) = 1$.

(b) Let $S_1, S_2, \ldots$ be a sequence of sample spaces with $S_n = \{x : l(x) = n\}$. Show that the probability density function $L_n(x) = L(x|l(x) = n)$ assigns probability $L_n(x) = 1/2^n$ to all x of length n, and zero probability to other x's, for $n = 1, 2, \ldots$.

1.6.7. [15] The *uniform* distribution λ over the continuous sample space $S = \{0, 1\}^\infty$, the set of one-way infinite binary sequences (the half-open interval of real numbers $[0, 1)$), is described in Example 1.6.5. Let the L_n's be as above. Show that $\lambda(\Gamma_x) = L_{l(x)}(x)$, for all $x = \{0, 1\}^*$.

1.7 Basics of Computability Theory

While the qualitative ideas immanent in Kolmogorov complexity had been around for a long time, their ultimate applicability in quantitative form became possible only after the rise of computability theory (equivalently, recursive function theory) in the 1930s. In this section we develop and review completely the basic notions of that theory insofar as they are needed in the sequel.

In 1936 Alan M. Turing (1912–1954) exhibited an exceedingly simple type of hypothetical machine and gave a brilliant demonstration that everything that can be reasonably said to be computed by a human computer using a fixed procedure can be computed by such a machine. As Turing claimed, any process that can be naturally called an effective procedure is realized by a Turing machine. This is known as *Turing's thesis*. Over the years, all serious attempts to give precise yet intuitively satisfactory definitions of a notion of 'effective procedure' in the widest possible sense have turned out to be equivalent—to define essentially the same class of processes. (In his original paper, Turing established the equivalence of his notion of 'effective procedure' with Alonzo Church's (1903–1995) notion of 'effective calculability.') *Church's thesis* states that in this sense, there is an objective notion of effective computability independent of a particular formalization.

While the formal part of Turing's paper is difficult to follow for the contemporary reader, the informal arguments he sets forth are as lucid and convincing now as they were then. To us it seems that it is the best introduction to the subject, and we reproduce this superior piece of expository writing below.

> "All arguments [for Turing's thesis] are bound to be, fundamentally, appeals to intuition, and for that reason rather unsatisfactory mathematically. The real question at issue is: 'what are the possible processes which can be carried out in computing (a number)?' The arguments which I shall use are of three kinds.
>
> (a) A direct appeal to intuition.
>
> (b) A proof of equivalence of two definitions (in case the new definition has a greater intuitive appeal).
>
> (c) Giving examples of large classes of numbers which are computable.
>
> Once it is granted that computable numbers are all 'computable [by Turing machines],' several other propositions of the same character follow. In particular it follows that, if there is a general process for determining whether a formula (of the Hilbert function calculus) is provable, then the determination can be carried out by machine. [...]
>
> Computing is normally done by writing certain symbols on paper. We may suppose this paper to be divided into squares like a child's arithmetic book. In elementary arithmetic the 2-dimensional character of the paper is sometimes used. But such use is always avoidable, and I think it will be agreed that the

two-dimensional character of paper is no essential of computation. I assume then that the computation is carried out on one-dimensional paper, on a tape divided into squares. I also suppose that the number of symbols which may be printed is finite. If we were to allow an infinity of symbols, then there would be symbols differing to an arbitrarily small extent.[1]

The effect of this restriction on the number of symbols is not very serious. It is always possible to use sequences of symbols in the place of single symbols. Thus an Arabic numeral such as 17 or 999999999999999 is normally treated as a single symbol. Similarly in any European language words are treated as single symbols (Chinese, however, attempts to have an enumerable infinity of [atomic] symbols). The differences in our point of view between single and compound symbols is that the compound symbols, if they are too lengthy, cannot be observed at one glance. This is in accordance with experience. We cannot tell at a glance whether 99999999999999999 and 999999999999999999 are the same. [...]

The behaviour of the [human] computer at any moment is determined by the symbols he is observing, and his 'state of mind' at that moment. We may suppose that there is a bound B to the number of symbols or squares which the computer can observe at one moment. If he wishes to observe more, he must use successive observations. We will also suppose that the number of states of mind which need be taken into account is finite. The reasons for this are of the same character as those which restrict the number of symbols. If we admit an infinity of states of mind, some of them will be 'arbitrarily close' and will be confused. Again, the restriction is not one which seriously affects computation, since the use of more complicated states of mind can be avoided by writing more symbols on the tape. [...]

Let us imagine the operations performed by the computer to be split up in 'simple operations' which are so elementary that it is not easy to imagine them further divided. Every such operation consists of some change of the physical system consisting of the computer and his tape. We know the state of the system if we know the sequence of symbols on the tape, which of these are observed by the computer (possibly with a special order), and the state of mind of the computer. We may suppose that in a simple operation not more than one symbol is altered. Any other changes can be split up into simple changes of this kind. The situation in regard to the squares whose symbols may be altered this way is the same as in regard to the observed squares. We may, therefore, without loss of generality, assume that the squares whose symbols are changed are the 'observed' squares. [...]

---

[1] If we regard a symbol as literally printed on a square, we may suppose that the square is $0 < x < 1$, $0 < y < 1$. The symbol is defined as the set of points in this square, viz., the set occupied by printer's ink. If these sets are restricted to be measurable, we can define the 'distance' between two symbols as the cost of transforming one symbol into the other if the cost of moving a unit area of printer's ink unit distance is unity, and there is an infinite supply of ink at $x = 2$, $y = 0$. With this topology the symbols form a conditionally compact space [Turing's note].

Besides these changes of symbols, the simple operations must include changes of distribution of observed squares. The new observed squares must immediately be recognized by the computer. I think it is reasonable to suppose that they can only be squares whose distance from the closest of the immediately previously observed squares does not exceed a certain fixed amount. Let us say that each of the new observed squares is within L squares of the immediately previously observed square. [...]

In connection to 'immediate recognizability,' it may be thought that there are other kinds of squares which are immediately recognizable. In particular, squares marked by special symbols may be taken as immediately recognizable. Now if these squares are marked only by single symbols there can be only a finite number of them, and we should not upset our theory by adjoining these marked squares to the observed squares. If, on the other hand, they are marked as a sequence of symbols, we cannot regard the process of recognition as a simple process. This is a fundamental point and should be illustrated. In most mathematical papers the equations and theorems are numbered. Normally the numbers go not beyond (say) 1000. It is, therefore, possible to recognize a theorem at a glance by its number. But if the paper was very long, we might reach Theorem 157767733443477; then, further on in the paper, we might find '... hence (applying Theorem 157767733443477) we have ....' In order to make sure which was the relevant theorem we should have to compare the two numbers figure by figure, possibly ticking the figures off in pencil to make sure of their not being counted twice. If in spite of this it is still thought that there are other 'immediately recognizable' squares, it does not upset my contention so long as these squares can be found by some process of which my type of machine is capable. [...]

The simple operations must therefore include:

(a) Changes of the symbol on one of the observed squares.

(b) Changes of one of the squares observed to another square within L squares of one of the previously observed squares.

It may be that some of these changes necessarily involve a change of state of mind. The most general single operation must therefore be taken to be one of the following:

(A) A possible change (a) of symbol together with a possible change of state of mind.

(B) A possible change (b) of observed squares, together with a possible change of state of mind.

The operation actually performed is determined, as has been suggested [above] by the state of mind of the computer and the observed symbols. In particular, they determine the state of mind of the computer after the operation. [...]

We may now construct a machine to do the work of this computer. To each state of mind of the computer corresponds an 'm-configuration' of the machine. The machine scans B squares corresponding to the B squares observed by the computer. In any move the machine can change a symbol on a scanned square or can change any one of the scanned squares to another square distant not more than L squares from one of the other scanned squares. [Without loss

of generality restrict L and B to unity.] The move which is done, and the succeeding configuration, are determined by the scanned symbol and the m-configuration. The machines just described do not differ very essentially from computing machines as defined (previously) and corresponding to any machine of this type a computing machine can be constructed to compute the same sequence, that is to say, the sequence computed by the computer. [...]

We suppose [above] that the computation is carried out on a tape; but we avoid introducing the 'state of mind' by considering a more physical and definitive counterpart of it. It is always possible for the computer to break off from his work, to go away and forget all about it, and later to come back and go on with it. If he does this he must leave a note of instructions (written in some standard form) explaining how the work is to be continued. This note is the counterpart of 'the state of mind.' We will suppose that the computer works in such a desultory manner that he never does more than one step and write the next note. Thus the state of progress of the computation at any stage is completely determined by the note of instructions and the symbols on the tape. That is, the state of the system may be described by a single expression (sequence of symbols), consisting of the symbols on the tape followed by a special marker (which we suppose not to appear elsewhere) and then by the note of instructions. This expression may be called the 'state formula.' We know that the state formula at any given stage is determined by the state formula before the last step was made, and we assume that the relation of these two formulae is expressible in the functional calculus. In other words, we assume that there is an axiom A which expresses the rules governing the behaviour of the computer, in terms of the relation of the state formula at any stage to the state formula at the preceding stage. If this is so, we can construct a machine to write down the successive state formulae, and hence to compute the required number."

1.7.1 Effective Enumerations and Universal Machines

We formalize Turing's description as follows: A *Turing machine* consists of a finite program, called the *finite control,* capable of manipulating a linear list of *cells*, called the *tape,* using one access pointer, called the *head* (Figure 1.1). We refer to the two directions on the tape as *right* and *left*. The finite control can be in any one of a finite set of states Q, and each tape cell can contain a 0, a 1, or a *blank* B. Time is discrete and the time instants are ordered $0, 1, 2, \ldots$, with 0 the time at which the machine starts its computation. At any time, the head is positioned over a particular cell, which it is said to *scan*. At time 0 the head is situated on a distinguished cell on the tape called the *start cell*, and the finite control is in a distinguished state q_0. At time 0 all cells contain B's, except for a contiguous finite sequence of cells, extending from the start cell to the right, which contain 0's and 1's. This binary sequence is called the *input*.

The device can perform the following basic operations:

1. it can write an element from $A = \{0, 1, B\}$ in the cell it scans; and

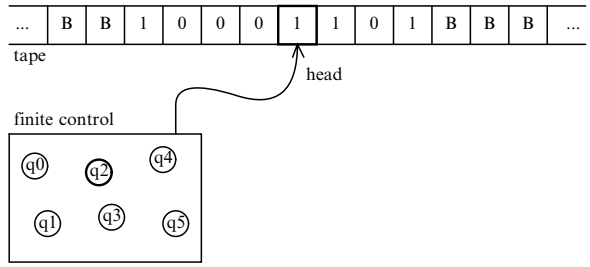


FIGURE 1.1. Turing machine

2. it can shift the head one cell left or right.

When the device is active it executes these operations at the rate of one operation per time unit (a *step*). At the conclusion of each step, the finite control takes on a state from Q. The device is constructed so that it behaves according to a finite *list* of *rules*. These rules determine, from the current state of the finite control and the symbol contained in the cell under scan, the operation to be performed next and the state to enter at the end of the next operation execution.

The rules have format (p, s, a, q): p is the current state of the finite control; s is the symbol under scan; a is the next operation to be executed of type (1) or (2) designated in the obvious sense by an element from $S = \{0, 1, B, L, R\}$; and q is the state of the finite control to be entered at the end of this step.

Any two distinct quadruples cannot have their first two elements identical: the device is *deterministic*. Not every possible combination of the first two elements has to be in the set; in this way we permit the device to perform *no* operation. In this case we say that the device *halts*. Hence, we can define a Turing machine by a mapping from a finite subset of $Q \times A$ into $S \times Q$. Given a Turing machine and an input, the Turing machine carries out a uniquely determined succession of operations, which may or may not terminate in a finite number of steps.

We can associate a partial function with each Turing machine in the following way: The input to the Turing machine is presented as an n-tuple $(x_1, \ldots, x_n)$ of binary strings in the form of a single binary string consisting of self-delimiting versions of the x_i's. The integer represented by the maximal binary string (bordered by blanks) of which some bit is scanned, or 0 if a blank is scanned, by the time the machine halts is called the *output* of the computation.

Definition 1.7.1 Under this convention for inputs and outputs, each Turing machine defines a partial function from n-tuples of integers onto the integers, $n \geq 1$.

We call such a function *partial recursive.* If the Turing machine halts for all inputs, then the function computed is defined for all arguments and we call it *total recursive,* or simply *recursive.*

We call a function with range $\{0,1\}$ a *predicate*, with the interpretation that the predicate of an n-tuple of values is 'true' if the corresponding function assumes value 1 for that n-tuple of values for its arguments and is 'false' or 'undefined' otherwise. Hence, we can talk about *partial* (*total*) *recursive predicates.*

Example 1.7.1 Consider x as a binary string. It is easy to see that the functions $l(x)$, $f(x) = \bar{x}$, $g(\bar{x}y) = x$, and $h(\bar{x}y) = y$ are partial recursive. Functions g and h are not total since the value for input 1111 is not defined. The function $g'(\bar{x}y)$ defined as 1 if $x = y$ and as 0 if $x \neq y$ is a recursive predicate. Consider x as an integer. The following functions are basic n-place total recursive functions: the *successor* function $\gamma^{(1)}(x) = x + 1$, the *zero* function $\zeta^{(n)}(x_1, \ldots, x_n) = 0$, and the *projection* function $\pi_m^{(n)}(x_1, \ldots, x_n) = x_m$ $(1 \leq m \leq n)$. ◇

Example 1.7.2 The function $\langle x, y\rangle = \bar{x}y$ is a total recursive one-to-one mapping from $\mathcal{N} \times \mathcal{N}$ into $\mathcal{N}$. We can easily extend this scheme to obtain a total recursive one-to-one mapping from k-tuples of integers into the integers, for each fixed k. Define $\langle n_1, n_2, \ldots, n_k\rangle = \langle n_1, \langle n_2, \ldots, n_k\rangle\rangle$.

Another total recursive one-to-one mapping from k-tuples of integers into the integers is $\langle n_1, n_2, \ldots, n_k\rangle = \bar{n}_1 \ldots \bar{n}_{k-1}\bar{n}_k$. ◇

Church's thesis. *The class of algorithmically computable numerical functions* (*in the intuitive sense*) *coincides with the class of partial recursive functions.*

Originally intended as a *proposal* to henceforth supply intuitive terms such as 'computable' and 'effective procedure' with a precise meaning as 'recursive' and 'recursive function,' Church's thesis has come into use as shorthand for a claim that from a given description of a procedure in terms of an informal set of instructions we can derive a formal one in terms of Turing machines.

It is possible to give an effective (computable) one-to-one pairing between natural numbers and Turing machines. This is called an *effective enumeration.* One way to do this is to encode the table of rules of each Turing machine in binary, in a canonical way.

Example 1.7.3 The only thing we have to do for every Turing machine is to encode the defining mapping T from $Q \times A$ into $S \times Q$. Giving each element

of $Q \bigcup S$ a unique binary code requires s bits for each such element, with $s = \lceil \log(d(Q) + 5) \rceil$. Denote the encoding function by e. Then the quadruple $(p, 0, B, q)$ is encoded as $e(p)e(0)e(B)e(q)$. If the number of rules is r, then $r \leq 3d(Q)$. We agree to consider the state of the first rule as the start state. The entire list of quadruples,

$$T = (p_1, t_1, s_1, q_1), (p_2, t_2, s_2, q_2), \ldots, (p_r, t_r, s_r, q_r),$$

is encoded as

$$E(T) = \bar{s}\bar{r}e(p_1)e(t_1)e(s_1)e(q_1) \ldots e(p_r)e(t_r)e(s_r)e(q_r).$$

Note that $l(E(T)) \leq 4rs + 2 \log rs + 4$. (Moreover, E is self-delimiting, which is convenient in situations in which we want to recognize the substring $E(T)$ as prefix of a larger string; see Section 1.4.)

We order the resulting binary strings lexicographically (according to increasing length). We assign an *index*, or Gödel number, $n(T)$ to each Turing machine T by defining $n(T) = i$ if $E(T)$ is the ith element in the lexicographic order of Turing machine codes. This yields a sequence of Turing machines $T_1, T_2, \ldots$ that constitutes the effective enumeration. One can construct a Turing machine to decide whether a given binary string x encodes a Turing machine, by checking whether it can be decoded according to the scheme above, that the tuple elements belong to $Q \times A \times S \times Q$, followed by a check whether any two different rules start with the same two elements. This observation enables us to construct 'universal' Turing machines. ◇

A *universal* Turing machine U is a Turing machine that can imitate the behavior of any other Turing machine T. It is a fundamental result that such machines exist and can be constructed effectively. Only a suitable description of T's finite program and input needs to be entered on U's tape initially. To execute the consecutive actions that T would perform on its *own* tape, U uses T's description to simulate T's actions on a representation of T's tape contents. Such a machine U is also called 'computation universal.' In fact, there are infinitely many such U's.

Example 1.7.4 We focus on a universal Turing machine U that uses the encoding above. It is not difficult, but tedious, to define a Turing machine in quadruple format that expects inputs of the format $1^i 0p$ and acts as follows: On the tape left of the input, U starts to generate the successive strings in $\{0, 1\}^*$ in lexicographic order according to increasing length and checks for each such string whether it encodes a Turing machine. For each Turing machine it finds, it replaces one 1 in 1^i in the input by a B. The binary string $E(T)$ that causes the delimiter 0 to be read encodes T with $n(T) = i$. Subsequently, U starts to execute the successive operations of T using p as input and the description $E(T)$ of T it has found. We omit the explicit construction of U. ◇

For the contemporary reader there should be nothing mysterious in the concept of a general-purpose computer which can perform any computation when supplied with an appropriate program. The surprising thing is that a general-purpose computer can be *very simple*: it has been shown that four tape symbols and seven states suffice easily in the above scheme [M. Minsky, *Computation: Finite and Infinite Machines*, Prentice-Hall, 1967]. This machine can be changed to, in the sense of being simulated by, our format using tape symbols $\{0, 1, B\}$ at the cost of an increase in the number of of states; see Section 1.12.

The effective enumeration of Turing machines $T_1, T_2, \ldots$ determines an effective enumeration of partial recursive functions $\phi_1, \phi_2, \ldots$ such that ϕ_i is the function computed by T_i, for all i. It is important to distinguish between a *function* ψ and a *name* for ψ. A name for ψ can be an *algorithm* that computes ψ, in the form of a Turing machine T. It can also be a natural number i such that ψ equals ϕ_i in the above list. We call i an *index* for ψ. Thus, each partial recursive ψ occurs many times in the given effective enumeration, that is, it has many indices.

Definition 1.7.2 The partial recursive function $\nu^{(2)}(i, x)$ computed by the universal Turing machine U is called the *universal partial recursive function.*

The generalization to n-place functions is straightforward. A partial recursive function $\nu^{(n+1)}(i, x_1, \ldots, x_n)$ is *universal* for all n-place partial recursive functions if for each partial recursive function $\phi^{(n)}(x_1, \ldots, x_n)$ there exists an i such that the mapping $\nu^{(n+1)}$ with the first argument fixed to i is identical to the mapping $\phi^{(n)}$. Here i is an index of $\phi^{(n)}$ with respect to $\nu^{(n+1)}$. For each n, we fix a partial recursive $(n + 1)$-place function that is universal for all n-place partial recursive functions. The following lemma is usually called the *enumeration theorem* for n-place partial recursive functions. Here z is the index of the universal function.

Lemma 1.7.1 *For each n there exists an index z such that for all i and $x_1, \ldots, x_n$, if $\phi_i^{(n)}(x_1, \ldots, x_n)$ is defined, then $\phi_z^{(n+1)}(i, x_1, \ldots, x_n) = \phi_i^{(n)}(x_1, \ldots, x_n)$, and $\phi_z^{(n+1)}(i, x_1, \ldots, x_n)$ is undefined otherwise. ($\phi_z^{(n+1)}$ is a universal partial recursive function that enumerates the partial recursive functions of n variables.)*

Proof. The machine of Example 1.7.4 is universal for all n-place partially recursive functions, for all n. □

It is easy to see that universality is oblivious to n-arity, at least in the chosen computational model of Turing machines. We say that a partial recursive function $\phi^{(n)}$ on arguments $x_1, \ldots, x_n$ is computed in *time* t (t steps or t operations) by Turing machine T if T computes $\phi^{(n)}$ and

T halts within t steps when it is started on the input corresponding to these arguments.

Lemma 1.7.2 *For all partial recursive functions $\phi^{(n)}$ there exists a recursive function $\psi^{(n+1)}$ such that $\psi^{(n+1)}(t, x_1, \ldots, x_n) = 1$ if $\phi^{(n)}(x_1, \ldots, x_n)$ can be computed in not more than t steps, and $\psi^{(n+1)}(t, x_1, \ldots, x_n) = 0$ otherwise.*

Proof. Let T be a Turing machine that computes ϕ. Modify T to a Turing machine T' that computes ψ from an input with one extra variable, the *clock*, containing the natural number t. Machine T' works exactly like T, except that it decrements the clock by one for every simulated step of T. If simulator T' enters a halting configuration of T with a positive clock value, then T' outputs 1. If T' decrements its clock to zero, then it halts and outputs 0. □

A set A is *recursively enumerable* if it is empty or the range of some total recursive function f. We say that f *enumerates* A. The intuition behind this definition is that there is a Turing machine for listing the elements of A in some arbitrary order with repetitions allowed. An equivalent definition is that A is recursively enumerable if it is accepted by a Turing machine. That is, for each element in A, the Turing machine halts in a distinguished accepting state, and for each element not in A the machine either halts in a nonaccepting state or computes forever.

A set A is *recursive* if it possesses a recursive characteristic function. That is, A is recursive iff there exists a recursive function f such that for all x, if $x \in A$, then $f(x) = 1$, and if $x \in \bar{A}$, then $f(x) = 0$ ($\bar{A}$ is the complement of A). An equivalent definition is that A is recursive if A is accepted by a Turing machine that always halts. Obviously, all recursive sets are recursively enumerable.

Example 1.7.5 The following sets are recursive: (i) the set of odd integers; (ii) the set of natural numbers; (iii) the empty set; (iv) the set of primes; (v) every finite set; (vi) every set with a finite complement. The following sets are recursively enumerable: (i) every recursive set; (ii) the set of indices i such that the range of ϕ_i is nonempty; (iii) the set $\{x :$ a run of at least x consecutive 0's occurs in $\pi\}$, where $\pi = 3.1415\ldots$. ◇

Lemma 1.7.3 (i) *A set A is recursive iff both A and its complement $\bar{A}$ are recursively enumerable.*

(ii) *An infinite set A is recursive iff it is recursively enumerable in increasing order. (Here we have postulated a total order on the elements of A. For instance, if $A \subseteq \mathcal{N}$ with the usual order, then ϕ enumerates A in increasing order if $\phi(i) < \phi(i+1)$, for all i.)*

Proof. Let $A \bigcup \bar{A} = \mathcal{N}$ and neither A nor $\bar{A}$ is the empty set. (Otherwise the lemma trivially holds.)

(i) (Only if) If A is recursive, then obviously its complement $\bar{A}$ is recursive as well. Moreover, both are by definition recursively enumerable.

(If) By assumption, A is the range of f, and $\bar{A}$ is the range of g, for two recursive functions f, g. Therefore, we can generate the list $f(0), g(0),\ f(1), g(1),\ f(2), \ldots$, and examine each element in turn. To decide whether a given x in $\mathcal{N}$ belongs to A we just compare it with each element in the above list. If $x = f(i)$ for some i, then $x \in A$. If $x = g(i)$ for some i, then $x \in \bar{A}$. Element x must occur in this list, since $A \bigcup \bar{A} = \mathcal{N}$.

(ii) (Only if) Trivial.

(If) Let A be recursively enumerated in increasing order $a_1, a_2, \ldots$. Then this yields a procedure to decide for each x whether x belongs to A by testing "$a_i = x$?" for $i = 1, 2, \ldots$ until either $a_i = x$ or $x < a_i$. □

Lemma 1.7.4 *Every infinite recursively enumerable set contains an infinite recursive subset.*

Proof. Let A be infinite and recursively enumerable. Let f be a recursive function with range A. Define a new recursive function g with $g(0) = f(0)$ and $g(x+1) = f(y)$, where y is the least value such that $f(y) > g(x)$. Let B be the range of g. Since A is infinite, the definition of B implies that B is infinite. Clearly g enumerates B in increasing order, so B is recursive by Lemma 1.7.3, Item (ii). □

The equivalent lemmas hold for recursive and recursively enumerable sets of n-tuples.

1.7.2 Undecidability of the Halting Problem

Turing's paper, and more so Kurt Gödel's paper, where such a result first appeared, are celebrated for showing that certain well-defined questions in the mathematical domain cannot be settled by *any* effective procedure for answering questions. One such question is, "which machine computations eventually terminate with a definite result, and which machine computations go on forever without a definite conclusion?" This is sometimes called the *halting problem.* Since all machines can be simulated by the universal Turing machine U, this question cannot be decided in the case of the single machine U, or more generally for *any* individual universal machine. The following lemma, due to Turing in 1936, formalizes this discussion. Let $\phi_1, \phi_2, \ldots$ be the standard enumeration of partial recursive functions.

Lemma 1.7.5 *There is no recursive function g such that for all x, y, we have $g(x, y) = 1$ if $\phi_x(y)$ is defined, and $g(x, y) = 0$ otherwise.*

Proof. Suppose the contrary, and define a partial recursive function ψ by $\psi(x) = 1$ if $g(x, x) = 0$, and $\psi(x)$ is undefined otherwise. (The definition of ψ gives by the assumption of total recursiveness of g an algorithm, and by Church's thesis or by explicit construction we can find a Turing machine to compute ψ.) Let ψ have an index y in the fixed enumeration of partial recursive functions, $\psi = \phi_y$. Then, $\phi_y(y)$ is defined iff $g(y, y) = 0$, according to ψ's definition. But this contradicts the assumption of existence of g as defined in the statement of the lemma. □

The trick used in this proof is called *diagonalization*; see Exercise 1.7.3 on page 40.

Definition 1.7.3 Define $K_0 = \{\langle x, y\rangle : \phi_x(y) < \infty\}$ as the *halting set*.

With this definition, Lemma 1.7.5 can be rephrased as, "The halting set K_0 is not recursive." It is easy to see that K_0 is recursively enumerable. The halting set is so ubiquitous that it merits the standard notation K_0. We shall also use the *diagonal halting set* $K = \{x : \phi_x(x) < \infty\}$. Just like K_0, the diagonal halting set is recursively enumerable; and the proof of Lemma 1.7.5 shows that K is not a recursive set.

Lemma 1.7.5 was preceded by the famous (first) incompleteness theorem of Kurt Gödel in 1931. Recall that a formal theory T consists of a set of well-formed formulas, *formulas* for short. For convenience these formulas are taken to be finite binary strings. Invariably, the formulas are specified in such a way that an effective procedure exists that decides which strings are formulas and which strings are not.

The formulas are the objects of interest of the theory and constitute the meaningful statements. With each theory we associate a set of *true* formulas and a set of *provable* formulas. The set of true formulas is 'true' according to some (often nonconstructive) criterion of truth. The set of provable formulas is 'provable' according to some (usually effective) syntactic notion of proof.

A *theory* T is simply any set of formulas. A theory is *axiomatizable* if it can be effectively enumerated. For instance, its axioms (initial formulas) can be effectively enumerated and there is an effective procedure that enumerates all proofs for formulas in T from the axioms. A theory is *decidable* if it is a recursive set. A theory T is *consistent* if not both formula x and and its negation $\neg x$ are in T. A theory T is *sound* if each formula x in T is true (with respect to the standard model of the natural numbers).

Hence, soundness implies consistency. A particularly important example of an axiomatizable theory is *Peano arithmetic*, which axiomatizes the standard elementary theory of the natural numbers.

Lemma 1.7.6 *There is a recursively enumerable set K_0 such that for every axiomatizable theory T that is sound and extends Peano arithmetic, there is a number n such that the formula "$n \notin K_0$" is true but not provable in T.*

Proof. Let $\phi_1, \phi_2, \ldots$ be the standard enumeration of partial recursive functions above, and let $K_0 = \{\langle x, y\rangle : \phi_x(y) < \infty\}$. That is, K_0 is the domain of a universal partial recursive function, Lemma 1.7.1. It is easy to see that the set K_0 is recursively enumerable. Moreover, all true statements of the form "$n \in K_0$" belong to Peano arithmetic.

Assume by way of contradiction that all true statements of the form "$n \notin K_0$" are provable in T. Then the complement of K_0 is recursively enumerable by enumerating the set of all provable statements in T. By Lemma 1.7.3, Item (i), if both K_0 and its complement are recursively enumerable, then K_0 is recursive. However, this contradicts the fact that K_0 is nonrecursive (Lemma 1.7.5). □

In his original proof, Gödel uses diagonalization to prove the incompleteness of any sufficiently rich logical theory T with a recursively enumerable axiom system, such as Peano arithmetic. By his technique he exhibits for such a theory an explicit construction of an *undecidable statement* y that says of itself "I am unprovable in T." The formulation in terms of recursive function theory, Lemma 1.7.6, is due to A. Church and S.C. Kleene. In the proof, diagonalization is needed to show that K_0 is not recursive. And elaboration of this proof would yield a similar explicit construction to the above one. Using Kolmogorov complexity we will be able to derive a new proof of Lemma 1.7.6, with essentially different examples of undecidable statements.

1.7.3 Semi-Computable Functions

Recall that $\mathcal{N}$, $\mathcal{Q}$, and $\mathcal{R}$, denote the nonnegative integers, the rational numbers, and the real numbers, respectively. We consider partial recursive functions $g(\langle\langle y, z\rangle, k\rangle) = \langle p, q\rangle$ and write $g(y/z, k) = p/q$, with y, z, p, q, k nonnegative integers. The extension to negative arguments and values is straightforward. The interpretation is that g is a *rational-valued* function of a *rational argument* and a *nonnegative integer argument.*

Definition 1.7.4 A partial function $f : \mathcal{Q} \to \mathcal{R}$ is *upper semicomputable* if it is defined by a rational-valued partial recursive function $\phi(x, k)$ with x a rational number and k a nonnegative integer such that $\phi(x, k+1) \leq \phi(x, k)$ for every k and $\lim_{k\to\infty} \phi(x, k) = f(x)$. This means that f can be computably approximated from above. A function f is lower semicomputable

if $-f$ is upper semicomputable. A function is called *semicomputable* if it is either upper semicomputable or lower semicomputable or both. If a function f is both upper semicomputable and lower semicomputable on its domain, then we call f *computable* (or recursive if the domain is integer or rational). The total function versions are defined similarly.

Thus, a total function $f : \mathcal{Q} \to \mathcal{R}$ is computable iff there is a total recursive function $g(x,k)$ such that $|f(x) - g(x,k)| < 1/k$.

In this way, we extend the notion of integer recursive functions to real-valued computable functions with rational arguments, and to real-valued semicomputable functions with rational arguments. The idea is that a semicomputable function can be approximated from one side by a recursive function, but we may never know how close we are to the real value. A computable function can be approximated to any degree of precision by a recursive function.

Example 1.7.6 The following properties are easily proven: A function $f : \mathcal{Q} \to \mathcal{R}$ is lower semicomputable iff the set $\{(x,r) : r \leq f(x),\ r \in \mathcal{Q}\}$ is recursively enumerable. Therefore, a lower semicomputable function is 'recursively enumerable from below,' and an upper semicomputable function is 'recursively enumerable from above.'

In the previous editions of this book, we used 'enumerable' for 'lower semicomputable,' 'coenumerable' for 'upper semicomputable,' and 'recursive' for 'computable.'

As stated, a function is computable iff it is both upper semicomputable and lower semicomputable. A recursive function is computable. Not all upper semicomputable and lower semicomputable functions are computable. ◇

Example 1.7.7 We give an example of a lower semicomputable function that is not computable. Let $K = \{x : \phi_x(x) < \infty\}$ be the diagonal halting set. Define $f(x) = 1$ if $x \in K$, and $f(x) = 0$ otherwise. We first show that $f(x)$ is lower semicomputable. Define $g(x,k) = 1$ if the Turing machine computing ϕ_x halts in at most k steps on input x, and $g(x,k) = 0$ otherwise. Obviously, g is a rational-valued recursive function. Moreover, for all x and k we have $g(x,k+1) \geq g(x,k)$, and $\lim_{k\to\infty} g(x,k) = f(x)$. Hence, f is lower semicomputable. However, if $f(x)$ were computable, then the set $\{x : f(x) = 1\}$, that is, the diagonal halting set K, would be recursive. But Lemma 1.7.5 shows that it is not. ◇

Example 1.7.8 In Section 1.6 we have defined the notion of measure functions as μ functions that map subsets of the real interval $[0,1)$ to $\mathcal{R}$. Considering $[0,1)$ as the isomorphic $S = \{0,1\}^\infty$, such functions are defined by the

values $\mu(\Gamma_x)$, where $\Gamma_x = \{x\omega : \omega \in \{0,1\}^\infty\}$ with $x \in \{0,1\}^*$ are the so-called cylinder sets. We can extend the notions of computability to set functions. A *measure* μ is *computable* (*upper semicomputable, lower semicomputable*) iff the function $f : \mathcal{N} \to \mathcal{R}$ defined by $f(x) = \mu(\Gamma_x)$ is computable (upper semicomputable, lower semicomputable). ◇

This is enough on semicomputable functions to tide us over Chapters 2 and 3, and we go into more detail in Chapter 4.

1.7.4 Feasible Computations

We give a brief introduction to computational complexity theory in order to provide some basic ideas and establish terminology required for applications of Kolmogorov complexity in Chapters 6, 7, and 8.

Theoretically, any recursive function is computable by a personal computer or by a Turing machine as shown in Figure 1.1. But a computation that takes 2^n steps on an input of length n would not be regarded as *practical* or *feasible*. No computer would ever finish such a computation in the lifetime of the universe even with n merely 1000. Computational complexity theory tries to identify problems that are feasibly computable.

If we have 10^9 processors each taking 10^9 steps/second, then we can execute $3.1 \times 10^{25} < 2^{100}$ steps/year.

In computational complexity theory, we are often concerned with languages. A *language* over a finite alphabet Σ is simply a subset of Σ^*. We say that a Turing machine *accepts* a language L if it outputs 1 when the input is a member of L and outputs 0 otherwise. That is, the Turing machine computes a predicate.

Definition 1.7.5 **(Computational complexity)** Let T be a Turing machine. For each input of length n, if T makes at most $t(n)$ moves before it stops, then we say that T runs in time $t(n)$, or has *time complexity* $t(n)$. If T uses at most $s(n)$ tape cells in the above computation, then we say that T uses $s(n)$ space, or has *space complexity* $s(n)$.

For convenience, we often give the Turing machine in Figure 1.1 a few more work tapes and designate one tape as a read-only input tape. Thus, each transition rule will be of the form $(p, \bar{s}, a, q)$, where $\bar{s}$ contains the scanned symbols on all the tapes, and p, a, q are as in Section 1.7.1, except that an operation now involves moving maybe more than one head.

We sometimes also make a Turing machine *nondeterministic* by allowing two distinct transition rules to have identical first two components. That is, a nondeterministic Turing machine may have different alternative moves at each step. Several other versions of Turing machines will be

discussed in later chapters. Turing machines are deterministic unless it is explicitly stated otherwise.

It is a fundamental and easy result that any k-tape Turing machine running in $t(n)$ time can be simulated by a Turing machine with just one work tape running in $t^2(n)$ time. Any Turing machine using $s(n)$ space can be simulated by a Turing machine with just one work tape using $s(n)$ space. For each k, if a language is accepted by a k-tape Turing machine running in time $t(n)$ (space $s(n)$), then it can also be accepted by another k-tape Turing machine running in time $ct(n)$ (space $cs(n)$), for any constant $c > 0$. This leads to the following definitions:

Definition 1.7.6 **(Complexity classes)** DTIME$[t(n)]$ is the set of languages accepted by multitape deterministic Turing machines in time $O(t(n))$;

NTIME$[t(n)]$ is the set of languages accepted by multitape nondeterministic Turing machines in time $O(t(n))$;

DSPACE$[s(n)]$ is the set of languages accepted by multitape deterministic Turing machines in $O(s(n))$ space;

NSPACE$[s(n)]$ is the set of languages accepted by multitape nondeterministic Turing machines in $O(s(n))$ space;

P is the complexity class $\bigcup_{c\in\mathcal{N}}$ DTIME$[n^c]$;

NP is the complexity class $\bigcup_{c\in\mathcal{N}}$ NTIME$[n^c]$;

PSPACE is the complexity class $\bigcup_{c\in\mathcal{N}}$ DSPACE$[n^c]$.

We will define more complexity classes in Chapter 7. Languages in P, that is, languages acceptable in *polynomial* time, are considered *feasibly* computable. The nondeterministic version for PSPACE turns out to be identical to PSPACE. The following relationships hold trivially,

$$\text{P} \subseteq \text{NP} \subseteq \text{PSPACE}.$$

It is one of the most fundamental open questions in computer science and mathematics to prove whether either of the above inclusions is proper. Research in computational complexity theory focuses on these questions. In order to solve these problems, one can identify the hardest problems in NP or PSPACE.

Definition 1.7.7 **(Oracle machine)** A Turing machine T with an *oracle* A, where A is a language over T's work tape alphabet, is denoted by T^A. Such a machine operates as a normal Turing machine with the exception that after it has computed a finite string x it can enter a special oracle state and ask whether $x \in A$. The machine T^A gets the correct "yes/no" answer in one step. An oracle machine can use this feature one or more times during each computation.

In Exercise 1.7.16 on page 43 we define the recursion-theoretic notion of reducing one language to another. Here, we scale this notion down to feasible size in computational complexity, by limiting the computational power used in the reduction to polynomial time.

A language A is called *polynomial-time Turing-reducible* to a language B, denoted by $A \leq_T^P B$, if given B as an *oracle*, there is a deterministic Turing machine that accepts A in polynomial time. That is, we can accept A in polynomial time given answers to membership in B for free.

Definition 1.7.8 **(NP-completeness)** A language A is called *polynomial-time many-to-one reducible* to a language B, denoted by $A \leq_m^P B$, if there is a function r that is polynomial-time computable, and for every a, $a \in A$ iff $r(a) \in B$. In both cases, if $B \in \mathrm{P}$, then so is A.

A language A is NP-hard if all languages in NP are Turing polynomial-time (equivalently, many-to-one polynomial-time) reducible to A. Consequently, if any NP-hard language is in P, then P = NP. If A is NP-hard and $A \in \mathrm{NP}$, then we say that A is NP-*complete*.

NP is the set of problems for which it is easy to show (give a certificate) that the answer is "yes," and P is the set of "yes/no" problems for which it is easy to find the answer. The technical sense of 'easy' is 'doable by a deterministic Turing machine in polynomial time.' The "P versus NP" question can be understood as whether problems for which it is easy to certify the answer are the same problems for which it is easy to find the answer. The relevance is this:

Normally, we do not ask questions unless we can recognize easily in a certain sense when we have been handed the correct answer. We are not normally interested in questions for which it would take a lifetime of work to check whether you got the answer you wanted. NP is about those questions that we are likely to want answers to.

This excellent explanation was given by one of the inventors of the notions P and NP, J.E. Edmonds [Interview, *FAUW Forum*, University of Waterloo, January 1993].

Example 1.7.9 A Boolean formula is in conjunctive normal form if it is a conjunction of disjunctions. For example,

$$f(x_1, x_2, x_3) = (x_1 + \bar{x}_2 + x_3)(\bar{x}_2 + x_3)(x_1 + x_3)$$

is in conjunctive normal form, and $x_1x_2 + x_2\bar{x}_3$ is not in conjunctive normal form. A Boolean formula $f(x_1, \ldots, x_n)$ is *satisfiable* if there is a Boolean-valued truth assignment $a_1, \ldots, a_n$ such that $f(a_1, \ldots, a_n) = 1$.

Definition 1.7.9 **(SAT)** Let SAT be the set of satisfiable Boolean formulas in conjunctive normal form. The *SAT problem* is to decide whether a given Boolean formula is in SAT.

This problem was the first natural problem shown to be NP-complete. Many practical issues seem to depend on fast solutions to this problem. Given a Boolean formula, a nondeterministic Turing machine can guess a correct truth assignment, and verify it. This takes only linear time. However, if we have to deterministically search for a satisfying truth assignment, there are 2^n Boolean vectors to test.

Intuitively, and as far as is known now, a deterministic Turing machine cannot do much better than simply searching through these Boolean vectors one by one, using an exponential amount of time. ◇

For each class, say P, if a language L is accepted in deterministic polynomial time using an oracle A, then we write $L \in \mathrm{P}^A$. If A is an NP-complete set, we also write $L \in \mathrm{P}^{\mathrm{NP}}$.

Definition 1.7.10 **(Polynomial hierarchy)** The so-called *polynomial hierarchy* consists of the following hierarchy of language classes: $\Sigma_1^p = \mathrm{NP}$; $\Delta_1^p = \mathrm{P}$; $\Sigma_{i+1}^p = \mathrm{NP}^{\Sigma_i^p}$; $\Delta_{i+1}^p = \mathrm{P}^{\Sigma_i^p}$; and $\Pi_i^p = \{\bar{L} : L \in \Sigma_i^p\}$.

Exercises

1.7.1. [10] We can rework the effective enumeration of Turing machines above and the definition of universal Turing machines using only tape alphabet $A = \{0, 1\}$ (as opposed to $\{0, 1, B\}$) such that the $d(A)d(Q)$ product increases at most by a fixed constant under the change of the original Turing machine to its simulator. Namely, we simply replace each program p in $\{0, 1\}^*$ by $\bar{p}$, and in simulated computation use only the alternate (say odd) cells that contain the original program p. To skip over the even administrative cells containing 0's and to detect even cells containing the delimiter 1 takes only a few extra states. In particular, it requires exactly the same number of extra states in *each* Turing machine modification. Prove the assertion concerning the $d(A)d(Q)$ product. What is the 'new' $l(c(T))$? Since clearly $\bar{c}$ suffices, we obtain $l(\bar{c}) \leq 2l(c(T)) + 1$, for all T.

1.7.2. [08] Show that there are exactly countably infinitely many partial recursive functions, and that there are exactly countably infinitely many (total) recursive functions.

1.7.3. [15] Georg Cantor proved that the total functions are not countable by introducing his famous *diagonalization* argument. Suppose the contrary, and count the functions in order $f_1, f_2, \ldots, f_i, \ldots$. Define a

new function g, which we shall prove to be not in this list, by $g(i) = f_i(i) + 1$, for all natural numbers i. By contradictory assumption, g occurs in the list, say $g = f_i$. But by definition, $g(i) \neq f_i(i)$, which gives the required contradiction. Use a similar argument to prove that there are functions that are not partial recursive.

1.7.4. [11] It is important to distinguish between computable *functions* and the *algorithms* that compute those functions. To each function there correspond many different algorithms that compute it. Show that in the effective enumeration of Turing machines, as treated in this section, each partial recursive function is computed by countably infinitely many different Turing machines. If $T_1, T_2, \ldots, T_i, \ldots$ is an effective enumeration of Turing machines and T_i computes partial recursive function ϕ_i, for all i, then each partial recursive function f occurs infinitely many times in the list $\phi_1, \phi_2, \ldots, \phi_i, \ldots$. This result is not an accident of our formalism and effective enumeration, but holds in general for all effective enumerations of algorithms that compute all partial recursive functions.

1.7.5. [25] Show that for every $m, n \geq 1$, there exists a recursive function $\psi = s_n^{(m+1)}$ of $m+1$ variables such that for all $x, y_1, \ldots, y_m$,

$$\phi_x^{(m+n)}(y_1, \ldots, y_m, z_1, \ldots, z_n) = \phi_{\psi(x,y_1,\ldots,y_m)}^{(n)}(z_1, \ldots, z_n),$$

for all variables $z_1, \ldots, z_n$. (Hint: prove the case $m = n = 1$. The proof is analogous for the other cases.)

Comments. This important result, due to Stephen C. Kleene, is usually called the *s-m-n theorem*. Source: H. Rogers, Jr., *Theory of Recursive Functions and Effective Computability*, McGraw-Hill, 1967; P. Odifreddi, *Classical Recursion Theory*, North-Holland, 1989.

1.7.6. [35] If P is the class of all partial recursive functions (for convenience restricted to one variable), then any map π from $\mathcal{N}$ (the natural numbers) *onto* P is called a *numbering*. The standard indexing of the Turing machines provides such a numbering, say π_0, and the indexing of the recursive function definitions provides another, say π_1. A numbering π is *acceptable*, or a *Gödel numbering*, if we can go back and forth effectively between π and π_0, that is:

(i) There is a recursive function f (not necessarily one-to-one) such that $f\pi_0 = \pi$.

(ii) There is a recursive function g (not necessarily one-to-one) such that $g\pi = \pi_0$.

(a) Show that π_1 is acceptable.

(b) Show that (i) is a necessary and sufficient condition that any π have a universal partial recursive function (satisfies an appropriate version of the enumeration theorem).

(c) Show that (ii) is a necessary and sufficient condition that any π have an appropriate version of the *s-m-n* theorem (Exercise 1.7.5).

(d) Show that (ii) implies that $\pi^{-1}(\phi)$ is infinite for every partial recursive ϕ of one variable.

(e) Show that we can replace (i) and (ii) by the requirement that there be a recursive isomorphism between π and π_0.

Comments. These results, due to H. Rogers, Jr., in 1958, give an abstract formulation of the basic work of Church, Kleene, Post, Turing, Markov, and others, that their basic formalizations of the notion of partial recursive functions are effectively isomorphic. It gives invariant significance to the notion of acceptable numbering in that major properties such as the enumeration theorem and the *s-m-n* theorem hold for every acceptable numbering. Note that (i) may be viewed as requiring that the numbering be 'algorithmic' in that each number yields an algorithm; and (ii) that the numbering be 'complete' in that it includes all algorithms. Source: H. Rogers, Jr., *Ibid.*

1.7.7. [20] Show that there is no recursive function f such that $f(x) = 1$ if $\phi_x(x)$ is defined, and $f(x) = 0$ otherwise.

Comments. This fact is known as the *recursive unsolvability of the halting problem.*

1.7.8. [20] Prove that the predicate f defined as $f(x) = 1$ if ϕ_x is total, and $f(x) = 0$ otherwise, is not total recursive.

1.7.9. [15] Prove that a set A is recursively enumerable iff A is the domain of a partial recursive function.

Comments. This is often called the *basic theorem of recursively enumerable sets.*

1.7.10. [16] Prove that a set A is recursively enumerable iff A is the range of some partial recursive function iff A is the range of a total recursive function or $\varnothing$.

1.7.11. [34] (a) Show that it is possible to effectively enumerate the partial recursive functions without repetition.

(b) Let $A = \{x : \phi_x \text{ is a total function}\}$. Prove that A is not recursively enumerable.

Comments. Items (a) and (b) are not contradictory. Hint: in Item (a), dovetail the computations of all partial recursive functions on all arguments. Attributed to R.A. Friedberg, 1958. Source: H. Rogers, Jr., *Theory of Recursive Functions and Effective Computability*, McGraw-Hill, 1967; P. Odifreddi, *Classical Recursion Theory*, North-Holland, 1989, pp. 230–232.

1.7.12. [20] Let $K = \{x : \phi_x(x) < \infty\}$. Prove that K is a recursively enumerable set that is not recursive.

1.7.13. [15] (a) Show that the function $\tau(x, y) = (x^2 + 2xy + y^2 + 3x + y)/2$ is a recursive one-to-one mapping from $\mathcal{N}^2$ *onto* $\mathcal{N}$. Show that this is not a prefix-code.

(b) Show that $\tau^{(k)}$ defined by $\tau^{(2)} = \tau$ and $\tau^{(k)}(x_1, x_2, \ldots, x_k) = \tau(\tau^{(k-1)}(x_1, x_2, \ldots, x_{k-1}), x_k)$ is a recursive one-to-one mapping from $\mathcal{N}^k$ *onto* $\mathcal{N}$. Show that this is not a prefix-code.

(c) Let $E : \mathcal{N} \to \mathcal{N}$ be an effective prefix-code with $E(x)$ the code word for source word x. Show that $E(\tau(x, y))$, and also $E(\tau^{(k)}(x_1, \ldots, x_k))$, for $k > 2$, are effective prefix-codes.

Comments. How good are these prefix-codes of k-tuples of integers in terms of density of the range of the code in $\mathcal{N}$? Clearly, Item (c) is the best possible (in terms of fixed E).

1.7.14. [20] A set A is recursively enumerable *without repetitions* if A equals the range of f, for some f that is recursive and one-to-one. Prove that A is infinite and recursively enumerable iff A is recursively enumerable without repetitions.

1.7.15. [25] (a) Show that there exists an infinite set having no infinite recursively enumerable subset. Such sets have been called *immune* by J.C.E. Dekker.

(b) If a set with this property has a recursively enumerable complement, then this complement was called *simple* by E.L. Post in 1944. Show that there exists a simple set.

Comments. By definition a simple set is recursively enumerable. A simple set is not recursive, since its complement is infinite but not recursively enumerable (Lemma 1.7.3, Item (i)). Source: H. Rogers, Jr., *Ibid.*

1.7.16. [35] In order of increasing generality we define some notions of reducibilities among sets:

A set A is *one-to-one reducible* to a set B ($A \leq_1 B$) if there exists a one-to-one recursive function f such that for all x we have $x \in A$ iff $f(x) \in B$.

A set A is *many-to-one reducible* to a set B ($A \leq_m B$) if there exists a many-to-one recursive function f such that for all x we have $x \in A$ iff $f(x) \in B$.

A set A is *Turing reducible* to a set B ($A \leq_T B$) if we can solve membership in A by a Turing machine that gets the solution to membership in B for free. (This is commonly accepted as formalizing the most general

intuitive notion of reducibility. It is also considered the most significant and useful such notion.)

Intuitively speaking, for r equal to 1, m, or T, reducibility $A \leq_r B$ means that to solve membership in B is at least as 'hard' as to solve membership in A, up to the reducibility notion r.

(a) Consider $\{x : \phi_x(y) < \infty \text{ for infinitely many } y\}$ and $\{x : \phi_x \text{ is total}\}$. Show that each of these sets is reducible to the other in all three senses discussed.

(b) Show that $\leq_r$ is reflexive and transitive for all discussed reducibilities r. Hence, $\equiv_r$ (both $\leq_r$ and $\geq_r$) is an equivalence relation, and $\leq_r$ induces a partial order of the equivalence classes of $\equiv_r$. One equivalence class is *below* another one if members of the first are reducible to members of the second, but not vice versa. We say that a set A on a lower r-equivalence class has a lower *degree of unsolvability* with respect to $\leq_r$.

(c) Consider the diagonal halting set $K = \{x : \phi_x(x) < \infty\}$, and let $A = \{x : \phi_x(y) < \infty \text{ for finitely many } y\}$. Show that K is r-reducible (all $r = 1, m, T$) to A (easy). Show that contrary to intuition, A is not r-reducible (any $r = 1, m, T$) to K (hard). (Hint: show that A is not recursively enumerable.) Therefore, A is of a higher r-degree of unsolvability than K.

(d) Show that the halting set K_0 and K are of the same r-degree of unsolvability (all $r = 1, m, T$).

(e) Show that all recursive sets are of lower r-degree of unsolvability than K_0 (all $r = 1, m, T$).

(f) The following notion is due to E.L. Post. A set A such that each recursively enumerable set is r-reducible to it is called *r-hard*. If A is both recursively enumerable and r-hard, then A is called *r-complete*. Show that the halting set K_0 in the proof of Lemma 1.7.6 is an example of an r-complete set (all $r = 1, m, T$). Show that K in Item (c) is another example.

Comments. Source: H. Rogers, Jr., *Ibid.*

1.7.17. [35] Use the definitions above. The following is known as *Post's problem* (1944). The recursive sets have lower r-degree of unsolvability than K_0, which has the highest r-degree of unsolvability in the recursively enumerable sets. Are there other r-degrees of unsolvability (all $r = 1, m, T$)? For $r = 1, m$, the first examples of such sets were the simple sets of the above exercise; see Item (c) below. (For $r = T$ the question is much harder, and the affirmative answer was provided (independently) by R.A. Friedberg and A.A. Muchnik only in 1956.)

(a) Show that for all A, A is m-complete iff A is 1-complete.

(b) Show that $\{x : \phi_x \text{ is total}\}$ and $\{x : \phi_x \text{ is not total}\}$ are incomparable under $\leq_m$. (Hint: see Exercise 7-11 in H. Rogers, Jr., *Ibid.*)

(c) Show that a simple set is neither recursive nor m-complete.

(d) Show that $\equiv_1$ and $\equiv_m$ do not coincide on the nonrecursive recursively enumerable sets, and hence $\leq_1$ and $\leq_m$ do not coincide on these sets either.

(e) (J.C.E. Dekker) Show that the m-degree of a simple set includes an infinite collection of distinct 1-degrees consisting entirely of simple sets.

Comments. From Item (c) it follows that there exist nonrecursive recursively enumerable sets (simple sets) that are not m-complete (and so by Item (a) not 1-complete). Source: H. Rogers, Jr., *Ibid.* The term 'Post's problem' is now used among recursion theorists only for the Turing reduction version.

1.7.18. [35] Consider the *generalized exponential* function f informally having the following property: $f(0, x, y) = y + x$, $f(1, x, y) = y \times x$, $f(2, x, y) = y^x, \ldots$. A more formal definition of f is given by $f(0, 0, y) = y$, $f(0, x + 1, y) = f(0, x, y) + 1$, $f(1, 0, y) = 0$, $f(z + 2, 0, y) = 1$, $f(z + 1, x + 1, y) = f(z, f(z + 1, x, y), y)$.

(a) Show that this function is recursive but not primitive recursive.

(b) Show that the function $A(x) = f(x, x, x)$, called the *Ackermann* generalized exponential function, is recursive but not primitive recursive. (It rises faster than any primitive recursive function.)

Comments. The function f was given by Wilhelm Ackermann (1926) as a first example of a recursive function that is not primitive recursive [*Math. Ann.* 99(1928), 118–133]. See also D. Hilbert [*Math. Ann.*, 95(1926), 161–190], who credits the proof to Ackermann. There are many variants of definitions of the Ackermann function. A common recursive definition is $A'(0, n) = n + 1$; $A'(i, 0) = A'(i - 1, 1)$ for $i \geq 0$; and $A'(i, n) = A'(i - 1, A'(i, n - 1))$ for $i, n > 0$ Then $A(x) = A'(x, x)$. This definition is apparently due to R.M. Robinson [*Bull. Amer. Math. Soc.* 54(1948), 987–993], and an earlier variant is due to Rózsa Péter [*Math. Ann.* 111(1935), 42–60]. An inherently iterative algorithm to compute A' is given by J.W. Grossman and R.Z. Zeitman [*Theoret. Comput. Sci.*, 37(1988), 327–330]. Another definition for the Ackermann function (source: P. Odifreddi) is h_ω defined as follows: $h_0(x) = x + 1; h_{n+1}(x) = h_n^{(x)}(x); h_\omega(x) = h_x(x)$ where $h_n^{(0)}(x) = x; h_n^{(z+1)}(x) = h_n(h_n^{(z)}(x))$. One of the advantages of this version is that it can be translated into the transfinite. Note: $h_1(x) = 2x; h_2(x) = x2^x$.

1.7.19. The function BB is defined in terms of Turing machines with a purely binary tape alphabet (no blanks) in quintuple format (rather than

quadruple format as above). For each n define the set $A_n = \{i : T_i$ has n states and $\phi_i(0) < \infty\}$. That is, T_i with i in A_n halts when it is started with $p = \epsilon$. We define $BB(n)$ as the maximal number of 1's in the output of any Turing machine in A_n when it is started on input ϵ. It is easy to see that $BB(1) = 1$, but more difficult to see that $BB(2) = 4$. The BB function is known as the *busy beaver* function. It is due to T. Rado [*Bell System Tech. J.* (1962), 877–884]. This was one of the first 'well-defined' incomputable or nonrecursive total functions. The definition is natural and uses no overt diagonalization.

(a) [12] Notice that the BB function is well defined, since you can show that $d(A_n) \leq (6n)^{2n}$.

(b) [15] Show that the BB function is incomputable. (Hint: it grows faster than any recursive function.)

(c) [35] Show that $BB(3) = 6$.

(d) [40] Show that $BB(4) = 13$.

(e) [30] Can you give a lower bound on BB such as the Ackermann generalized exponential function?

(f) [O45] Is $BB(5) > 4{,}098$?

Comments. See A.K. Dewdney, *Scientific American*, 251 (August 1984), 19–23; *Ibid*, 252 (March 1985), 23; A.H. Brady, pp. 259–278 in: *The Universal Turing Machine: A Half-Century Survey*, R. Herken, ed., Oxford Univ. Press, 1988; H. Marxen and J. Buntrock, *EATCS Bull.*, 40(1990), 247–251. For the quadruple format see [A. Oberschelp, K. Schmidt-Goettsch, G. Todt, Castor Quadruplorum, *Archive for Math. Logic*, 27(1988), 35–44].

1.7.20. [39] Let f be any recursive function. Prove that there exists an n such that $\phi_n = \phi_{f(n)}$.

Comments. This result, and its elaborations to more complicated versions, is usually called the *second recursion theorem* or the *fixed-point theorem for recursion theory*. The n is called a *fixed-point* value for f. Standard applications include the following: There exists an e such that the only element in the domain of ϕ_e is e itself, and more generally, it allows us to write programs that know their own index. Source: H. Rogers, Jr., *Theory of Recursive Functions and Effective Computability*, McGraw-Hill, 1967.

1.7.21. [37] In Exercise 1.7.16 we studied the notions of reducibility and degree. We now look at a coarser classification. An n-ary relation R is in the *arithmetic hierarchy* if it is either recursive or, for some m, can be expressed as

$$\{\langle x_1, \ldots, x_n\rangle : (Q_1 y_1) \ldots (Q_m y_m) S(x_1, \ldots, x_n, y_1, \ldots, y_m)\}, \qquad (1.5)$$

where each Q_i denotes the existential quantifier 'there exists' ($\exists$) or the universal quantifier 'for all' ($\forall$), and S is an $(n+m)$-ary recursive relation. The *number of alternations* in the prefix sequence of quantifiers is the number of pairs of adjacent but unlike quantifiers. For each $n > 0$, Formula 1.5 is a Σ_n^0-form if the first quantifier is $\exists$ and the number of alternations is $n-1$. A Σ_0^0-form has no quantifiers. For each $n > 0$, Formula 1.5 is a Π_n^0-form if the first quantifier is $\forall$ and the number of alternations is $n-1$. A Π_0^0-form has no quantifiers. A relation R is in Σ_n^0 if it can be expressed by a Σ_n^0-form. A relation R is in Π_n^0 if it can be expressed by a Π_n^0-form. For each $n \geq 0$, define $\Delta_n^0 = \Sigma_n^0 \bigcap \Pi_n^0$.

(a) Show that $\Sigma_0^0 = \Pi_0^0$ is the class of recursive sets, and that Σ_1^0 is the class of recursively enumerable sets.

(b) Show that R is in Σ_n^0 iff the complement of R is in Π_n^0.

(c) Show that $\Sigma_n^0 \bigcup \Pi_n^0 \subseteq \Sigma_{n+1}^0 \bigcap \Pi_{n+1}^0$.

(d) Show that for each $n > 0$, we have $\Sigma_n^0 - \Pi_n^0 \neq \varnothing$, and hence, by (c), that for each $n > 0$ we have $\Pi_n^0 - \Sigma_n^0 \neq \varnothing$.

Comments. The result mentioned as Item (d) is called the *hierarchy theorem.* In conjunction with Item (c) it shows that the classes $\Sigma_0^0, \Sigma_1^0, \ldots$ form a strictly increasing sequence. Source: H. Rogers, Jr., *Ibid.*

1.7.22. [13] A real number $r \in [0,1]$ is called a *recursive real number* if there exists a total recursive function ϕ such that $r = 0.\omega$ with $\phi(i) = \omega_i$, for all i. We call ω a *recursive sequence.* A recursive sequence of recursive reals is a sequence $r_1, r_2, \ldots$ of reals if there is a total recursive function ψ in two arguments such that $\psi(i,j) = r_{i,j}$ and $r_i = 0.r_{i,1}r_{i,2}\ldots$. Show that not all real numbers are recursive; show that there are only countably many recursive numbers; and show that there is a recursive sequence of recursive reals that converges to a real, but not to a recursive one.

1.8 The Roots of Kolmogorov Complexity

The notion of Kolmogorov complexity has its roots in probability theory, information theory, and philosophical notions of randomness, and came to fruition using the recent development of the theory of algorithms. The idea is intimately related to problems in both probability theory and information theory. These problems as outlined below can be interpreted as saying that the related disciplines are not tight enough; they leave things unspecified that our intuition tells us should be dealt with.

1.8.1 A Lacuna of Classical Probability Theory

An adversary claims to have a true fair coin. However, when he flips it a hundred times, the coin comes up a hundred heads in a row. Upon seeing this, we claim that the coin cannot be fair. The adversary, however, appeals to probability theory, which says that every sequence of outcomes of a hundred coin flips is equally likely having probability $1/2^{100}$, and one sequence had to come up.

Probability theory gives us no basis to challenge an outcome *after* it has happened. We could only exclude unfairness in advance by putting a penalty side bet on an outcome of 100 heads. But what about $1010\ldots$? What about an initial segment of the binary expansion of π?

Regular sequence $\Pr(00000000000000000000000000) = 1/2^{26}$,

Regular sequence $\Pr(01000110110000010100111001) = 1/2^{26}$,

Random sequence $\Pr(10010011011000111011010000) = 1/2^{26}$.

The first sequence is regular, but what is the distinction of the second sequence and the third? The third sequence was generated by flipping a quarter. The second sequence is very regular: $0, 1, 00, 01, \ldots$. The third sequence will pass (pseudo)randomness tests.

In fact, classical probability theory cannot express the notion of *randomness of an individual sequence*. It can only express expectations of properties of outcomes of random processes, that is, the expectations of properties of the total set of sequences under some distribution.

Only relatively recently, this problem has found a satisfactory resolution by combining notions of computability and statistics to express the complexity of a finite object. This complexity is the length of the shortest binary program from which the object can be effectively reconstructed. It may be called the *algorithmic information content* of the object. This quantity turns out to be an attribute of the object alone, and absolute (in the technical sense of being recursively invariant). It is the *Kolmogorov complexity* of the object.

1.8.2 A Lacuna of Information Theory

Shannon's classical information theory assigns a quantity of information to an ensemble of possible messages. All messages in the ensemble being equally probable, this quantity is the number of bits needed to count all possibilities. This expresses the fact that each message in the ensemble can be communicated using this number of bits. However, it does not say anything about the number of bits needed to convey any individual message in the ensemble. To illustrate this, consider the ensemble consisting of all binary strings of length 9999999999999999.

By Shannon's measure, we require 9999999999999999 bits on average to encode a string in such an ensemble. However, the string consisting

of 9999999999999999 1's can be encoded in about 55 bits by expressing 9999999999999999 in binary and adding the repeated pattern 1. A requirement for this to work is that we have agreed on an algorithm that decodes the encoded string. We can compress the string still further when we note that 9999999999999999 equals $3^2 \times 1111111111111111$, and that 1111111111111111 consists of 2^4 1's.

Thus, we have discovered an interesting phenomenon: the description of some strings can be compressed considerably, provided they exhibit enough regularity. This observation, of course, is the basis of all systems to express very large numbers and was exploited early on by Archimedes in his treatise *The Sand Reckoner*, in which he proposes a system to name very large numbers:

> "There are some, King Golon, who think that the number of sand is infinite in multitude [... or] that no number has been named which is great enough to exceed its multitude. [...] But I will try to show you, by geometrical proofs, which you will be able to follow, that, of the numbers named by me [...] some exceed not only the mass of sand equal in magnitude to the earth filled up in the way described, but also that of a mass equal in magnitude to the universe."

However, if regularity is lacking, it becomes more cumbersome to express large numbers. For instance, it seems easier to compress the number one billion, than the number one billion seven hundred thirty-five million two hundred sixty-eight thousand and three hundred ninety-four, even though they are of the same order of magnitude.

1.9 Randomness

This brings us to the main root of Kolmogorov complexity, the notion of randomness. There is a certain inevitability in the development that led A.N. Kolmogorov (1903–1987) to use the recently developed theory of effective computability to resolve the problems attending the proper definition of a random sequence. Indeed, the main idea involved had already been formulated with unerring intuition by P.S. Laplace, but could not be properly quantified at the time (see Chapter 4).

In the context of the above discussion, random sequences are sequences that cannot be compressed. Now let us compare this with the common notions of mathematical randomness. To measure randomness, criteria have been developed that certify this quality. Yet, in recognition that they do not measure 'true' randomness, we call these criteria 'pseudo' randomness tests. For instance, statistical surveys of initial sequences of decimal digits of π have failed to disclose any significant deviations from randomness. But clearly, this sequence is so regular that it can be described by a simple program to compute it, and this program can be expressed in a few bits.

"Any one who considers arithmetical methods of producing random digits is, of course, in a state of sin. For, as has been pointed out several times, there is no such thing as a random number—there are only methods to produce random numbers, and a strict arithmetical procedure is of course not such a method. (It is true that a problem we suspect of being solvable by random methods may be solvable by some rigorously defined sequence, but this is a deeper mathematical question than we can go into now.)" [von Neumann]

This fact prompts more sophisticated definitions of randomness. Notably, Richard von Mises (1883–1953) proposed notions that approach the very essence of true randomness. This is related to the construction of a formal mathematical theory of probability, to form a basis for real applications, in the early part of this century. While von Mises's objective was to justify the applications to real phenomena, Kolmogorov's classic 1933 treatment constructs a purely axiomatic theory of probability on the basis of set-theoretic axioms.

"This theory was so successful, that the problem of finding the basis of real applications of the results of the mathematical theory of probability became rather secondary to many investigators. [... however] the basis for the applicability of the results of the mathematical theory of probability to real 'random phenomena' must depend in some form on the *frequency concept of probability*, the unavoidable nature of which has been established by von Mises in a spirited manner." [Kolmogorov]

The point made is that the axioms of probability theory are designed so that abstract probabilities can be computed, but nothing is said about what probability really means, or how the concept can be applied meaningfully to the actual world. Von Mises analyzed this issue in detail, and suggested that a proper definition of probability depends on obtaining a proper definition of a random sequence. This makes him a frequentist—a supporter of the frequency theory.

The frequency theory to interpret probability says, roughly, that if we perform an experiment many times, then the ratio of favorable outcomes to the total number n of experiments will, *with certainty*, tend to a limit, p say, as $n \to \infty$. This tells us something about the *meaning* of probability, namely, that the measure of the positive outcomes is p. But suppose we throw a coin 1000 times and wish to know what to expect. Is 1000 enough for convergence to happen? The statement above does not say. So we have to add something about the rate of convergence. But we cannot assert a *certainty* about a particular number of n throws, such as "the proportion of heads will be $p \pm \epsilon$ for large enough n (with ϵ depending on n)." We can at best say "the proportion will lie between $p \pm \epsilon$ with at least such and such probability (depending on ϵ and n_0) whenever $n > n_0$." But now we have defined probability in an obviously circular fashion.

In 1919 von Mises proposed to eliminate the problem by simply dividing all infinite sequences into special random sequences (called *collectives*), having relative frequency limits, which are the proper subject of the calculus of probabilities and other sequences. He postulates the existence of random sequences as certified by abundant empirical evidence, in the manner of physical laws, and derives mathematical laws of probability as a consequence. In his view a naturally occurring sequence can be nonrandom or unlawful in the sense that it is not a proper collective.

Von Mises views the theory of probabilities insofar as they are numerically representable as a physical theory of definitely observable phenomena, repetitive or mass events, for instance, as found in games of chance, population statistics, Brownian motion. 'Probability' is a primitive notion of the theory comparable to those of 'energy' or 'mass' in other physical theories.

Whereas energy or mass exists in fields or material objects, probabilities exist only in the similarly mathematical idealization of collectives (random sequences). All problems of the theory of probability consist in deriving, according to certain rules, new collectives from given ones and calculating the distributions of these new collectives. The exact formulation of the properties of the collectives is secondary and must be based on empirical evidence. These properties are the existence of a limiting relative frequency and randomness.

The property of randomness is a generalization of the abundant experience in gambling houses, namely, the impossibility of a successful gambling system. Including this principle in the foundation of probability, von Mises argues, we proceed in the same way as the physicists did in the case of the energy principle. Here too, the experience of hunters of fortune is complemented by solid experience of insurance companies, and so forth.

A fundamentally different approach is to justify a posteriori the application of a purely mathematically constructed theory of probability, such as the theory resulting from the Kolmogorov axioms. Suppose we can show that the appropriately defined random sequences form a set of measure one, and without exception satisfy all laws of a given axiomatic theory of probability. Then it appears practically justifiable to assume that as a result of an (infinite) experiment only random sequences appear.

Von Mises's notion of infinite random sequences of 0's and 1's (collective) essentially appeals to the idea that no gambler, making a fixed number of wagers of 'heads,' at fixed odds [say p versus $1-p$] and in fixed amounts, on the flips of a coin [with bias p versus $1-p$], can have profit in the long run from betting according to a system instead of betting at random. Says Church: "this definition [below] ... while clear as to general intent, is too inexact in form to serve satisfactorily as the basis of a mathematical theory."

Definition 1.9.1 An infinite sequence $a_1, a_2, \ldots$ of 0's and 1's is a random sequence in the special meaning of *collective* if the following two conditions are satisfied:

1. Let f_n be the number of 1's among the first n terms of the sequence. Then

$$\lim_{n\to\infty} \frac{f_n}{n} = p, \text{ for some } p,\ 0 < p < 1.$$

2. A *place-selection rule* is a partial function ϕ from the finite binary sequences to the values 0 and 1 with the purpose of selecting one after another those indices n for which $\phi(a_1 a_2 \dots a_{n-1}) = 1$. We require (1), with the same limit p, also for every infinite subsequence

$$a_{n_1} a_{n_2} \dots$$

obtained from the sequence by some *admissible* place-selection rule. (We have not yet formally stated which place-selection rules are admissible.)

The existence of a relative frequency limit is a strong assumption. Empirical evidence from long runs of dice throws in gambling houses or with death statistics in insurance mathematics suggests that the relative frequencies are *apparently convergent*. But clearly, no empirical evidence can be given for the existence of a definite limit for the relative frequency. However long the test run, in practice it will always be finite, and whatever the apparent behavior in the observed initial segment of the run, it is always possible that the relative frequencies keep oscillating forever if we continue.

The second condition ensures that no strategy using an admissible place-selection rule can select a subsequence that allows different odds for gambling than a subsequence that is selected by flipping a fair coin. For example, let a casino use a coin with probability $p = \frac{1}{4}$ of coming up heads and a payoff for heads equal to three times the payoff for tails. This 'law of excluded gambling strategy' says that a gambler betting in fixed amounts cannot make more profit in the long run betting according to a system than from betting at random.

"In everyday language we call random those phenomena where we cannot find a regularity allowing us to predict precisely their results. Generally speaking, there is no ground to believe that random phenomena should possess any definite probability. Therefore, we should distinguish between randomness proper (as absence of any regularity) and stochastic randomness (which is the subject of probability theory). There emerges the problem of finding reasons for the applicability of the mathematical theory of probability to the real world." [Kolmogorov]

Intuitively, we can distinguish between sequences that are irregular and do not satisfy the regularity implicit in stochastic randomness, and sequences that are irregular but do satisfy the regularities associated with

stochastic randomness. Formally, we will distinguish the second type from the first type by whether a certain complexity measure of the initial segments goes to a definite limit. The complexity measure referred to is the length of the shortest description of the prefix (in the precise sense of Kolmogorov complexity) divided by its length. It will turn out that almost all infinite strings are irregular of the second type and satisfy all regularities of stochastic randomness.

"In applying probability theory we do not confine ourselves to negating regularity, but from the hypothesis of randomness of the observed phenomena we draw definite positive conclusions." [Kolmogorov]

Considering the sequence as fair coin tosses with $p = \frac{1}{2}$, the second condition in Definition 1.9.1 says that there is no *strategy* ϕ (*law of excluded gambling strategy*) that ensures that a player betting at fixed odds and in fixed amounts on the tosses of the fair coin will make infinite gain. That is, no advantage is gained in the long run by following some system, such as betting 'heads' after each run of seven consecutive tails, or (more plausibly) by placing the nth bet 'heads' after the appearance of $n+7$ tails in succession. According to von Mises, the above conditions are sufficiently familiar and an uncontroverted empirical generalization to serve as the basis of an applicable calculus of probabilities.

Example 1.9.1 It turns out that the naive mathematical approach to a concrete formulation, admitting simply *all* partial functions, comes to grief as follows: Let $a = a_1a_2\ldots$ be any collective. Define ϕ_1 as $\phi_1(a_1\ldots a_{i-1}) = 1$ if $a_i = 1$, and undefined otherwise. But then $p = 1$. Defining ϕ_0 by $\phi_0(a_1\ldots a_{i-1}) = b_i$, with b_i the complement of a_i, for all i, we obtain by the second condition of Definition 1.9.1 that $p = 0$. Consequently, if we allow functions like ϕ_1 and ϕ_0 as strategies, then von Mises's definition cannot be satisfied at all. ◇

In the 1930s, Abraham Wald proposed to restrict the a priori admissible ϕ to any arbitrary fixed countable set of functions. Then collectives do exist. But which countable set? In 1940, Alonzo Church proposed to choose a set of functions representing computable strategies. According to Church's thesis, Section 1.7, this is precisely the set of *recursive functions*. With recursive ϕ, not only is the definition completely rigorous, and random infinite sequences do exist, but moreover they are abundant, since the infinite random sequences with $p = \frac{1}{2}$ form a set of measure one. From the existence of random sequences with probability $\frac{1}{2}$, the existence of random sequences associated with other probabilities can be derived. Let us call sequences satisfying Definition 1.9.1 with recursive ϕ *Mises–Wald–Church random*. That is, the involved *Mises–Wald–Church place-selection rules* consist of the partial recursive functions.

Appeal to the theorem of Wald above yields as a corollary that the set of Mises–Wald–Church random sequences associated with any fixed probability has the cardinality of the continuum. Moreover, each Mises–Wald–Church random sequence qualifies as a normal number. (A number is *normal* if each digit of the base, and each block of digits of any length, occurs with equal asymptotic frequency.) Note, however, that not every normal number is Mises–Wald–Church random. This follows, for instance, from Champernowne's sequence (or number),

$$0.12345678910111213141516171819 20\ldots,$$

due to D.G. Champernowne, which is normal in the scale of 10 and whose ith digit is easily calculated from i. The definition of a Mises–Wald–Church random sequence implies that its consecutive digits cannot be effectively computed. Thus, an existence proof for Mises–Wald–Church random sequences is necessarily nonconstructive.

Unfortunately, the von Mises–Wald–Church definition is not yet good enough, as was shown by J. Ville in 1939. There exist sequences that satisfy the von Mises–Wald–Church definition of randomness, with limiting relative frequency of ones of $\frac{1}{2}$, but nonetheless have the property

$$\frac{f_n}{n} \geq \frac{1}{2} \text{ for all } n.$$

The probability of such a sequence of outcomes in random flips of a fair coin is zero. Intuition: if you bet 1 all the time against such a sequence of outcomes, then your accumulated gain is always positive! Similarly, other properties of randomness in probability theory such as the law of the iterated logarithm do not follow from the von Mises–Wald–Church definition.

For a better understanding of the problem revealed by Ville, and its subsequent solution by P. Martin-Löf in 1966, we look at some aspects of the methodology of probability theory. Consider the sample space of all one-way infinite binary sequences generated by fair coin tosses. We call a sequence 'random' if it is 'typical.' It is not 'typical,' say 'special,' if it has a particular distinguishing property. An example of such a property is that an infinite sequence contains only finitely many ones. There are infinitely many such sequences. But the probability that such a sequence occurs as the outcome of fair coin tosses is zero. 'Typical' infinite sequences will have the converse property, namely, they contain infinitely many ones.

In fact, one would like to say that 'typical' infinite sequences will have *all converse properties* of the properties that can be enjoyed by 'special' infinite sequences. This is formalized as follows: If a particular property, such as containing infinitely many occurrences of ones (or zeros), the

law of large numbers, or the law of the iterated logarithm, has been shown to have probability one, then one calls this a *law of randomness*. A sequence is 'typical,' or 'random,' if it satisfies all laws of randomness.

But now we are in trouble. Since *all* complements of singleton sets in the sample space have probability one, it follows that the intersection of all sets of probability one is empty. Thus, there are no random infinite sequences!

Martin-Löf, using ideas related to Kolmogorov complexity, succeeded in defining random infinite sequences in a manner that is free of such difficulties. He observed that all laws that are proven in probability theory to hold with probability one are effective (as defined in Section 1.7). That is, we can effectively test whether a particular infinite sequence does not satisfy a particular law of randomness by effectively testing whether the law is violated on increasingly long initial segments.

The natural formalization is to identify the effective test with a partial recursive function. This suggests that one ought to consider not the intersection of all sets of measure one, but only the intersection of all sets of measure one with recursively enumerable complements. (Such a complement set is expressed as the union of a recursively enumerable set of cylinders). It turns out that this intersection has again measure one. Hence, almost all infinite sequences satisfy all effective laws of randomness with probability one. This notion of infinite random sequences is related to infinite sequences of which all finite initial segments have high Kolmogorov complexity.

The notion of randomness satisfied by both the Mises–Wald–Church collectives and the Martin-Löf random infinite sequences is roughly that *effective tests* cannot detect regularity. This does not mean that a sequence may not exhibit regularities that cannot be effectively tested. Collectives generated by nature, as postulated by von Mises, may very well always satisfy stricter criteria of randomness. Why should collectives generated by quantum-mechanical phenomena care about mathematical notions of computability? Again, satisfaction of all effectively testable prerequisites for randomness is some form of regularity. Maybe nature is more lawless than adhering strictly to regularities imposed by the statistics of randomness.

Until now the discussion has centered on infinite random sequences where the randomness is defined in terms of limits of relative frequencies. However,

"The frequency concept based on the notion of *limiting frequency* as the number of trials increases to infinity does not contribute anything to substantiate the application of the results of probability theory to real practical problems where we always have to deal with a finite number of trials." [Kolmogorov]

The practical objection against both the relevance of considering infinite sequences of trials and the existence of a relative frequency limit is

concisely put in J.M. Keynes's famous phrase, "In the long run we shall all be dead." It seems more appealing to try to define randomness for finite strings first, and only then define random infinite strings in terms of randomness of initial segments.

The approach of von Mises to define randomness of infinite sequences in terms of *unpredictability* of continuations of finite initial sequences under certain laws (like recursive functions) did not lead to satisfying results. Although certainly inspired by the random sequence debate, the introduction of Kolmogorov complexity marks a definite shift of point of departure, namely, to define randomness of sequences by the fact that no program from which an initial segment of the sequence can be computed is significantly shorter than the initial segment itself, rather than that no program can predict the next elements of the sequence.

Finite sequences that cannot be effectively described by a significantly shorter description than their literal representation are called random. Our aim is to characterize random infinite sequences as sequences of which all initial finite segments are random in this sense (Section 3.6). A related approach characterizes random infinite sequences as sequences all of whose initial finite segments pass all effective randomness tests (Section 2.5).

Initially, before the idea of complexity, Kolmogorov proposed a close analogy to von Mises's notions in the finite domain. Consider a generalization of place-selection rules insofar as the selection of a_i can depend on a_j with $j > i$ [A.N. Kolmogorov, *Sankhyā*, Series A, 25(1963), 369–376]. Let Φ be a finite set of such generalized place-selection rules. Kolmogorov suggested that an arbitrary finite binary sequence a of length $n \geq m$ can be called (m, ϵ)-random with respect to Φ if there exists some p such that the relative frequency of the 1's in the subsequences $a_{i_1} \ldots a_{i_r}$ with $r \geq m$, selected by applying some ϕ in Φ to a, all lie within ϵ of p. (We discard ϕ that yield subsequences shorter than m.) Stated differently, the relative frequency in this finite subsequence is approximately (to within ϵ) invariant under any of the methods of subsequence selection that yield subsequences of length at least m. Kolmogorov has shown that if the cardinality of Φ satisfies

$$d(\Phi) \leq \frac{1}{2} e^{2m\epsilon^2(1-\epsilon)},$$

then for any p and any $n \geq m$ there is some sequence a of length n that is (m, ϵ)-random with respect to Φ.

Exercises

1.9.1. [08] Consider the sequence 101001000100001000001 ..., that is, increasing subsequences of 0's separated by single 1's. What is the limiting relative frequency of 1's and 0's? Is this sequence a collective?

1.9.2. [15] Suppose we are given a coin with an unknown bias: the probability of heads is p and of tails is $(1 - p)$, p some unknown real

number $0 < p < 1$. Can we use this coin to simulate a perfectly fair coin?

Comments. The question asks for an effective construction of a collective of 0's and 1's with limiting frequency $\frac{1}{2}$ from a collective of 0's and 1's with unknown limiting frequency p. Source: J. von Neumann, Various techniques used in connection with random digits, in: *Collected Works*, Vol. V, A.H. Traub, ed., Macmillan, 1963; W. Hoeffding and G. Simons, *Ann. Math. Statist.*, 41(1970), 341–352; T.S. Ferguson, *Ann. Math. Statist.*, 41(1970), 352–362; P. Elias, *Ann. Math. Statist.*, 43(1972), 865–870.

1.9.3. [15] Suppose the sequence $a_1a_2 \ldots a_i \ldots$ has all the properties of a collective with $p = \frac{1}{2}$.

(a) The limiting relative frequency of the subsequence 01 equals the limiting frequency of subsequence 11. Compute these limiting relative frequencies.

(b) Form a new sequence $b_1b_2 \ldots b_i \ldots$ defined by $b_i = a_i + a_{i+1}$ for all i. Then the b_i's are 0, 1, or 2. Show that the limiting relative frequencies of 0, 1, and 2 are $\frac{1}{4}$, $\frac{1}{2}$, and $\frac{1}{4}$, respectively.

(c) The new sequence constructed in Item (b) satisfies requirement (1) of a collective: the limiting relative frequencies constituent elements of the sample space $\{0, 1, 2\}$ exist. Prove that it does *not* satisfy requirement (2) of a collective.

(d) Show that the limiting relative frequency of the subsequence 21 will generally differ from the relative frequency of the subsequence 11.

Comments. Hint for Item (c): show that the subsequences 02 and 20 do not occur. Use this to select effectively a subsequence of $b_1b_2 \ldots b_i \ldots$ with limiting relative frequency of 2's equal to zero. This phenomenon was first observed by M. von Smoluchowski [*Sitzungsber. Wien. Akad. Wiss., Math.-Naturw. Kl., Abt. IIa*, 123(1914) 2381–2405; 124(1915) 339–368] in connection with Brownian motion, and called probability 'after-effect.' Also: R. von Mises, *Probability, Statistics, and Truth*, Macmillan, 1933.

1.9.4. [10] Given an infinite binary string that is a collective with limiting frequency of 1's equal to $p = \frac{1}{2}$, show how to construct a collective over symbols 0, 1, and 2 (a ternary collective) with equal limiting frequencies of $\frac{1}{3}$ for the number of occurrences of 0's, 1's, and 2's.

Comments. Hint: use Exercise 1.9.2.

1.9.5. [M40] Let $\omega_1\omega_2 \ldots$ be an infinite binary sequence, and let $f_n = \omega_1 + \omega_2 + \cdots + \omega_n$.

(a) A sequence ω is said to satisfy the *infinite recurrence law* if $f_n = \frac{1}{2}n$ infinitely often. It can be shown that the set of infinite binary sequences having the infinite recurrence property has measure one in the set of all infinite binary sequences with respect to the usual binary measure. Show that there are infinite binary sequences that are Mises–Wald–Church random satisfying $f_n \geq \frac{1}{2}n$ for all n.

(b) Show that there are infinite binary sequences that are Mises–Wald–Church random and satisfy $\limsup_{n\to\infty}(\sum_{i=1}^{n}\omega_i - \frac{1}{2}n)/\sqrt{n \ln\ln n} > 1/\sqrt{2}$ (the violate the law of the iterated logarithm), Exercise 1.10.5.

Comments. Since Items (a) and (b) are satisfied with probability zero in the set of all infinite binary strings, we can conclude that Mises–Wald–Church random strings do not satisfy all laws of probability that hold with probability one (the laws of randomness). Source: J. Ville, *Étude Critique de la Concept de Collectif*, Gauthier-Villars, 1939.

1.9.6. [32] What happens if we restrict our set of admissible selection functions to those computable by finite-state machines instead of Turing machines? First we need some definitions. Let $\omega = \omega_1\omega_2\ldots$ be an infinite binary string. For each $m = 0, 1, 00, 01, \ldots$, let f_n be the number of occurrences of m in $\omega_{1:n}$. We say that ω is *k-distributed* if $\lim_{n\to\infty} f_n/n = 1/2^{l(m)}$ for all m with $l(m) \leq k$. We say that ω is *∞-distributed,* or *normal,* if it is k-distributed for all integers k.

A *finite-state place-selection rule* ϕ is a function computed by a finite-state machine with value 0 or 1 for each finite binary sequence. Given an infinite sequence ω, ϕ determines a subsequence $\omega_{i_1}\omega_{i_2}\ldots$ by selecting one after another the indices n for which $\phi(\omega_1\omega_2\ldots\omega_{n-1}) = 1$. (Formally, in terms of the definition of a Turing machine in Section 1.7, we can think of ϕ as being computed by a Turing machine $T : Q\times A \to S\times Q$ with $T(\cdot, d) = (R, \cdot)$ for all $d = 0, 1$ and $T(\cdot, B) = (a, \cdot)$ for some $a = 0, 1$.) Let $\omega_{i_1}\omega_{i_2}\ldots$ be the subsequence of ω selected by ϕ, and let c_n be the number of ones in the first n bits. Assuming that ϕ has infinitely many values 1 on prefixes of ω, define $\limsup_{n\to\infty} c_n/n$ as the *prediction ratio* of ϕ for ω. Finally, a finite-state prediction function ϕ is a *predictor* for ω if its prediction ratio is greater than $\frac{1}{2}$.

(a) Show that every ω that is k-distributed but not $(k+1)$-distributed has a $(k+1)$-state predictor.

(b) Show that there are no predictors for ∞-distributed ω.

(c) Show that the subsequence selected from an ∞-distributed ω by a finite-state selection function is again ∞-distributed.

(d) Show that there are ∞-distributed sequences that can be predicted by stronger models of computation such as Turing machines. Conclude that there are ∞-distributed sequences that are not random in the sense

of von Mises–Wald–Church. (Hint: use Champernowne's sequence presented in the main text.)

Comments. Since predictors are machines that, in the long run, have some success in making correct predictions on ω, we can say that ω appears *random* to ϕ if ϕ is not a predictor of ω. Then, ∞-distributed sequences are precisely the sequences that appear random to all finite-state selection functions. In terms of gambling, let a gambler pay \$1.00 for each prediction he makes and receive \$2.00 for each correct prediction. If the sequence supplied by the house is ∞-distributed, and the gambler makes unboundedly many predictions, then no matter what finite-state selection function he uses, the limit superior of the ratio (paid \$)/(received \$) goes to one. Source: V.N. Agafonov, *Soviet Math. Dokl.*, 9(1968), 324–325 (English transl.); C.P. Schnorr and H. Stimm, *Acta Informatica*, 1(1972), 345–359; and, apparently independently, M.G. O'Connor, *J. Comput. System Sci.* 37(1988), 324–336. See also: T. Kamae, *Israel J. Math.*, 16(1973), 121–149; T. Kamae and B. Weiss, *Israel J. Math.*, 21(1975), 101–111.

1.9.7. [33] Investigate related problems as in Exercise 1.9.6 by replacing finite-state machines (that is, regular languages) by slightly more complex languages such as deterministic one-counter languages or linear languages. Show that there are languages of both types such that selection according to them does not preserve normality (∞-distributedness), and that in fact, for both types of languages it is possible to select a constant sequence from a normal one.

Comments Source: W. Merkle, J. Reimann, *Theor. Comput. Systems*, 39(2006), 685–697.

1.9.8. [O35] Investigate the problems as in Exercises 1.9.6 and 1.9.7, for push-down automata, time- or space-bounded classes, and primitive recursive functions.

1.10 Prediction and Probability

The question of quantitative probability based on complexity was first raised and treated by R.J. Solomonoff, in an attempt to obtain a completely general theory of inductive reasoning. Let us look at some predecessors in this line of thought.

The so-called *weak law of large numbers,* formulated by Jacob Bernoulli (1654–1705) in his *Ars Conjectandi*, published posthumously in 1713, states that if an experiment with probability of success p is repeated n times, then the proportion of successful outcomes will approach p for large n. Such a repetitive experiment is called a sequence of *Bernoulli trials* generated by a $(p, 1-p)$ *Bernoulli process*, and the generated sequence of outcomes is called a *Bernoulli sequence*.

Thomas Bayes (1702–1761), in "An essay towards solving a problem in the doctrine of chances" [*Philos. Trans.*, London, 53(1763), 376–398, and 54(1764), 298–310], suggested the 'inverse of Bernoulli's problem.' The resulting method, sometimes referred to as *inverse probability*, was further analyzed by P.S. Laplace, who also attached Bayes's name to it. In Bayesian approaches it is assumed that there is some true, or a priori (prior), distribution of probabilities over objects. Then an object with unknown probability p is drawn. Provided with a (nonempirical) prior probability, together with empirical data and the probabilistic model of these data, Bayes's rule supplies a way to calculate a *posterior* or *inferred* probability distribution. We then can give a numerical estimate for p, for example, by choosing the maximum posterior probability.

The formal statement of Bayes's rule was given in Section 1.6. The procedure is most easily explained by example. Suppose we have an urn containing a large number of dice with the faces numbered 1 through 6. Each die has a (possibly different) unknown probability p of casting 6, which may be different from the $\frac{1}{6}$ that it is for a true die. A die is drawn from the urn and cast n times in total, producing the result 6 in m of those casts. Let $P(X = p)$ be the probability of drawing a die with attribute p from the urn. This $P(X = p)$ is the prior distribution. In von Mises's interpretation, if we repeatedly draw a die from the urn, with replacement, then the relative frequency with which a die with given value of p appears in these drawings has the limiting value $P(X = p)$. The probability of obtaining m outcomes 6 in n throws of a die with attribute p is

$$P(Y = m|n, p) = \binom{n}{m} p^m (1 - p)^{n-m},$$

the number of ways to select m items from n items, multiplied by the probability of m successes and $(n-m)$ failures. Hence, the probability of drawing a die with attribute p and subsequently throwing m outcomes 6 in n throws with it is the product $P(X = p)\ P(Y = m|n, p)$.

For the case under discussion, Bayes's problem consists in determining the probability of m outcomes 6 in n casts being due to a die with a certain given value of p. The answer is given by the posterior, or inferred, probability distribution

$$P(X = p|n, m) = \frac{P(X = p)P(Y = m|n, p)}{\sum_p P(X = p)P(Y = m|n, p)},$$

the sum taken over all values of attribute p. If we repeat this experiment many times, then the limiting value of the probability of drawing a die with attribute value p, given that we throw m 6's out of n, is $P(X = p|n, m)$.

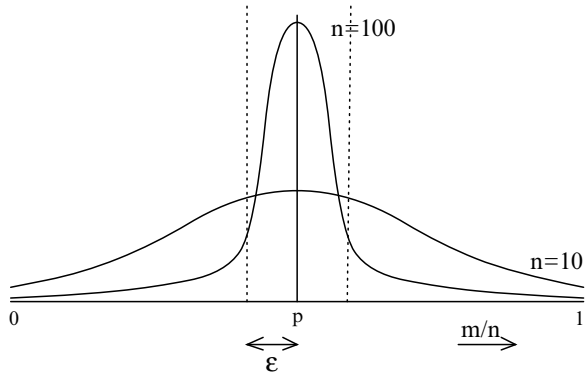


FIGURE 1.2. Inferred probability for increasing n

The interesting feature of this approach is that it quantifies the intuition that if the number of trials n is small, then the inferred distribution $P(X = p|n, m)$ depends heavily on the prior distribution $P(X = p)$. However, if n is large, then *irrespective* of the prior $P(X = p)$, the inferred probability $P(X = p|n, m)$ condenses more and more around $m/n = p$. To analyze this, we consider $P(X = p|n, m)$ as a continuous distribution with fixed n. Clearly, $P(X = p|n, m) = 0$ for $m < 0$ and $m > n$. Let $\epsilon > 0$ be some constant.

Consider the area under the tails of $P(Y = m|n, p)$ for $m \leq (p - \epsilon)n$ and $m \geq (p + \epsilon)n$ (the area such that $|p - m/n| \geq \epsilon$). Whatever ϵ we choose, for each $\delta > 0$ we can find an n_0 such that this area is smaller than δ for all $n \geq n_0$. This can be shown in several ways. We show this by appealing to a result that will be used several times later on.

The probability of m successes out of n independent trials with probability p of success is given by the *binomial distribution*

$$P(Y = m|n, p) = \binom{n}{m} p^m (1 - p)^{n-m}. \tag{1.6}$$

The deviation ϵ (where $0 \leq \epsilon$) from the average number of successes np in n experiments is analyzed by estimating the combined tail probability

$$P(|m - np| > \epsilon pn) = \sum_{|m-np|>\epsilon pn} \binom{n}{m} p^m (1 - p)^{n-m}$$

of the binomial distribution, Figure 1.2. We give a variant of the classical estimate and omit the proof.

Lemma 1.10.1 **(Chernoff bounds)** *Assume the notation above. For* $0 \leq \epsilon \leq 1$,

$$P(|m - pn| > \epsilon pn) \leq 2e^{-\epsilon^2 pn/3}. \tag{1.7}$$

Each tail separately can be bounded by half of the right-hand side.

This shows that for every $\epsilon > 0$ (and $\epsilon \leq 1$),

$$\lim_{n\to\infty} \int_{(p-\epsilon)n}^{(p+\epsilon)n} P(X = p|n, m)\, d\, m = 1. \tag{1.8}$$

Example 1.10.1 Let us give a numerical example. Let p take values $0.1, 0.2, \ldots, 0.9$ with equal probability $P(X = 0.1) = P(X = 0.2) = \cdots = P(X = 0.9) = \frac{1}{9}$. Let $n = 5$ and $m = 3$. Then the inferred probabilities are $P(X = p|5, 3) = 0.005$ for $p = 0.1$, 0.031 for $p = 0.2$, up to 0.21 for $p = 0.6$, and down again to 0.005 for $p = 0.9$, the combined probabilities summing up to 1. If we pick a die and do no experiments, then the probability that it is from any particular category is $\frac{1}{9} \approx 0.11$. If, however, we know already that it has had three throws of 6 out of five throws, then the probability that it belongs to category $p = 0.1$ becomes smaller than 0.11, namely, 0.005, and the probability that it belongs to category $p = 0.6$ increases to 0.21. In fact, the inferred probability that $0.5 \leq p \leq 0.7$ is 0.59, while the inferred probability for the other six p values is only 0.41.

Consider the same prior distribution $P(X = p)$ but set $n = 500$ and $m = 300$. Then, the inferred probability for $p = 0.6$ becomes $P(X = 0.6|500, 300) = 0.99995$. This means that it is now almost certain that a die that throws 60 percent 6's has $p = 0.6$.

We have seen that the probability of inference depends on (a) the prior probability $P(X = p)$ and (b) the observed results from which the inference is drawn. We have varied (b), but what happens if we start from a different $P(X = p)$? Let the new prior distribution $P(Y = p)$ be $P(Y = 0.i) = i/45$, $i = 1, \ldots, 9$. Then the corresponding inferred probability for $n = 5$ and $m = 3$ is $P(Y = 0.1|5, 3) = 0.001$, $P(Y = 0.2|5, 3) = 0.011$, up to $P(Y = 0.6|5, 3) = 0.21$, and finally $P(Y = 0.9|5, 3) = 0.07$. For this small sequence, these values are markedly different from the previous X values, although the highest values are still around $p = 0.6$. But if we now increase the number of observations to $n = 500$ with the same relative frequency of success $m = 300$, then the resulting Y values correspond to the X values but for negligible differences. ◇

Equation 1.8 shows that as the number of trials increases indefinitely, the limiting value of the observed relative frequency of success in the trials approaches the true probability of success, with probability one. This holds no matter what prior distribution the die was selected from. In case the initial probabilities of the events are unknown, Bayes's rule is a correct tool to make inferences about the probability of events from frequencies based on many observations. For small sequences of observations, however, we need to know the initial probability to make justified inferences.

Solomonoff addresses precisely this issue. Suppose we are faced with a problem we have to solve in which there has been much experience. Then either we know outright how to solve it, or we know the frequency of success for different possible methods. However, if the problem has never occurred before, or only a small number of times, and the prior distribution is unknown, as it usually is, the inference method above is undefined or of poor accuracy. To solve this quandary Solomonoff proposes a *universal* prior probability. The idea is that this universal prior probability serves in a well-defined sense as well as the *true* prior probability, provided this true prior probability is computable in the sense of Section 1.7.

Solomonoff argues that all inference problems can be cast in the form of extrapolation from an ordered sequence of binary symbols. A principle to enable us to extrapolate from an initial segment of a sequence to its continuation will either require some hypothesis about the source of the sequence or a definition of what we mean by extrapolation. Two popular and useful metaphysical principles for extrapolation are those of simplicity (Occam's razor, commonly attributed to the fourteenth-century scholastic philosopher William of Ockham, but emphasized about twenty years before Ockham by John Duns Scotus), and indifference. The principle of simplicity asserts that the simplest explanation is the most reliable. The principle of indifference asserts that in the absence of grounds enabling us to choose between explanations we should treat them as equally reliable. We do not supply any details here, because we shall extensively return to this matter in Chapter 4 and Chapter 5.

Exercises

1.10.1. [25] Let an experiment in which the outcomes are 0 or 1 with fixed probability p for outcome 1 and $1-p$ for outcome 0 be repeated n times. Such an experiment consists of a sequence of *Bernoulli trials* generated by a $(p, 1-p)$ Bernoulli process, see page 59.

Show that for each $\epsilon > 0$ the probability that the number S_n of outcomes 1 in the first n trials of a single sequence of trials satisfies $n(p-\epsilon) < S_n < n(p+\epsilon)$ goes to 1 as n goes to infinity.

Comments. This is J. Bernoulli's law of large numbers [*Ars Conjectandi*, Basel, 1713, Part IV, Ch. 5, p. 236], the so-called *weak law of large numbers*. This law shows that with great likelihood in a series of n trials the proportion of successful outcomes will approximate p as n grows larger. The following interpretation of the weak law is false: "if Alice and Buck toss a perfect coin n times, then we can expect Alice to be in the lead roughly half of the time, regardless of who wins." It can be shown that if Buck wins, then it is likely that he has been in the lead for practically the whole game. Thus, contrary to common belief, the time

average of S_n $(1 \le n \le m)$ over an individual game of length m has nothing to do with the so-called ensemble average of the different S_n's associated with all possible games (the ensemble consisting of 2^n games) at a given moment n, which is the subject of the weak law. Source: W. Feller, *An Introduction to Probability Theory and Its Applications*, Vol. 1, Wiley, 1968.

1.10.2. [30] In an infinite sequence of outcomes generated by a $(p, 1-p)$ Bernoulli process, let $A_1, A_2, \ldots$ be an infinite sequence of events each of which depends only on a finite number of trials in the sequence. Denote the probability of A_k occurring by P_k. (A_k may be the event that k consecutive 1's occur between the 2^kth trial and the 2^{k+1}th trial. Then $P_k \le (2p)^k$.)

(a) Prove that if $\sum P_k$ converges, then with probability one only finitely many A_k occur.

(b) Prove that if the events A_k are mutually independent, and if $\sum P_k$ diverges, then with probability one infinitely many A_k occur.

Comments. These two assertions are known as the *Borel–Cantelli Lemmas*. Source: W. Feller, *Ibid.*

1.10.3. [M30] Prove the limit in Equation 1.8 associated with the condensation of the posterior probability.

Comments. This may be called the *inverse weak law of large numbers*, since it shows that we can infer with great certainty the original probability (drawn from an unknown distribution) by performing a single sequence of a large number of trials. Note that this is a different statement from the weak law of large numbers.

1.10.4. [M37] Consider one-way infinite binary sequences generated by a $(p, 1-p)$ Bernoulli process. Let S_n be as in Exercise 1.10.1.

(a) Show that for every $\epsilon > 0$, we have probability one that $|pn - S_n| < \epsilon n$ for *all* but finitely many n.

(b) Define the *reduced number of successes* $S_n^* = (S_n - pn)/\sqrt{np(1-p)}$. Prove the much stronger statement than Item (a) that with probability one, $|S_n^*| < \sqrt{2a \ln n}$ (where $a > 1$) holds for all but finitely many n.

Comments. Item (a) is a form of the *strong law of large numbers* due to F.P. Cantelli (1917) and G. Pólya (1921). Note that this statement is stronger than the weak law of large numbers. The latter says that S_n/n is likely to be near p, but does not say that S_n/n is bound to stay near p as n increases. The weak law allows that for infinitely many n, there is a k with $n < k < 2n$ such that $S_k/k < p - \epsilon$. In contrast, the strong law asserts that with probability one, $p - S_n/n$ becomes small

and remains small. Item (b) is due to A.N. Kolmogorov [*Math. Ann.*, 101(1929), 126–135]. Source: W. Feller, *Ibid.*

1.10.5. [M42] Consider a sequence generated by a $(p, 1-p)$ Bernoulli process. Show that $\limsup_{n\to\infty} S_n^*/\sqrt{2\ln\ln n} = 1$ with probability one.

Comments. For reasons of symmetry, $\liminf_{n\to\infty} S_n^*/\sqrt{2\ln\ln n} = -1$. This remarkable statement, known as the *law of the iterated logarithm* is due to A.I. Khintchin [*Fundamenta Mathematicae* 6(1924), 9–20] and was generalized by A.N. Kolmogorov [*Math. Ann.*, 101(1929), 126–135]. For an explanation of its profundity, implications, and applications see also W. Feller, *Ibid.*

1.10.6. [M33] Consider a Bernoulli processes with unknown probability p of a successful outcome. Assume that the prior probability of the bias p is uniformly distributed over the real interval $(0, 1)$. Prove that after m successful outcomes in n independent trials, the expectation of a successful outcome in the $(n+1)$th trial is given by $(m+1)/(n+2)$.

Comments. The above reduces to binary Bernoulli processes $(p, 1-p)$ with probability p of 'success' and probability $1-p$ of 'failure,' that is, independent flips of a coin with unknown bias p. This is P.S. Laplace's celebrated *law of succession*. Hint: The prior probability density $P(X = p)$ is uniform with $\int_{p=a}^{b} P(X = p) = b - a$ $(0 \le a \le b \le 1)$. The term $\Pr(Y = m|n, p) = \binom{n}{m} p^m (1-p)^{n-m}$ is the probability of the event of m successes in n trials with probability p of success. The probability of obtaining m successes in n trials at all is $\Pr(Y = m|n) = \int_{p=0}^{1} \binom{n}{m} p^m (1-p)^{n-m} \mathrm{d}p$. The requested expectation is the p-expectation of the posterior in Bayes's rule, that is, $\int_{p=0}^{1} p\ P(Y|n, p)\mathrm{d}p / P(Y = m|n)$. The integrals are beta functions; decompose these into gamma functions and use the relation of the latter to factorials. Source: P.S. Laplace, *A Philosophical Essay on Probabilities*, Dover, 1952. (Originally published in 1819. Translated from the 6th French edition.)

1.11 Information Theory and Coding

It seldom happens that a detailed mathematical theory springs forth in essentially final form from a single publication. Such was the case with information theory, which properly began only with the appearance of C.E. Shannon's paper "The mathematical theory of communication" [*Bell System Technical J.*, 27(1948), 379–423, 623–656]. In this theory we ignore the *meaning* of a message; we are interested *only* in the problem of *communicating* a message between a *sender* and a *receiver* under the assumption that the universe of possible messages is known to both the sender and the receiver.

This notion of information is a measure of one's freedom of choice when one selects a message. Given the choice of transmitting a message consisting of the contents of this entire book, and the message "let's get a beer," the information concerned is precisely one bit. Obviously this does not capture the information content of the individual object itself. Kolmogorov's intention for introducing algorithmic complexity is as a measure of the information content of individual objects.

We develop the basic ideas in a purely combinatorial manner. This is easier and more fundamental, suffices for our purpose, and does not need extra probabilistic assumptions. The set of possible messages from which the selection takes place is often called an *ensemble*. Information, according to Shannon, is an ensemble notion. For our purpose it is sufficient to consider only *countable* ensembles.

The *entropy* of a random variable X with outcomes in an ensemble S is the quantity $H(X) = \log d(S)$. This is a measure of the uncertainty in choice before we have selected a particular value for X, and of the information produced from the set if we assign a specific value to X. By choosing a particular message a from S, we remove the entropy from X by the assignment $X = a$ and produce or transmit *information* $I = \log d(S)$ by our selection of a. Since the information is usually measured in the number of bits I' needed to be transmitted from sender to receiver, $I' = \lceil \log d(S) \rceil$.

Example 1.11.1 The number of different binary strings $\bar{u}$ with $l(\bar{u}) = 2n + 1$ is 2^n. This gives an information content in each such message of $I = n$, and encoding in a purely binary system requires $I' = n$ bits. $\diamond$

Note that while a random variable X usually ranges over a finite set of alternatives, say $a, b, \ldots, c$, the derived theory is so general that it also holds if we let X range over a set of sequences composed from these alternatives, which may even be infinite.

If we have k independent random variables X_i, each of which can take n_i values, respectively, for $i = 1, 2, \ldots, k$, then the number of combinations possible is $n = n_1 n_2 \ldots n_k$, and the entropy is given by

$$H(X_1, X_2, \ldots, X_k) = \log n_1 + \log n_2 + \cdots + \log n_k = \log n. \tag{1.9}$$

Let us look at the efficiency with which an individual message consisting of a *sequence* $x_1 x_2 \ldots x_k$ of symbols, each x_i being a selection of a random variable X drawn from the same ensemble of s alternatives, can be transmitted. We use the derivation to motivate the formal definition of the entropy of a random variable. The *Morse code* used in telegraphy suggests the general idea. In, say, English, the frequency of use of the letter e is 0.12, while the frequency of the letter w is only 0.02. Hence,

a considerable saving on average encoded message length can result by encoding e by a shorter binary string than w.

Assume that the random variable X can take on the alternative values $\{x_1, x_2, \ldots, x_s\}$ and that x_i occurs k_i times in the message $x = x_1 x_2 \ldots x_k$, with $k_1 + k_2 + \cdots + k_s = k$. Under these constraints there is altogether an ensemble of

$$\binom{k}{k_1, k_2, \ldots, k_s} = \frac{k!}{k_1! k_2! \cdots ! k_s!}$$

possible messages of length k, one of which is x. In the combinatorial approach we define the entropy of an ensemble as the efficiency with which any message from this ensemble can be transmitted. To determine the actual message $x_1 x_2 \ldots x_k$, we must at least give its ordinal in the ensemble. To reconstruct the message it suffices to give first the ordinal k and the ordinal $(k_1, k_2, \ldots, k_s)$ of the ensemble in $(s+1)\log k$ bits, and then give the ordinal of the message in the ensemble. Therefore, we can transmit the message in $h(x)$ bits, with

$$\log \frac{k!}{k_1! k_2! \cdots ! k_s!} \leq h(x) \leq (s+1)\log k + \log \frac{k!}{k_1! k_2! \cdots ! k_s!}.$$

The *frequency* of each symbol x_i is defined as $p_i = k_i/k$. Recall the approximation $k \log k + O(k)$ for $\log(k!)$ from Stirling's formula, Exercise 1.5.4 on page 17. For fixed frequencies $p_1, p_2, \ldots, p_s$ and large k we obtain

$$h(x) \sim k \sum p_i \log \frac{1}{p_i},$$

the sum taken in the obvious way. In information-theoretic terminology it is customary to say that the messages are produced by a stochastic source that emits symbols x_i with given probabilities p_i. With abuse of terminology and notions, henceforth we use '*probability*' for 'frequency.' (Under certain conditions on the stochastic nature of the source this transition can be rigorously justified.)

Definition 1.11.1 Define the *entropy* of a random variable X with the given probabilities $P(X = x_i) = p_i$ by

$$H(X) = \sum p_i \log \frac{1}{p_i}, \tag{1.10}$$

and therefore

$$h(x) \sim kH(X). \tag{1.11}$$

Is there a coding method that actually achieves the economy in average message length implied by Equation 1.11? Clearly, we have to encode symbols with high probabilities as short binary strings and symbols with low probabilities as long binary strings.

Example 1.11.2 We explain the *Shannon–Fano code*. Suppose we want to map messages over a fixed alphabet to binary strings. Let there be n symbols (also called basic messages or source words). Order these symbols according to decreasing probability, say $N = \{1, 2, \ldots, n\}$ with probabilities $p_1, p_2, \ldots, p_n$. Let $P_r = \sum_{i=1}^{r-1} p_i$, for $r = 1, \ldots, n$. The binary code $E : N \to \{0,1\}^*$ is obtained by coding r as a binary number $E(r)$, obtained by truncating the binary expansion of P_r at length $l(E(r))$ such that

$$\log \frac{1}{p_r} \leq l(E(r)) < 1 + \log \frac{1}{p_r}.$$

This code is the *Shannon–Fano code*. It has the property that highly probable symbols are mapped to short code words and symbols with low probability are mapped to longer code words. Moreover,

$$2^{-l(E(r))} \leq p_r < 2^{-l(E(r))+1}.$$

Note that the code for the symbol r differs from all codes of symbols $r+1$ through n in one or more bit positions, since for all i with $r+1 \leq i \leq n$,

$$P_i \geq P_r + 2^{-l(E(r))}.$$

Therefore the binary expansions of P_r and P_i differ in the first $l(E(r))$ positions. This means that E is one-to-one, and it has an inverse: the decoding mapping E^{-1}. Even better, since no value of E is a prefix of any other value of E, the set of code words is a prefix-code. This means we can recover the source message from the code message by scanning it from left to right without look-ahead.

If H_1 is the average number of bits used per symbol of an original message, then $H_1 = \sum_r p_r l(E(r))$. Combining this with the previous inequality, we obtain

$$\sum_r p_r \log \frac{1}{p_r} \leq H_1 < \sum_r p_r \left(1 + \log \frac{1}{p_r}\right) = 1 + \sum_r p_r \log \frac{1}{p_r}.$$

From this it follows that $H_1 \sim H(X)$ for large n, with $H(X)$ the entropy per symbol of the source. ◇

Example 1.11.3 How much information can a random variable X convey about a random variable Y? Taking again a purely combinatorial approach, this notion

is captured as follows: If X ranges over S_X and Y ranges over S_Y, then we look at the set U of possible events $(X = x, Y = y)$ consisting of joint occurrences of event $X = x$ and event $Y = y$. If U does not equal the Cartesian product $S_X \times S_Y$, then this means that there is some dependency between X and Y. Considering the set $U_x = \{(x, y) : (x, y) \in U\}$ for $x \in S_X$, it is natural to define the *conditional entropy* of Y given $X = x$ as $H(Y|X = x) = \log d(U_x)$. This suggests immediately that the information given by $X = x$ about Y is

$$I(X = x : Y) = H(Y) - H(Y|X = x).$$

For example, if $U = \{(1, 1), (1, 2), (2, 3)\}$, so that $U \subseteq S_X \times S_Y$ with $S_X = \{1, 2\}$ and $S_Y = \{1, 2, 3, 4\}$, then $I(X = 1 : Y) = 1$ and $I(X = 2 : Y) = 2$.

In this formulation it is obvious that $H(X|X = x) = 0$, and that $I(X = x : X) = H(X)$. This approach amounts to the assumption of *uniform distribution* of the probabilities concerned. ◇

We can develop the generalization of Example 1.11.3, taking into account the frequencies or probabilities of the occurrences of the different values X and Y can assume. Let the *joint probability* $p(x, y)$ be defined as the probability of the joint occurrence of event $X = x$ and event $Y = y$. The *marginal probabilities* $p_1(x)$ and $p_2(y)$ are defined by $p_1(x) = \sum_y p(x, y)$ and $p_2(y) = \sum_x p(x, y)$ and are the probability of the occurrence of the event $X = x$ and the probability of the occurrence of the event $Y = y$, respectively. This leads to the following self-evident formulas for joint variables X, Y:

$$H(X, Y) = \sum_{x,y} p(x, y) \log \frac{1}{p(x, y)},$$
$$H(X) = \sum_x p_1(x) \log \frac{1}{p_1(x)},$$
$$H(Y) = \sum_y p_2(y) \log \frac{1}{p_2(y)},$$

where summation over x is taken over all outcomes of the random variable X and summation over y is taken over all outcomes of the random variable Y. In all of these equations the entropy quantity on the left-hand side achieves the maximum for equal probabilities on the right-hand side. One can show that

$$H(X, Y) \leq H(X) + H(Y), \tag{1.12}$$

with equality only in the case that X and Y are independent.

The conditional probability $p(y|x)$ of outcome $Y = y$ given outcome $X = x$ for random variables X and Y (not necessarily independent) is defined by

$$p(y|x) = \frac{p(x,y)}{\sum_y p(x,y)},$$

Section 1.6.2. This leads to the following analysis of the information in X about Y by first considering the *conditional entropy* of Y given X as the average of the entropy for Y for each value of X weighted by the probability of getting that particular value:

$$\begin{aligned} H(Y|X) &= \sum_x p_1(x) H(Y|X = x) \\ &= \sum_x p_1(x) \sum_y p(y|x) \log \frac{1}{p(y|x)} \\ &= \sum_{x,y} p(x,y) \log \frac{1}{p(y|x)}. \end{aligned}$$

The quantity on the left-hand side tells us how uncertain we are about the outcome of Y when we know an outcome of X. With

$$\begin{aligned} H(X) &= \sum_x p_1(x) \log \frac{1}{p_1(x)} \\ &= \sum_x \left(\sum_y p(x,y) \right) \log \frac{1}{\sum_y p(x,y)} \\ &= \sum_{x,y} p(x,y) \log \frac{1}{\sum_y p(x,y)}, \end{aligned}$$

and substituting the formula for $p(y|x)$, we obtain $H(X) = H(X,Y) - H(Y|X)$. Rewrite this expression as

$$H(X,Y) = H(X) + H(Y|X). \tag{1.13}$$

This can be interpreted as "the uncertainty of the joint event (X,Y) is the uncertainty of X plus the uncertainty of Y given X." Combining Equations 1.12, 1.13 gives $H(Y) \geq H(Y|X)$, which can be taken to imply that knowledge of X can never increase uncertainty of Y. In fact, uncertainty in Y will be decreased unless X and Y are independent. Finally, the *information* in the outcome $X = x$ about Y is defined as

$$I(X = x : Y) = H(Y) - H(Y|X = x). \tag{1.14}$$

Here the quantities $H(Y)$ and $H(Y|X = x)$ on the right-hand side of the equation are always equal to or less than the corresponding quantities under the uniform distribution we analyzed first. The values of

the quantities $I(X = x : Y)$ under the assumption of uniform distribution of Y and $Y|X = x$ versus any other distribution are not related by inequality in a particular direction. The equalities $H(X|X = x) = 0$ and $I(X = x : X) = H(X)$ hold under any distribution of the variables. Since $I(X = x : Y)$ is a function of outcomes of X, while $I(Y = y : X)$ is a function of outcomes of Y, we do not compare them directly. However, forming the expectation defined as

$$\mathbf{E}(I(X = x : Y)) = \sum_x p_1(x) I(X = x : Y),$$
$$\mathbf{E}(I(Y = y : X)) = \sum_y p_2(y) I(Y = y : X),$$

and combining Equations 1.13 and 1.14, we see that the resulting quantities are equal. Denoting this quantity by $I(X;Y)$ and calling it the *mutual information* in X and Y, we see that this information is *symmetric*:

$$I(X;Y) = \mathbf{E}(I(X = x : Y)) = \mathbf{E}(I(Y = y : X)). \tag{1.15}$$

The quantity $I(X;Y)$ symmetrically characterizes to what extent random variables X and Y are correlated. An inherent problem with probabilistic definitions (which is avoided by the combinatorial approach) is that although $\mathbf{E}(I(X = x : Y))$ is always positive. But for some probability distributions, $I(X = x : Y)$ can turn out to be negative—which definitely contradicts our naive notion of information content.

Example 1.11.4 Suppose we want to exchange the information about the outcome $X = x$ and it is known already that the outcome $Y = y$ is the case, that is, x has property y. Then we require (using the Shannon–Fano code, Example 1.11.2 on page 68 and Lemma 4.3.3 on page 274) about $\log 1/P(X = x|Y = y)$ bits to communicate x. On average, over the joint distribution $P(X = x, Y = y)$ we use $H(X|Y)$ bits, which is optimal by Shannon's noiseless coding theorem. In fact, exploiting the mutual information paradigm, the expected information $I(Y;X)$ that outcome $Y = y$ gives about outcome $X = x$ is the same as the expected information that $X = x$ gives about $Y = y$, and is never negative. Yet there may certainly exist *individual* y such that $I(Y = y : X)$ is negative. For example, we may have $\mathcal{X} = \{0, 1\}$, $\mathcal{Y} = \{0, 1\}$, $P(X = 1|Y = 0) = 1$, $P(X = 1|Y = 1) = \frac{1}{2}$, $P(Y = 1) = \epsilon$. Then $I(Y;X) = H(\epsilon, 1-\epsilon)$, whereas $I(Y = 1 : X) = H(\epsilon, 1-\epsilon) + \epsilon - 1$. For small ϵ, this quantity is smaller than 0. ◇

Example 1.11.5 **(Information inequality)** Writing Equation 1.15 out, we obtain

$$I(X;Y) = \sum_x \sum_y p(x,y) \log \frac{p(x,y)}{p_1(x)p_2(y)}. \tag{1.16}$$

Another way to express this is as follows: a well-known criterion for the difference between a given distribution $q_1(x)$ and another distribution $q_2(x)$ we want to compare it with, is the so-called *Kullback–Leibler divergence*

$$D(q_1 \parallel q_2) = \sum_x q_1(x) \log \frac{q_1(x)}{q_2(x)}. \tag{1.17}$$

It has the important property that

$$D(q_1 \parallel q_2) \geq 0, \tag{1.18}$$

with equality only iff $q_1(x) = q_2(x)$ for all x. This is called the *information inequality*. Thus, the mutual information is the Kullback–Leibler divergence between the joint distribution $p(x, y)$ and the product $p_1(x)p_2(y)$ of the two marginal distributions. If this quantity is 0, then $p(x, y) = p_1(x)p_2(y)$ for every pair x, y, which is the same as saying that X and Y are independent random variables. ◇

Example 1.11.6 **(Data processing inequality)** Is it possible to increase the mutual information between two random variables by processing the outcomes in some deterministic manner? The answer is negative: For every function T we have

$$I(X;Y) \geq I(X;T(Y)), \tag{1.19}$$

that is, mutual information between two random variables cannot be increased by processing their outcomes in any deterministic way. The same holds in an appropriate sense for randomized processing of the outcomes of the random variables. This fact is called the *data-processing inequality*. The reason why it holds is that Equation 1.15 on page 71 is expressed in terms of joint probabilities and marginal probabilities. Processing X or Y will not increase the value of the expression in the right-hand side of the latter equation. If the processing of the arguments just renames them in a one-to-one manner, then the expression keeps the same value. If the processing eliminates or merges arguments, then it is easy to check from the formula that the expression value does not increase. ◇

The development of this theory immediately gave rise to at least two different questions. The first observation is that the concept of information as used in the theory of communication is a probabilistic notion, which is natural for information transmission over communication channels. Nonetheless, as we have seen from the discussion, we tend to identify *probabilities* of messages with *frequencies* of messages in a sufficiently long sequence, which under some conditions on the stochastic source

can be rigorously justified. For instance, Morse code transmissions of English telegrams over a communication channel can be validly treated by probabilistic methods even if we (as is usual) use empirical frequencies for probabilities. The great probabilist Kolmogorov remarks, "If something goes wrong here, the problem lies in the vagueness of our ideas of the relation between mathematical probability theory and real random events in general." (See also the discussion about the foundations of probability in Section 1.9.)

The second observation is more important and is exemplified in Shannon's statement, "Messages have *meaning* [. . . however . . .] the semantic aspects of communication are irrelevant to the engineering problem." In other words, can we answer a question such as, "what is the information in this book" by viewing it as an element of a set of possible books with a probability distribution on it? Or that the individual sections in this book form a random sequence with stochastic relations that damp out rapidly over a distance of several pages? And how to measure the quantity of hereditary information in biological organisms, as encoded in DNA? Again there is the possibility of seeing a particular form of animal as one of a set of possible forms with a probability distribution on it. This seems to be contradicted by the fact that the calculation of all possible life forms in existence at any one time on earth would give a ridiculously low figure like 2^{100}.

We are interested in a measure of information content of an *individual finite object*, and in the information conveyed about an individual finite object by another individual finite object. Here, we want the information content of an object x to be an attribute of x *alone*, and not to depend on, for instance, the means chosen to describe this information content. Making the natural restriction that the description method should be effective, the information content of an object should be recursively invariant (Section 1.7) among the different description systems. Pursuing this thought leads straightforwardly to Kolmogorov complexity.

1.11.1 Prefix-Codes

The main issues we treat here, apart from the basic definitions of prefix-codes, are Kraft's inequality, the noiseless coding theorem, and optimal and universal codes for infinite source-word alphabets.

We repeat some definitions of Section 1.4. Let D be any function $D : V^* \to \mathcal{N}$, where V is a finite code-word alphabet. It is common to use $V = \{0, 1\}$. The domain of D is the set of code words and the range of D is the set of source words. If $D(y) = x$, then y is a code word for source word x, and D is the decoding function. The set of all code words for source word x is the set $D^{-1}(x) = \{y : D(y) = x\}$ and $E = D^{-1}$ is the encoding relation (or encoding function if D^{-1} happens to be a function).

We may identify the source words (natural numbers) with their corresponding finite binary strings according to the enumeration in Equation 1.3. We often identify a code with its code-word alphabet (the domain of D).

We consider the natural extension of E to a relation $E' \subseteq \mathcal{N}^* \times V^*$ defined by

1. $E'(\epsilon) = \epsilon$; and
2. if x and y are in $\mathcal{N}$, then $E'(xy) = E(x)E(y)$.

We ignore the difference between E and E' and denote both by E.

Example 1.11.7 It is immediately clear that in general we cannot uniquely recover x and y from $E(xy)$. Let E be the identity mapping. Then we have $E(00)E(00) = 0000 = E(0)E(000)$. ◇

If we want to encode a sequence $x_1x_2 \dots x_n$ with $x_i \in \mathcal{N}$ $(i = 1, 2, \dots)$, then we call $x_1x_2 \dots x_n$ the *source sequence* and $y_1y_2 \dots y_n$ with $y_i = E(x_i)$ $(i = 1, 2, \dots)$ the *code sequence*. A code is *uniquely decodable* if for each source sequence of finite length, the code sequence corresponding to that source sequence is different from the code sequence corresponding to any other source sequence.

Example 1.11.8 In coding theory, attention is often restricted to the case where the source-word alphabet is finite, say the range of D equals $\{1, 2, \dots, n\}$. If there is a constant l_0 such that $l(y) = l_0$ for all code words y, then we call D a *fixed-length* code. It is easy to see that $l_0 \geq \log n$. For instance, in teletype transmissions the source has an alphabet of $n = 32$ letters, consisting of the 26 letters in the Latin alphabet plus 6 special characters. Hence, we need $l_0 = 5$ binary digits per source letter. In electronic computers we often use the fixed-length ASCII code with $l_0 = 8$. ◇

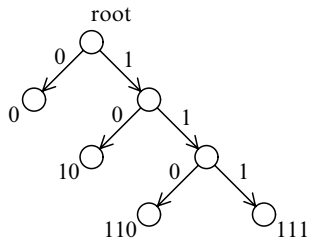


FIGURE 1.3. Binary tree for $E(1) = 0$, $E(2) = 10$, $E(3) = 110$, $E(4) = 111$

When the set of source words is infinite, say $\mathcal{N}$, we have to use *variable-length codes*. If no code word is the prefix of another code word, then each code sequence is uniquely decodable. This explains our interest in so-called prefix-codes.

Definition 1.11.2 A code is a *prefix-code* or *instantaneous code* if the set of code words is prefix-free (no code word is a prefix of another code word; Section 1.4).

In order to decode a code sequence of a prefix-code, we simply start at the beginning and decode one code word at a time. When we come to the end of a code word, we know it is the end, since no code word is the prefix of any other code word in a prefix-code. Every prefix-code is a uniquely decodable code. For example, if $E(1) = 0$, $E(2) = 10$, $E(3) = 110$, $E(4) = 111$ as in Figure 1.3, then 1421 is encoded as 0111100, which can be easily decoded from left to right in a unique way.

Not every uniquely decodable code satisfies the prefix condition. For example, if $E(1) = 0$, $E(2) = 01$, $E(3) = 011$, $E(4) = 0111$, then every code word is a prefix of every longer code word as in Figure 1.4. But unique decoding is trivial, since the beginning of a new code word is always indicated by a zero. Prefix-codes are distinguished from other uniquely decodable codes by the property that the end of a code word is always recognizable as such. This means that decoding can be accomplished without the delay of observing subsequent code words, which is why prefix-codes are also called instantaneous codes.

Example 1.11.9 A convenient graphical way to consider codes is by representing each code word as a node of a directed binary tree. If a node has two outgoing arcs, one of them is labeled with zero and the other with one. If a node has one outgoing arc, it is labeled either by zero or by one. The tree may be finite or infinite. There are also nodes without outgoing

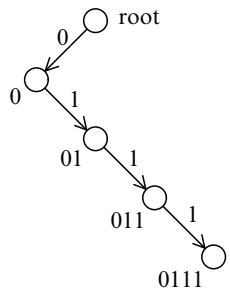


FIGURE 1.4. Binary tree for $E(1) = 0$, $E(2) = 01$, $E(3) = 011$, $E(4) = 0111$

arcs. These are called *external nodes*, and the nodes with outgoing arcs are called *internal nodes*. Each code word is represented by a node such that the consecutive zeros and ones on the branch from the root to that node form that code word. Clearly, for each code there is a tree with a node representing each code word, and such that there is only one node corresponding to each code word. We simplify each tree representation for a code such that it contains only nodes corresponding to code words, together with the intermediate nodes on a branch between root and code-word nodes. For E to be a prefix-code, it is a necessary and sufficient condition that in the simplified tree representation the nodes corresponding to code words be precisely the nodes without outgoing arcs. ◇

1.11.2 The Kraft Inequality

It requires little reflection to realize that prefix-codes waste potential code words, since the internal nodes of the representation tree cannot be used, and in fact, neither are the potential descendants of the external nodes used. Hence, we can expect that the code-word length exceeds the (binary) source-word length in prefix-codes. Quantification of this intuition leads to a precise constraint on code-word lengths for codes satisfying the prefix condition. This important relation is known as the *Kraft inequality* and is due to L.G. Kraft.

Theorem 1.11.1 *Let $l_1, l_2, \ldots$ be a finite or infinite sequence of natural numbers. There is a prefix-code with this sequence as lengths of its binary code words iff*

$$\sum_n 2^{-l_n} \leq 1.$$

Proof. (Only if) Recall the standard one-to-one correspondence between a finite binary string x and the interval $\Gamma_x = [0.x, 0.x + 2^{-l(x)})$ on the real line, Sections 1.4, 1.6, 2.5. Observe that the length of the interval corresponding to x is $2^{-l(x)}$. A prefix-code corresponds to a set of disjoint such intervals in $[0, 1)$, which proves that the inequality holds for prefix-codes.

(If) Suppose $l_1, l_2, \ldots$ are given such that the inequality holds. We can also assume that the sequence is nondecreasing. Choose disjoint adjacent intervals $I_1, I_2, \ldots$ of lengths $2^{-l_1}, 2^{-l_2}, \ldots$ from the left end of the interval $[0, 1)$. In this way, for each $n \geq 1$, the right end of I_n is $\sum_{i=1}^{n} 2^{-l_i}$. Note that the right end of I_n is the left end of I_{n+1}. Since the sequence of l_i's is nondecreasing, each interval I_n equals Γ_x for some binary string x of length $l(x) = l_n$. Take the binary string x corresponding to I_n as the nth code word. □

Example 1.11.10 Not every code of which the code-word lengths satisfy the inequality is a prefix-code. For instance, the code words 0, 00, and 11 satisfy the inequality, but 0 is a prefix of 00. ◇

Example 1.11.11 There is good reason for our emphasis on prefix-codes. Namely, Theorem 1.11.1 remains valid if we replace 'prefix-code' by 'uniquely decodable code.' This follows directly from the observation (proof omitted) that if a code has code-word lengths $l_1, l_2, \ldots$ and it is uniquely decodable, then the Kraft inequality must be satisfied.

This important fact means that every uniquely decodable code can be replaced by a prefix-code without changing the set of code-word lengths. Hence, all propositions concerning code-word lengths apply to uniquely decodable codes and to the subclass of prefix-codes. Accordingly, in looking for uniquely decodable codes with minimal average code-word length we can restrict ourselves to prefix-codes. ◇

1.11.3 Optimal Codes

A uniquely decodable code is *complete* if the addition of any new code word to its code-word alphabet results in a nonuniquely decodable code. It is easy to see that a code is complete iff equality holds in the associated Kraft inequality. Does completeness imply optimality in any reasonable sense? Given a source that produces source words from $\mathcal{N}$ according to probability distribution P, it is possible to assign code words to source words in such a way that any code word sequence is uniquely decodable, and moreover the average code-word length is minimal.

Definition 1.11.3 Let $D : \{0,1\}^* \to \mathcal{N}$ be a prefix-code with one code word per source word. Let $P(x)$ be the probability of source word x, and let the length of the code word for x be l_x. We want to minimize the number of bits we have to transmit. In order to do so, we must minimize the *average code-word length* $L_{D,P} = \sum_x P(x) l_x$. We define the *minimal average code-word length* as $L = \min\{L_{D,P} : D \text{ is a prefix-code}\}$. A prefix-code D such that $L_{D,P} = L$ is called an *optimal prefix-code* with respect to prior probability P of the source words.

The (minimal) average code length of an (optimal) code does not depend on the details of the set of code words, but only on the set of code-word lengths. It is just the expected code-word length with respect to the given distribution. C.E. Shannon discovered that the minimal average code-word length is about equal to the entropy of the source-word alphabet. This is known as the *noiseless coding theorem*. The adjective 'noiseless' emphasizes that we ignore the possibility of errors.

Theorem 1.11.2 *Let L and P be as above. If $H(P) = \sum_x P(x) \log 1/P(x)$ is the entropy, then*

$$H(P) \leq L \leq H(P) + 1.$$

Proof. First prove the upper bound $L \leq H(P)+1$. Let $l_x = \lceil \log 1/P(x) \rceil$ for $x = 1, 2, \ldots$. Therefore, $1 \geq \sum_x P(x) \geq \sum_x 2^{-l_x}$. By Kraft's inequality, Theorem 1.11.1, there exists a prefix-code with code-word lengths $l_1, l_2, \ldots$. Hence,

$$L \leq \sum_x P(x) l_x \leq \sum_x P(x) \left(\log \frac{1}{P(x)} + 1 \right) = H(P) + 1,$$

which finishes the proof of the second inequality $L \leq H(P) + 1$.

We now prove the lower bound $H(P) \leq L$. Let $L = \sum_x P(x) l_x$. Since $\sum_x P(x) = 1$ and $\sum_x (2^{-l_x} / \sum_x 2^{-l_x}) = 1$, by concavity of the logarithm function (see Equations 5.4 and 5.2.1 on page 353), we have

$$\sum_x P(x) \log \frac{1}{P(x)} \leq \sum_x P(x) \log \frac{\sum_x 2^{-l_x}}{2^{-l_x}}. \tag{1.20}$$

Equation 1.20 implies

$$\sum_x P(x) \log \frac{1}{P(x)} \leq \sum_x P(x) l_x + \left(\sum_x P(x) \right) \log \sum_x 2^{-l_x}. \tag{1.21}$$

Since $\sum_x P(x) = 1$, $L = \sum_x P(x) l_x$, and $H(P) = \sum_x P(x) \log 1/P(x)$, Equation 1.21 can be rewritten as

$$H(P) \leq L + \log \sum_x 2^{-l_x}. \tag{1.22}$$

Since D is a prefix-code, it follows from Kraft's inequality, Theorem 1.11.1, that $\sum_x 2^{-l_x} \leq 1$. Thus, $\log \sum_x 2^{-l_x} \leq 0$. Hence by Equation 1.22, $H(P) \leq L$. □

Example 1.11.12 We can now settle in the negative the question whether complete code-word alphabets are necessarily optimal for all prior distributions. Let E be a prefix-code with source alphabet $\{0, \ldots, k+1\}$ defined by $E(x) = 1^{x-1}0$ for $x = 1, 2, \ldots, k$ and $E(k+1) = 1^k$. Then E is obviously complete. It has an average code-word length $L_{E,P} = \sum_x P(x) x$ with respect to the probability distribution P. If $P(x) = 2^{-x}$ for $x = 1, 2, \ldots, k$ and $P(k+1) = 2^{-k}$, then $L_{E,P} = \sum_x P(x) \log 1/P(x)$, so that the expected code-word length is exactly equal to the entropy and hence to the minimal code-word length L by the noiseless coding theorem, Theorem 1.11.2. But for the uniform distribution $P(x) = 1/(k+1)$ for $x = 1, 2, \ldots, k+1$ we find that $L_{E,P} \gg L$, so that the code is not optimal with respect to this distribution. ◇

It is obviously important to find optimal prefix-codes. We have seen that completeness has not much to do with it. For optimality of finite codes we must choose code-word lengths corresponding to the probabilities of the encoded source words. This idea is implemented in the Shannon–Fano code, Example 1.11.2 on page 68 and Lemma 4.3.3 on page 274.

Example 1.11.13 Another issue is the effectiveness of the decoding process. The decoder needs to match the code-word patterns to the code-word sequence in order to retrieve the source word sequence. For a finite code-word alphabet, the code can be stored in a finite table. For infinite code-word alphabets we must recognize the code words.

Definition 1.11.4 A prefix-code is called *self-delimiting* if there is a Turing machine that decides whether a given word is a code word, never reading beyond the word itself, and moreover, computes the decoding function. (With respect to the Turing machine, each code word has an implicit end marker.)

As an example, let us define the minimal description length for elements in $\mathcal{N}$ with respect to the class of Turing machines $\mathcal{T}$. Fix a self-delimiting code $E : \mathcal{T} \to \mathcal{N}$. Denote the code word for Turing machine T by $E(T)$. Then the minimum description length of $x \in \mathcal{N}$ with respect to E is defined as $\min\{l(E(T)y) : T \text{ on input } y \text{ halts with output } x\}$. This minimum description length of x is actually an alternative definition of the Kolmogorov complexity $C(x)$. ◇

1.11.4 Universal Codes

For finite codes, the optimality is governed by how closely the set of code-word lengths corresponds to the probability distribution on the set of source words. In the proof of the noiseless coding theorem, Theorem 1.11.2, we chose a code that corresponds to the probability distribution of the source words. But the actual probability distribution may be unknown, nonrecursive, or it may be unclear how to determine the characteristics of the source. For example, consider the encoding of the printed English language, which emanates from many different sources with (it is to be assumed) different characteristics. Can we find a code that is optimal for *any* probability distribution, rather than for a *particular* one?

Example 1.11.14 How does one transmit any sufficiently long source-word sequence in an optimal-length code-word sequence without knowing the characteristics of the source, and in particular, without knowing in advance the relative frequencies of source words for the entire sequence? The solution is as follows:

Split the source-word sequence $x = x_1 x_2 \dots x_n$ of length n into $m \leq n$ blocks $b_1 b_2 \dots b_m$. Then we encode each b_i by, for instance, first giving

the numbers of occurrences of the source words in b_i and second the index of b_i in the lexicographically ordered ensemble of source words determined by these numbers. For instance, let b_i have length n_i, and let there be q source words whose numbers of occurrences of the different source words in b_i are, respectively, $k_1, k_2, \ldots, k_q$, $\sum_j k_j = n_i$. Encode b_i by the frequency vector $(k_1, k_2, \ldots, k_q)$, together with the index h of b_i in the ensemble of all possible sequences of length n_i in which the source words occur with these frequencies,

$$h \leq \binom{n_i}{k_1, k_2, \ldots, k_q}.$$

The encoding of b_i in standard decodable format with all items except the last one self-delimiting as, say, $\overline{k_1} \ldots \overline{k_q} h$, takes at most

$$O(q \log n_i) + \log \frac{n_i!}{k_1! k_2! \cdots k_q!}$$

bits. Defining the frequency of each source word a_j in block b_i as $p_j = k_j/n_i$, we find that the length of this encoding of b_i for large n_i and fixed p_j's approaches $n_i(\sum_j p_j \log 1/p_j)$. By the noiseless coding theorem, Theorem 1.11.2, the minimal average code-word length for a source-word sequence b_i produced by a stochastic source that emits source word a_j with probability $P^{(i)}(a_j) = p_j$ is given by $\sum_j p_j \log 1/p_j = H(P^{(i)})$.

That is, we can separately encode each block b_i asymptotically optimally, without knowing anything about the overall relative frequencies. As a result, the overall message x is encoded asymptotically in length

$$n_1 H(P^{(1)}) + n_2 H(P^{(2)}) + \cdots + n_m H(P^{(m)}).$$

It turns out that this implies that x is optimally encoded as well, since calculation shows that

$$nH(P) \geq n_1 H(P^{(1)}) + n_2 H(P^{(2)}) + \cdots + n_m H(P^{(m)}),$$

with $H(P)$ the entropy based on the overall source-word sequence x and $H(P^{(i)})$ the entropy of the individual blocks b_i. ◇

Suppose we use variable-length binary blocks b_i as in Example 1.11.14 as code words for a countably infinite set of source words such as $\mathcal{N}$. Then a universal code is a code that optimizes the average code-word length, independent of the distribution of the source words, in the following sense:

Definition 1.11.5 Let $C = \{w_1, w_2, \ldots\} \subseteq \{0,1\}^*$ be an infinite alphabet of uniquely decodable code words, and let $\mathcal{N}$ be a set of source words with probability

distribution P that assigns positive probability to each source word. Let code C assign code word w_x to source word $x \in \mathcal{N}$. Then C is *universal* if there is a constant c, independent of P, such that

$$\frac{\sum_x P(x)l(w_x)}{\max\{H(P),1\}} \leq c,$$

for all P with $0 < H(P) < \infty$. A universal code C is *asymptotically optimal* if there is an f such that

$$\frac{\sum_x P(x)l(w_x)}{\max\{H(P),1\}} \leq f(H(P)) \leq c,$$

with $\lim_{H(P)\to\infty} f(H(P)) = 1$.

Example 1.11.15 For a nonempty finite binary string $x = x_1x_2\ldots x_n$ we defined $\bar{x} = 1^n0x_1x_2\ldots x_n$. For example, 01011 is coded as 11111001011. Let C be defined as the set of binary strings resulting from this construction: $C = \{\bar{x} : x \in \{0,1\}^*\}$. The proof that this is a universal set of code words, but not an asymptotically optimal one, is omitted.

However, a relatively minor improvement yields an asymptotically optimal code-word alphabet. This time we encode x not by $\bar{x}$, but by $E(x) = \overline{l(x)}x$, that is, by encoding first the length of x in prefix-free form, followed by the literal representation of x. For example, 01011 is now coded as 1101001011, encoding $l(x)$ as 10 according to Equation 1.3. Code E is prefix-free, since if we know the length of x as well as the start of its literal representation, then we also know where it ends. The length set of this code is given by $l(E(x)) = l(x) + 2l(l(x)) + 1$ for $x \in \mathcal{N}$. The proof that this code is asymptotically optimal universal is omitted. ◇

This shows that there are asymptotically optimal prefix-codes. We now inquire how far we can push the idea involved.

Example 1.11.16 It is straightforward to improve on Example 1.11.15 by iterating the same idea, as in Equation 1.4 on page 13. That is, we precede x by its length $l(x)$, and in turn precede $l(x)$ by its own length $l(l(x))$, and so on. That is, the prefix-code E_i, for all finite strings x, is defined by $E_i(x) = E_{i-1}(l(x))x$ and $E_1(x) = \bar{x}$. For example, with $i = 3$, the string 01011 is coded as 1011001011, using the correspondence of Formula 1.3. This is a kind of ladder code, where the value of i is supposed to be fixed and known to coder and decoder. The length of $E_i(x)$ is given by

$$l(E_i(x)) = \begin{cases} 2l(x)+1 & \text{if } i = 1, \\ l(x) + l(E_{i-1}(l(x))) & \text{if } i > 1. \end{cases}$$

For each fixed $i > 1$ the prefix-code E_i is self-delimiting, universal, and asymptotically optimal. ◇

Can we improve this? Define $l^k(x) = l(l^{k-1}(x))$, the k-fold iteration of taking the length, and $l^1(x) = l(x)$. Define a coding

$$E^*(x) = l^k(x)0l^{k-1}(x)0\ldots 0x1,$$

with k the number of steps necessary to get $l^k(x) = 1$. For instance, the string 01011 is coded as 10100010111, using the correspondence of Equation 1.3. Again, this code is self-delimiting, universal, and asymptotically optimal. The code-word length is given by

$$l(E^*(x)) = 1 + \sum_{i=1}^{k}(l^i(x) + 1).$$

This is within an $O(k)$ additive term of $l^*(x)$, defined as

$$l^*(x) = \log x + \log\log x + \log\log\log x + \cdots, \tag{1.23}$$

where the sum involves only the positive terms. The number of terms is denoted by $\log^* x$. It can be proved that $\sum_x 2^{-l^*(x)} = c$ is finite, with $c = 2.865064\ldots$. If we put $l_x = l^*(x) + \log c$, then the Kraft inequality is satisfied with equality.

However, a lower bound on the code length is set by $\ell^*(x)$, defined as

$$\ell^*(x) = \begin{cases} l(x) + \ell^*(l(x)) & \text{if } l(x) > 1, \\ l(x) & \text{if } l(x) = 0, 1. \end{cases} \tag{1.24}$$

It can be shown that

$$l^*(x) - \ell^*(x) \leq \log^* x, \tag{1.25}$$
$$\sum_x 2^{-\ell^*(x)} = \infty;\ \sum_x 2^{-l^*(x)} < 3.$$

Hence, although ℓ^* is fairly close to l^*, by the divergence of the ℓ^* series in Equation 1.25 it follows by the Kraft inequality, Theorem 1.11.1, that there is no prefix-code with length set $\{\ell^*(x) : x \in \mathcal{N}\}$.

1.11.5 Statistics

Statistics deals with gathering data, ordering and representing data, and using the data to determine the process that causes the data. That this viewpoint is a little too simplistic is immediately clear: suppose that the true cause of a sequence of outcomes of coin flips is a fair coin, where both sides come up with equal probability. It is possible that the sequence consists of 'heads' only. Suppose that our statistical inference method succeeds in identifying the true cause (fair coin flips) from these data. Such a method is clearly at fault: from an all-heads sequence a good inference should conclude that the cause is a coin with a heavy bias toward 'heads,' irrespective of what the true cause is. That is, a good inference method must assume that the data are typical for the cause—we don't aim at finding the true cause, but we aim at finding a

cause for which the data are as typical as possible. Such a cause is called a *model* for the data. For some data it may not even make sense to ask for a true cause. This suggests that truth is not our goal; but within given constraints on the model class we try to find the model for which the data are most typical in an appropriate sense, the model that best fits the data. Considering the available model class as a magnifying glass, finding the best-fitting model for the data corresponds to finding the position of the magnifying glass that best brings the object into focus.

In introducing the notion of sufficiency in classical statistics, R.A. Fisher (1890–1962) observed:

> "The statistic chosen should summarize the whole of the relevant information supplied by the sample. This may be called the Criterion of Sufficiency ... In the case of the normal curve of distribution it is evident that the second moment is a sufficient statistic for estimating the standard deviation." [Fisher]

A 'sufficient' statistic of the data contains all information in the data about the model class. Sufficiency is related to the concept of data reduction. Suppose that we have data consisting of n bits. If we can find a sufficient statistic that takes values of $O(\log n)$ bits, then we can reduce the original data to the sufficient statistic with no loss of information about the model class.

This notion has a natural interpretation in terms of mutual information, Equation 1.15 on page 71, so that we may just as well think of a probabilistic sufficient statistic as a concept in information theory. Let $\{P_\theta : \theta \in \Theta\}$ be a family of distributions, with *parameters* $\theta \in \Theta$, of a random variable X that takes values in a finite or countable *set of data* $\mathcal{X}$. Such a family is also called a *model class*. A *statistic* S is a function $S : \mathcal{X} \to \mathcal{S}$ taking values in some set $\mathcal{S}$. We also call $S(x)$ a statistic of data $x \in \mathcal{X}$. A statistic S is said to be 'sufficient' for the model class Θ if all information about any θ present in the observation x is also present in the coarser-grained observation s. Formally, let $p_\theta(x) = P_\theta(X = x)$, and let $p_\theta(x|S(x) = s)$ denote the probability mass function of the conditional distribution. Define

$$p_\theta(s) = \sum_{x:S(x)=s} p_\theta(x),$$

$$p_\theta(x|S(x) = s) = \begin{cases} p_\theta(x)/p_\theta(s) & \text{if } S(x) = s, \\ 0 & \text{if } S(x) \neq s. \end{cases}$$

Definition 1.11.6 A statistic S is *sufficient* if there exists a function $q : \mathcal{X} \times \mathcal{S} \to \mathcal{R}$ such that

$$q(x, s) = p_\theta(x|S(x) = s), \tag{1.26}$$

for every $\theta \in \Theta$, $s \in \mathcal{S}$, and $x \in \mathcal{X}$.

Example 1.11.17 Let $\mathcal{X} = \{0,1\}^n$, let $X = (X_1, \ldots, X_n)$ be a sequence of n independently and identically distributed random variables X_i, each of which is a coin flip with probability θ of outcome 1 (success) and probability $1-\theta$ of outcome 0 (failure). The corresponding model class is $\{P_\theta : \theta \in (0,1) \subseteq \mathcal{R}\}$. Then, with outcome $X = x$ ($x = x_1 \ldots x_n$),

$$p_\theta(x) = p_\theta(x_1 \ldots x_n) = \theta^{S(x)}(1-\theta)^{n-S(x)},$$

where $S(x) = \sum_{i=1}^n x_i$ is the number of 1's in x. This function $S : \mathcal{X} \to \{0, \ldots, n\}$ is a sufficient statistic for the model class above. Namely, choose an element P_θ from the model class, with parameter $\theta \in (0,1)$, and an $s \in \{0, \ldots, n\}$. Then all x's with s ones and $n-s$ zeros are equally probable. The number of such x's is $\binom{n}{s}$. Therefore,

$$p_\theta(x|S(x) = s) = \begin{cases} 1/\binom{n}{s} & \text{if } S(x) = s, \\ 0 & \text{otherwise,} \end{cases} \tag{1.27}$$

for every $\theta \in (0,1)$. That is, the distribution $p_\theta(x|S(x) = s)$ is independent of the parameter θ. Equation 1.27 satisfies Equation 1.26, with $q(x,s)$ defined as the uniform probability of an x with exactly s ones. Therefore, $S(x)$ is a sufficient statistic relative to the model class in question. ◇

Example 1.11.18 **(Relation to mutual information)** The definition of sufficient statistic, Equation 1.26, can also be formulated in terms of mutual information. Choose some prior distribution over Θ, the parameter set for our model class. We denote the probability mass function of this distribution by p_1. In this way, we can define joint distributions

$$p(\theta, x) = p_1(\theta)p_\theta(x),$$
$$p(\theta, S(x)) = p_1(\theta)p_\theta(S(x)),$$

and the mutual information items

$$I(\Theta; X) = \sum_{\theta,x} p(\theta,x) \log \frac{p(\theta,x)}{\sum_\theta p(\theta,x) \sum_x p(\theta,x)},$$

$$I(\Theta; S(X)) = \sum_{\theta,S(x)} p(\theta,S(x)) \log \frac{p(\theta,S(x))}{\sum_\theta p(\theta,S(x)) \sum_{S(x)} p(\theta,S(x))}.$$

This leads to an alternative formulation of the notion of sufficient statistic of Definition 1.11.6 in terms of mutual information.

Lemma 1.11.1 *A statistic S is sufficient iff $I(\Theta; X) = I(\Theta; S(X))$ under all prior distributions $p_1(\theta)$.*

That is, a statistic is sufficient iff the mutual information between model random variable and data random variable is invariant under coarse-graining the data by taking the statistic. We defer the proof to Exercise 1.11.15 on page 90. ◇

Example 1.11.19 **(Minimal sufficient statistic)** Continue Example 1.11.17 on page 84. Consider a statistic T that counts the number of 1's in x that are followed by a 1. This statistic is not sufficient. But the combined statistic $V(x) = (S(x), T(x))$, with $S(x)$ counting the number of 1's in x as in Example 1.11.17, is sufficient, since it contains all information in x about the model class concerned. Such a statistic is overly sufficient, as is the data x itself, since it gives too much detail of x with respect to the model class in question. Generally, we want to obtain a statistic that gives just sufficient information, and anything less is insufficient. A statistic is a *minimal* sufficient statistic with respect to an indexed model class $\{p_\theta\}$ if it is a function of all other sufficient statistics: it contains no irrelevant information and maximally compresses the information in the data about the model class. For the family of normal distributions, the sample mean is a minimal sufficient statistic, but the sufficient statistic consisting of the mean of the even samples in combination with the mean of the odd samples is not minimal. Note that one cannot improve on sufficiency: The data-processing inequality, Equation 1.19 on page 72, states that

$$I(\Theta; X) \geq I(\Theta; S(X)),$$

for every function S, and that for randomized functions S an appropriate related expression holds. That is, mutual information between data random variable and model random variable cannot be increased by processing the data sample in any way. ◇

1.11.6 Rate Distortion

Let x belong to a set $\mathcal{X}$ of source words. Suppose we want to communicate source words using a code of at most r bits for each such word. (We call r the *rate*.) If 2^r is smaller than $d(\mathcal{X})$, then this is clearly impossible. However, for every x we can try to use a representation y that in some sense is close to x. Assume that the representations are chosen from a set $\mathcal{Y}$, possibly different from $\mathcal{X}$. Its elements are the *destination words*. We are given a function from $\mathcal{X} \times \mathcal{Y}$ to the reals, called the *distortion measure*. It measures the lack of fidelity, which we call *distortion*, of the destination word y against the source word x.

Example 1.11.20 For a given binary string x of length n and precision $\delta \in [0, \frac{1}{2}]$ we may look for a simple string y of length n such that the Hamming distance between x and y does not exceed δn. Such questions are related to lossy

compression, where we have a trade-off between the compressed length and the distortion (a certain distance between the original object and the lossily compressed object). ◇

The classical rate-distortion theory was initiated by Shannon, and we briefly recall his approach. A single-letter distortion measure is a function d that maps elements of $\mathcal{X} \times \mathcal{Y}$ to the reals. Define the distortion between words x and y of the same length n over alphabets $\mathcal{X}$ and $\mathcal{Y}$, respectively, as

$$d^n(x,y) = \frac{1}{n}\sum_{i=1}^{n} d(x_i, y_i).$$

Let X be a random variable with values in $\mathcal{X}$. Consider the random variable X^n with values in $\mathcal{X}^n$ that is the sequence $X_1, \ldots, X_n$ of n independent copies of X. We want to encode words of length n over $\mathcal{X}$ by words over $\mathcal{Y}$ so that the number of all code words is small and the expected distortion between outcomes of X^n and their codes is small. The trade-off between the expected distortion and the number of code words used is expressed by the *rate-distortion* function denoted by $r^n(\delta)$. It maps every $\delta \in \mathcal{R}$ to the minimal natural number r (we call r the *rate*) having the following property: There is an encoding function $E : \mathcal{X}^n \to \mathcal{Y}^n$ with a range of cardinality at most 2^r such that the expected distortion between the outcomes of X^n and their corresponding codes is at most δ.

In 1959 Shannon gave the following nonconstructive asymptotic characterization of $r^n(\delta)$. Let Z be a random variable with values in $\mathcal{Y}$. Let $H(Z)$, $H(Z|X)$ stand for the Shannon entropy and conditional Shannon entropy, respectively. Let $I(X;Z) = H(Z) - H(Z|X)$ denote the mutual information in X and Z, and let $\mathbf{E}d(X,Z)$ stand for the expected value of $d(X,Z)$. For a real δ, let $R(\delta)$ denote the minimal $I(X;Z)$ subject to $\mathbf{E}d(X,Z) \leq \delta$. That such a minimum is attained for all δ can be shown by compactness arguments.

Theorem 1.11.3 *For every n and δ we have $r^n(\delta) \geq nR(\delta)$. Conversely, for every δ and every positive ϵ, we have $r^n(\delta+\epsilon) \leq n(R(\delta)+\epsilon)$ for all large enough n.*

Exercises

1.11.1. [10] To use an extra symbol like 2 is costly when expressed in bits. Show that the coding of strings consisting of k zeros and ones and one 2 requires messages of about $k + \log k$ bits.

Comments. Hint: there are $2^k(k+1)$ such strings.

1.11.2. [13] Suppose we have a random variable X that can assume values a, b, c, d with probabilities $\frac{1}{2}$, $\frac{1}{4}$, $\frac{1}{8}$, and $\frac{1}{8}$, respectively, with no dependency between the consecutive occurrences.

(a) Show that the entropy $H(X)$ is $\frac{7}{4}$ (bits per symbol).

(b) Show that the code E with $E(a) = 0$, $E(b) = 10$, $E(c) = 110$, and $E(d) = 111$ achieves this limiting value.

1.11.3. [10] Let M be the set of possible source messages, and let $E : M \to \{0, 1\}^*$ be a prefix-code.

(a) Let M be a set of messages using symbols in an alphabet A. Show that if E is a prefix-code on A, then the homomorphism induced by E is a prefix-code on M.

(b) Show that the Shannon–Fano code presented in the main text is a prefix-code.

1.11.4. [26] Prove that the entropy function H has the following four properties:

(a) For given n and $\sum_{i=1}^{n} p_i = 1$, the function $H(p_1, p_2, \ldots, p_n)$ takes its largest value for $p_i = 1/n$ $(i = 1, 2, \ldots, n)$. That is, the scheme with the most uncertainty is the one with equally likely outcomes.

(b) $H(X, Y) = H(X) + H(Y|X)$. That is, the uncertainty in the product of schemes x and y equals the uncertainty in scheme x plus the uncertainty of y given that x occurs.

(c) $H(p_1, p_2, \ldots, p_n) = H(p_1, p_2, \ldots, p_n, 0)$. That is, adding the impossible event or any number of impossible events to the scheme does not change its entropy.

(d) $H(p_1, p_2, \ldots, p_n) = 0$ iff one of the numbers $p_1, p_2, \ldots, p_n$ is one and all the others are zero. That is, if the result of the experiment can be predicted beforehand with complete certainty, then there is no uncertainty as to its outcome. In all other cases the entropy is positive.

Comments. Source: C.E. Shannon, *Bell System Tech. J.*, 27(1948), 379–423, 623–656.

1.11.5. [32] Prove the following theorem. Let $H(p_1, p_2, \ldots, p_n)$ be a function defined for any integer n and for all values $p_1, p_2, \ldots, p_n$ such that $p_i \geq 0$ $(i = 1, 2, \ldots, n)$, $\sum_{i=1}^{n} p_i = 1$. If for any n this function is continuous with respect to all its arguments, and if it has the properties in Items (a), (b), and (c) of the previous exercise, then $H(p_1, p_2, \ldots, p_n) = \lambda \sum_{i=1}^{n} p_i \log 1/p_i$, where λ is a positive constant.

Comments. This is called the *entropy uniqueness theorem*. It shows that our choice of expression for the entropy for a finite probability scheme

is the only one possible if we want to have certain general properties that seem necessary in view of the intended meaning of the notion of entropy as a measure of uncertainty or as amount of information. Source: A.I. Khintchin, *Mathematical Foundations of Information Theory*, Dover, 1957.

1.11.6. [08] Define a code c by $E(1) = 00$, $E(2) = 01$, $E(3) = 10$, $E(4) = 11$, $E(5) = 100$.

(a) Show that the code sequence 00011011100 is uniquely decodable.

(b) Show that the code sequence 100100 can be decoded into two different source sequences.

1.11.7. [09] (a) Define code E by $E(x) = 01^x$ for $x = 1, 2, \ldots$. Show that E is uniquely decodable, but it is not a prefix-code.

(b) Define a code E by $E(x) = 1^x0$ for $x = 1, 2, \ldots$. Show that this is a prefix-code and (hence) uniquely decodable.

1.11.8. [19] Let $K = \{x : \phi_x(x) < \infty\}$ be the diagonal halting set (Section 1.7). Let $k_1, k_2, \ldots$ be the list of elements of K in increasing order. Define code E by $E(x) = 1^{k_x}0$.

(a) Show that E is a prefix-code and uniquely decodable.

(b) Show that E is not recursive.

(c) Show that the decoding function E^{-1} is not recursive.

Comments. Hint: the set K is recursively enumerable but not recursive. Codes with nonrecursive code-word alphabets are abundant, since there are uncountably many codes (prefix-codes), while there can be only countably many recursive code-word alphabets.

1.11.9. [22] Let a code have the set of code-word lengths $l_1, l_2, \ldots$. Show that if the code is uniquely decodable, then the Kraft inequality, Theorem 1.11.1, must be satisfied.

Comments. Thus, Theorem 1.11.1 holds for the wider class of uniquely decodable codes. This is called the *McMillan–Kraft Theorem.* Source: R.G. Gallager, *Information Theory and Reliable Communication*, Wiley, 1968. Attributed to B. McMillan.

1.11.10. [26] (a) Show that the set of code words $\{\bar{x} : x \in \mathcal{N}\}$ is a universal code-word alphabet. Show that it is *not* asymptotically optimal.

(b) Show that the set of code words $\{\overline{l(x)}x : x \in \mathcal{N}\}$ is a universal code-word alphabet. Show that it is also asymptotically optimal.

Comments. Source: P. Elias, *IEEE Trans. Inform. Theory*, IT-21(1975), 194–203.

1.11.11. [37] The prefix-code E^* of Example 1.11.16 is an asymptotically optimal universal code because E_2 is already one.

(a) Show that $\sum_n 2^{-l^*(n)} = c$ with $c = 2.865064\ldots$. Show that the Kraft inequality, Theorem 1.11.1, is satisfied with equality for the length set $l_n = l^*(n) + \log c$, $n = 1, 2, \ldots$.

(b) Let the code E^+ be defined by

$$E^+(x) = \bar{k}l^k(x)l^{k-1}(x)\ldots x,$$

where the length function is iterated until the value $l^k(x) = 1$. Show that E^+ is prefix-free. Show that this representation of the integers is even more compact than E^*.

Comments. Source for Item (a): J. Rissanen, *Ann. Statist.*, 11(1983), 416–431.

1.11.12. [29] The function $\log^* n$ denotes the number of times we can iterate taking the binary logarithm with a positive result, starting from n. This function grows extremely slowly. It is related to the Ackermann function of Exercise 1.7.18. In that notation it is a sort of inverse of $f(3, x, 2)$.

(a) Let $l_1, l_2, \ldots$ be any infinite integer sequence that satisfies the Kraft inequality, Theorem 1.11.1. Show that $l_n > l^*(n) - 2\log^* n$ for infinitely many n.

(b) Show that $\log^* n$ is unbounded and primitive recursive. In particular, show that although $\log^* n$ grows very slowly, it does not grow more slowly than any unbounded primitive recursive function.

Comments. Hint: use exercises in Section 1.7. Because $\log^* n$ grows very slowly, we conclude that $l^*(n)$ is not far from the least asymptotic upper bound on the code-word-length set for all probability sequences on the positive integers. In this sense it plays a similar role for binary prefix-codes as our one-to-one pairing of natural numbers and binary strings in Equation 1.3 plays with respect to arbitrary binary codes. Source: J. Rissanen, *Ibid.*

1.11.13. [M30] We consider convergence of the series $\sum_n 2^{-f(k,\alpha;n)}$ for $f(k, \alpha; n) = \log n + \log\log n + \cdots + \alpha \log^{(k)} n$, with $\log^{(k)}$ the k-fold iteration of the logarithmic function defined by $\log^{(1)} n = \log n$ and $\log^{(k)} n = \log\log^{(k-1)} n$ for $k > 1$.

(a) Show that for each $k \geq 2$, the series above diverges if $\alpha \leq 1$ and that the series converges if $\alpha > 1$.

(b) Show that the series $\sum_n n^{-\alpha}$ diverges for $\alpha \leq 1$ and converges for $\alpha > 1$. (This is the case $k = 1$.)

Comments. This gives an exact borderline between convergence and divergence for the series in Kraft's inequality, Theorem 1.11.1. Hint: use Cauchy's condensation test for convergence of series. Source: K. Knopp, *Infinite Sequences and Series*, Dover, 1956. Attributed to N.H. Abel.

1.11.14. [22] We derive a lower bound on the minimal average codeword length of prefix-codes. Consider the standard correspondence between binary strings and integers as in Equation 1.3. Define $f(n) = l(n) + l(l(n)) + \cdots + 2$. Show that $\sum_n 2^{-f(n)} = \infty$.

Comments. Hint: Let the number of terms in $f(n)$, apart from the 2, be $h(n) = 0$ for $n = 2$ and $1 + h(h(n))$ for $n > 2$ (this defines function h). Define $s_i = \sum_{h(n)=i} 2^{-f(n)}$ and $\sum_{n=3}^{\infty} 2^{-f(n)} = \sum_{i=0}^{\infty} s_i$. We show that all s_i are equal to 1, using induction on i. Clearly $s_0 = 2^{-0} = 1$. Also

$$\begin{aligned} s_{i+1} &= \sum_{h(n)=i+1} 2^{-f(n)} = \sum_{h(l(n))=i} 2^{-f(n)} \\ &= \sum_{h(m)=i} \sum_{l(n)=m} 2^{-(m+f(m))} = \sum_{h(m)=i} 2^m 2^{-(m+f(m))} = s_i. \end{aligned}$$

1.11.15. [32] Prove Lemma 1.11.1.

Comments. The notion of sufficient statistic is due to R.A. Fisher, *Philos. Trans. Royal Soc., London, Sec. A*, 222(1922), 309–368. The mutual information version is given in [T.M. Cover, J.A. Thomas, *Elements of Information Theory*, Wiley, New York, 1991, pp. 36–38]. Hint: the relationship in Lemma 1.11.1 between mutual information and sufficient statistic is due to S. Kullback, *Information Theory and Statistics*, Wiley, New York, 1959.

1.12 State × Symbol Complexity

A fourth historical root of Kolmogorov complexity seems to be another issue (other than information theory) raised by Shannon. In his paper "A universal Turing machine with two internal states" [pp. 129–153 in: *Automata Studies*, C.E. Shannon and J. McCarthy, eds., Princeton Univ. Press, 1956] he showed that there is a simple way of changing each Turing machine using $m > 2$ states to a Turing machine using only two states and computing essentially the same function. This requires expanding the number of tape symbols, since the state of the original machine needs to be stored and updated on the tape of the corresponding two-state machine. The main result of this exercise is the construction of a two-state universal Turing machine. It turns out that a one-state universal Turing machine does not exist. Trying to minimize the other main resource, the number of tape symbols, is resolved even more simply: two tape symbols suffice, namely, by the expedient of encoding the m different tape symbols in binary strings of $O(\log m)$ length. The string

of all 0's is reserved to encode the distinguished blank symbol B. It is not very difficult, but tedious, now to adapt the finite control to make the whole arrangement work. Again this is the minimum, since it is impossible to construct a universal Turing machine with only one tape symbol.

It turns out that in both cases (reducing the number of states to two, or the number of tape symbols to two), the *product* of the number of states and the number of tape symbols increases at most eightfold. This suggests that this product is a relatively stable measure of the complexity of description of algorithms in the syntax of our Turing machine formalism. Following this idea, Shannon proposed the state–symbol product as a measure of complexity of description of algorithms. In particular, we can classify computable functions by the smallest state–symbol product of Turing machines that compute it. Here we assume of course a fixed formalism to express the Turing machines. It is a straightforward insight that this product is closely related to the number of bits to syntactically specify the Turing machine in the usual notation.

Gregory Chaitin, in papers appearing in 1966 and 1969, analyzed this question in great detail. In his 1969 paper, observing that each Turing machine can be coded as a program for a fixed-reference universal machine, he formulated as a variant of Shannon's approach the issue of the lengths of the shortest programs of the reference Turing machine to calculate particular finite binary strings.

Exercises

1.12.1. [12] It is usual to allow Turing machines with arbitrarily large tape alphabets A (with the distinguished blank symbol B serving the analogous role as before). Use the quadruple formalism for Turing machines as defined earlier. How many Turing machines with m states and n tape symbols are there? (Count the blank tape symbol B as one of the n tape symbols.)

1.12.2. [20] Define Turing machines in quadruple format with arbitrarily large tape alphabets A, and state sets Q, $d(A), d(Q) < \infty$. Show that each such Turing machine with state set Q and tape alphabet A can be simulated by a Turing machine with tape alphabet A', $d(A') = 2$, and state set Q' such that $d(A')d(Q') \leq cd(A)d(Q)$, for some small constant c. Determine c. Show that the analogous simulation with $d(A') = 1$ is impossible ($d(Q') = \infty$).

Comments. See C.E. Shannon, pp. 129–153 in: *Automata Studies*, C.E. Shannon and J. McCarthy, eds., Princeton University Press, 1956. This is also the source for the next exercise.

1.12.3. [25] Show that each such Turing machine with state set Q and tape alphabet A can be simulated by a Turing machine with state set Q', $d(Q') = 2$, and tape alphabet A' such that $d(A')d(Q') \leq cd(A)d(Q)$, for

some small positive constant c. Determine c. Show that the analogous simulation with $d(Q') = 1$ is impossible (implies $d(A') = \infty$).

1.12.4. [37] Give a universal Turing machine with $d(A)d(Q) \leq 35$.

Comments. Such a construction was first found by M. Minsky [*Ann. Math.*, 74(1961), 437–455].

1.13 History and References

In notation concerning binary strings and so forth we follow A.K. Zvonkin and L.A. Levin [*Russ. Math. Surveys*, 25:6(1970), 83–124]. (According to the Cyrillic alphabet of the original version of this important survey of Kolmogorov complexity 'Zvonkin' precedes 'Levin.' This author order was maintained in the English translation.) The big-O notation is discussed in [D.E. Knuth, *SIGACT News*, 4:2(1976), 18–24; P.M.B. Vitányi and L.G.L.T. Meertens, *SIGACT News*, 16:4(1985), 56–59; D.E. Knuth, *Fundamental Algorithms*, Addison-Wesley, 1973].

The basics of combinatorics can be found in many textbooks, for instance [D.E. Knuth, *Fundamental Algorithms*, Addison-Wesley, 1973; W. Feller, *An Introduction to Probability Theory and Its Applications*, Wiley, 1968]. Turing machines were introduced by A.M. Turing in an important paper [*Proc. London Math. Soc.*, 42(1936), 230–265; Correction, *Ibid.*], where also the quoted material can be found. Related notions were introduced also in [E.L. Post, *J. Symb. Logic*, 1936]. The standard textbook references for basic computability theory are [H. Rogers, Jr., *Theory of Recursive Functions and Effective Computability*, McGraw-Hill, 1967; P. Odifreddi, *Classical Recursion Theory*, North-Holland, 1989]. The notion of semicomputable functions was, perhaps, first used in [A.K. Zvonkin and L.A. Levin, *Ibid.*]. In the previous editions of the current book they are called '(co-)enumerable functions.' Material on computable (recursive) and semicomputable real numbers partly arises from this context. For related work see [M.B. Pour-El and J.I. Richards, *Computability in Analysis and Physics*, Springer-Verlag, 1989].

In Section 1.7.4, we introduced the basic terminology of computational complexity theory that we will need in Chapter 7. For computational complexity theory see [J. Hartmanis, *Feasible computations and provable complexity properties*, SIAM, 1978; J.L. Balcázar, J. Diaz, and J. Gabarró, *Structural Complexity*, Springer-Verlag, 1988; and M.R. Garey and D.S. Johnson, *Computers and Intractability*, Freeman, 1979].

A.N. Kolmogorov's classic treatment of the set-theoretic axioms of the calculus of probabilities is his slim book *Grundbegriffe der Wahrscheinlichkeitsrechnung*, Springer-Verlag, 1933; English translation published by Chelsea, New York, 1956. A standard textbook in probability theory

is [W. Feller, *An Introduction to Probability Theory and Its Applications*, Wiley, 1968].

General statistical tests for (pseudo)randomness of sequences are extensively treated in [D.E. Knuth, *Seminumerical Algorithms*, Addison-Wesley, 1981]. Statistical testing for randomness of the decimal representations of π, e, and $\sqrt{2}$ specifically is found in [D.E. Knuth, *Ibid.*, 144–145; S.S. Skiena, *Complex Systems*, 1(1987), 361–366]. The quoted remark of J. von Neumann (1903–1957) appears in [Various techniques used in connection with random digits, *Collected Works*, Vol.V, Macmillan, 1963]. R. von Mises's foundation of probability theory based on frequencies is set forth in [*Mathemat. Zeitsch.*, 5(1919), 52–99; Correction, *Ibid.*, 6(1920); *Probability, Statistics and Truth*, Macmillan, 1939]. In the 1920 correction to this paper, von Mises writes, "In der Arbeit von Prof. Helm aus Dresden in Bd 1 der Naturphilosophie 1902, 364– 'Wahrscheinlichkeitstheorie des Kollektivbegriffes' dort wird auch die Umkehrung der Wahrscheinlichkeitsdefinition, nämlich ihre Zurückführung auf Kollektive statt auf Bereiches gleich möglicher Fälle, in der Hauptsache bereits vorgebildet." ("In the paper of Prof. Helm from Dresden in volume 1 of Naturphilosophie, 1902, 364– 'Wahrscheinlichkeitstheorie des Kollektivbegriffes' the key elements of the inverse definition of probability, namely its foundation on collectives instead of its foundation on domains of equally probable events, are also already exhibited.") A critical comparison of different (foundational) theories of probabilities is [T.L. Fine, *Theories of Probability*, Academic Press, 1973]. The formulation of the apparent circularity in the frequency foundation of probability is from [J.E. Littlewood, *Littlewood's Miscellany*, Cambridge Univ. Press, 1986, 71–73]. The quotations of Kolmogorov, as well as the finitary version of von Mises's collectives, are from [*Sankhyā*, Series A, 25(1963), 369–376]. The work in improving von Mises's notion of admissible place selections is due to A. Wald [*Ergebnisse eines Mathematischen Kolloquiums*, Vol. 8, 1937, 38–72], who proved existence of collectives under the restriction of the number of admissible place-selection rules to countably infinite; to A. Church [*Bull. Amer. Math. Soc.*, 46(1940), 130–135], who further restricted admissible place selections to computable functions; and to J. Ville [*Etude Critique de la Notion de Collectif*, Gauthier-Villars, 1939], who showed that the von Mises–Wald–Church definitions still fail to satisfy randomness properties such as the law of the iterated logarithm. Champernowne's number was found by D.G. Champernowne [*J. London Math. Soc.*, 8(1933), 254–260]. The entire issue of randomness of individual finite and infinite sequences is thoroughly reviewed by D.E. Knuth [*Ibid.*, 142–169; summary, history, and references: 164–166]. A recent survey on the foundational issues of randomness was given by M. van Lambalgen [*J. Symb. Logic*, 52(1987), 725–755; *Random Sequences*, Ph.D. thesis, Universiteit van Amsterdam, 1987].

Our treatment of Bayes's rule follows R. von Mises [*Probability, Statistics and Truth*, Macmillan, 1939]. The idea of a universal a priori probability is due to R.J. Solomonoff. The Chernoff bounds, Lemma 1.10.1, are due to H. Chernoff [*Ann. Math. Stat.*, 23(1952), 493–509]. The form we use is from L.G. Valiant and D. Angluin [*J. Comput. System Sci.*, 18:2(1979), 155–193], who in turn base it on [P. Erdős and J. Spencer, *Probabilistic Methods in Combinatorics*, Academic Press, 1974, p. 18].

Information theory was introduced as an essentially complete new mathematical discipline in C.E. Shannon's classic paper [*Bell System Tech. J.*, 27(1948), 379–423, 623–656]. Standard textbooks are [R.G. Gallager, *Information Theory and Reliable Communication*, Wiley & Sons, 1968; T.M. Cover and J.A. Thomas, *Elements of Information Theory*, Wiley & Sons, New York, 1991]. Our discussion of information theory used the original papers of Shannon. The Shannon–Fano code is due to C.E. Shannon [*Bell System Tech. J.*, 27(1948), 379–423, 623–656]. It is also attributed to R.M. Fano. The discussion is nonstandard insofar as we followed Kolmogorov's idea that the combinatorial approach to information theory should precede the probabilistic approach, in order to emphasize the logical independence of the development of information theory from probabilistic assumptions [*Problems Inform. Transmission*, 1(1965), 1–7; *Russian Math. Surveys*, 38:4(1983), 29–40]. The latter papers mention the following justification. In linguistic analysis it is natural to take such a purely combinatorial approach to the notion of the entropy of a language. This entropy is an estimate of the flexibility of a language, a number that measures the diversity of possibilities for developing grammatically correct sentences from a given dictionary and grammar. Using S.I. Ozhegov's Russian dictionary, M. Ratner and N.D. Svetlova obtained the estimate $h = (\log N)/n = 1.9 \pm 0.1$ with N the number of Russian texts of length n (number of letters including spaces). This turns out to be much larger than the upper estimate for entropy of literary texts that can be obtained by various methods of guessing continuations. Naturally so, since literary texts must meet many requirements other than grammatical correctness. For this discussion and further remarks about change of entropy under translations, and the entropy cost of adhering to a given meter and rhyme scheme, see A.N. Kolmogorov [*Sankhyā*, Series A, 25(1963), 369–376]. For instance, Kolmogorov reports that classic rhyming iambic tetrameter requires a freedom in handling verbal material characterized by a residual entropy of ca. 0.4. "The broader problem of measuring the information connected with creative human endeavor is of the utmost significance."

The general theory of coding and prefix-codes as in Section 1.11.1 is treated in [R.G. Gallager, *Information Theory and Reliable Communication*, Wiley, 1968]. The ubiquitous Kraft inequality for prefix-codes, Theorem 1.11.1, is due to L.G. Kraft [*A Device for Quantizing, Group-*

ing, and Coding Amplitude Modulated Pulses, M. Sc. thesis, Dept. Electr. Eng., MIT, Cambridge, Mass., 1949]. C.E. Shannon's noiseless coding theorem, Theorem 1.11.2, establishing the minimal average codeword length is from [*Bell System Tech. J.*, 27(1948), 379–423, 623–656]. Universal codes for infinite sets and unknown probability distributions were first proposed by Kolmogorov [*Problems Inform. Transmission*, 1:1(1965), 1–7]. For further developments such as universal coding we used [P. Elias, *IEEE Trans. Inform. Theory*, IT-21(1975), 194–203; S.K. Leung-Yan-Cheong and T.M. Cover *IEEE Trans. Inform. Theory*, IT-24(1978), 331–339; J. Rissanen, *Ann. Stat.*, 11(1983), 416–431; *Stochastic Complexity in Statistical Inquiry*, World Scientific, 1989].

The notion of sufficient statistic is due to R.A. Fisher, *Philos. Trans. Royal Soc., London, Sec. A*, 222(1922), 309–368. The mutual information version is given in [T.M. Cover, J.A. Thomas, *Elements of Information Theory*, Wiley, New York, 1991, pp. 36–38]. The relationship in Lemma 1.11.1 between mutual information and sufficient statistic is due to S. Kullback, *Information Theory and Statistics*, Wiley, New York, 1959.

Rate-distortion theory was introduced by C.E. Shannon [*Bell System Tech. J.*, 27(1948), 379–423, 623–656] and treated in detail in [*IRE National Convention Record, Part 4*, 1959, 142–163]. A textbook is T. Berger, *Rate Distortion Theory: A Mathematical Basis for Data Compression*, Prentice-Hall, 1971.

The state–symbol complexity measure for Turing machines (and recursive functions) was apparently first discussed by C.E. Shannon [*Automata Studies*, C.E. Shannon and J. McCarthy, eds., Princeton Univ. Press, 1956, 129–153].

Kolmogorov complexity originated with the discovery of universal descriptions, and a recursively invariant approach to the concepts of complexity of description, randomness, and a priori probability. Historically, it is firmly rooted in R. von Mises's notion of random infinite sequences as discussed above. The first work in this direction is possibly K. Gödel's *On the length of proofs* of 1936, in which he proves that adding axioms to undecidable systems shortens the proofs of many theorems (thus using length as a measure of the complexity of proofs). With the advent of electronic computers in the 1950s, a new emphasis on computer algorithms and a maturing general recursive function theory, ideas tantamount to Kolmogorov complexity, came to many people's minds, because "when the time is ripe for certain things, these things appear in different places in the manner of violets coming to light in early spring" (Wolfgang Bolyai to his son Johann in urging him to claim the invention of non-Euclidean geometry without delay [Herbert Meschkowski, *Noneuclidean Geometry*, Academic Press, New York, 1964, p. 33]). Thus, with Kolmogorov com-

plexity one can associate three inventors, in chronological order: R.J. Solomonoff, of Cambridge, Massachusetts, USA; A.N. Kolmogorov, of Moscow, Russia; and G.J. Chaitin, of New York, USA.

Already in November 1960, R.J. Solomonoff published a *Zator Company* technical report [A preliminary report on a general theory of inductive inference, Tech. Rept. ZTB-138, Zator Company, Cambridge, Mass., November 1960] that presented the basic ideas of the theory of algorithmic complexity as a means to overcome serious problems associated with the application of Bayes's rule in statistics. Ray Solomonoff was born on July 25, 1926, in Cleveland, Ohio, in the United States. He studied physics during about 1946–1950 at the University of Chicago (he recalls the lectures of E. Fermi and R. Carnap) and obtained an M.Sc. from that university. From about 1951–1956 he worked in the electronics industry doing math and physics and designing analog computers, working half-time. His scientific autobiography is published as [*J. Comput. System Sci.*, 55(1997), 73–88].

Solomonoff's objective was to formulate a completely general theory of inductive reasoning that would overcome shortcomings in Carnap's [*Logical Foundations of Probability*, Univ. Chicago Press, 1950]. Following some more technical reports, in a long journal paper [*Inform. Contr.*, 7(1964), 1–22, 224–254], he introduced 'Kolmogorov' complexity as an auxiliary concept to obtain a universal a priori probability and proved the invariance theorem, Theorem 2.1.1. The mathematical setting of these ideas is described in some detail in Section 1.10. Solomonoff's work has led to a novel approach in statistics [T.L. Fine, *Ibid.*], leading to applicable inference procedures such as the minimal description length principle; see Chapter 5.

This makes Solomonoff the first inventor and raises the question whether we ought to talk about 'Solomonoff complexity.' However, the name 'Kolmogorov complexity' for shortest effective description length has become well entrenched and commonly understood. Solomonoff was primarily interested in universal a priori probability, while Kolmogorov later, independently, discovered and investigated the associated complexity for its own sake. We will associate Solomonoff's name with the universal distribution and Kolmogorov's name with the descriptional complexity. It has become customary to designate the entire area dealing with descriptional complexity, algorithmic information, and algorithmic probability loosely by the name 'Kolmogorov complexity' or 'algorithmic information theory.' (Associating Kolmogorov's name with the area may be viewed as an example in the sociology of science of the Matthew effect, first noted in the Gospel according to Matthew, 25: 29–30, "For to every one who has more will be given, and he will have in abundance; but from him who has not, even what he has will be taken away.") A comprehensive ac-

count of Solomonoff's ideas and their genesis is presented in the history and references section of Chapter 4 and in Chapter 5.

Solomonoff's publications apparently received little attention until Kolmogorov started to refer to them from 1968 onward. These papers contain in veiled form suggestions about randomness of finite strings, incomputability of Kolmogorov complexity, computability of approximations to the Kolmogorov complexity, and resource-bounded Kolmogorov complexity. M. Minsky referred to Solomonoff's work [*Proc. I.R.E.*, January 1961, 8–30; p. 43 in *Proc. Symp. Appl. Math. XIV*, Amer. Math. Soc., 1962]. To our knowledge, these are the earliest documents outlining an algorithmic theory of descriptions.

The great Russian mathematician Andrei N. Kolmogorov was born 25 April 1903 in Tambov, Russia, and died 20 October 1987 in Moscow. Many biographical details can be found in the Soviet Union's foremost mathematics journal, *Uspekhi Mat. Nauk*, translated into English as *Russian Math. Surveys* [B.V. Gnedenko, 28:5(1973), 5–16, P.S. Aleksandrov, 38:4(1983), 5–7; N.N. Bogolyubov, B.V. Gnedenko, and S.L. Sobolev, 38:4(1983), 9–27; A.N. Kolmogorov, 41:6(1986), 225–246; and the entire memorial issue 43:6(1988), especially 1–39 by V.M. Tikhomirov]. Three volumes of Kolmogorov's (mathematical) *Selected Works* have been published by Nauka, Moscow (in Russian) in 1985 through 1987; they are translated in English and published by Kluwer (volume 1) and Springer (volumes 2 and 3). The writings on algorithmic complexity are collected in volume 3 of the *Selected Works*. In the Western literature, see the memorial issue [*Annals of Probability*, 17:3(1989)], especially the scientific biography on pp. 866–944 by A.N. Shiryaev, and an evaluation of Kolmogorov's contributions to information theory and to algorithmic complexity, on pp. 840–865 by T.M. Cover, P. Gács, and R.M. Gray. See also the obituary in [*Bull. London Math. Soc.*, 22:1(1990), 31–100] and [V.A. Uspensky, *J. Symb. Logic*, 57:2(1992), 385–412].

In 1933 A.N. Kolmogorov supplied probability theory with a powerful mathematical foundation in his [*Grundbegriffe der Wahrscheinlichkeitsrechnung*, Springer-Verlag, 1933]. Following a four-decade-long controversy on von Mises's conception of randomness, in which Kolmogorov played little part, the content of which is set forth in some detail in Section 1.9, Kolmogorov finally introduced complexity of description of finite individual objects as a measure of individual information content and randomness, and proved the invariance theorem, Theorem 2.1.1, in his paper of spring 1965 [*Problems Inform. Transmission*, 1(1965), 1–7]. His objective was primarily, apart from resolving the question of randomness of objects, to revise information theory by an algorithmic approach to the information content of individual objects, in contrast to the traditional way discussed in Section 1.11. According to A.N. Shiryaev, [*Annals of Probability*, 17:3(1989), 921], Kolmogorov described the essence and

background to the algorithmic approach in his report to the Probability Section of the Moscow Mathematical Society on April 24, 1963:

> "One often has to deal with very long sequences of symbols. Some of them, for example, the sequences of symbols in a 5-digit logarithm table, permit a simple logical definition and therefore might be obtained by the computations (though clumsy at times) of a simple pattern. [...] Others seem not to admit any sufficiently simple 'legitimate' way to construct them. It is supposed that such is the case for a rather long segment in a table of random numbers. [...] There arises the question of constructing a rigorous mathematical theory to account for these differences in behavior." [Kolmogorov]

Says Kolmogorov, "I came to similar conclusions [as Solomonoff], before becoming aware of Solomonoff's work, in 1963–1964" [*IEEE Trans. Inform. Theory*, IT 14:5(1968), 662–664]. And again, "The basic discovery, which I have accomplished independently from and simultaneously with R. Solomonoff, lies in the fact that the theory of algorithms enables us to eliminate this arbitrariness [of interpretive mechanisms for descriptions leading to different lengths of shortest descriptions for the same object, and hence to different 'complexities' with respect to different interpretive mechanisms] by the determination of a 'complexity' which is almost invariant (the replacement of one method by another leads only to the supplement of the bounded term)" [A.N. Shiryaev, *Ibid.*, 921]. In the case of the other two inventors, the subject we are concerned with appears, as it were, out of the blue. But Kolmogorov's involvement strikes one as the inevitable confluence of interests of this great scientist: his lifelong fascination with the foundations of probability theory and randomness, his immediate appreciation of information theory upon its formulation by Shannon, and his vested interest in the theory of algorithms witnessed by, for instance, A.N. Kolmogorov and V.A. Uspensky, *Uspekhi Mat. Nauk*, 13:4(1958), 3–28 [in Russian; translated *Amer. Math. Soc. Transl. (2)*, 29(1963), 217–245]. The new ideas were vigorously investigated by his associates. These included the Swedish mathematician P. Martin-Löf, visiting Kolmogorov in Moscow during 1964–1965, who investigated the complexity oscillations of infinite sequences and proposed a definition of infinite random sequences that is based on constructive measure theory [*Inform. Contr.*, 9(1966), 602–619; *Z. Wahrsch. Verw. Geb.,* 19(1971), 225–230]. There is a related development in set theory and recursion theory, namely, the notion of 'generic object' in the context of 'forcing.' For example, being a member of the arithmetic generic set is analogous (but not precisely) to being a member of the intersection of all arithmetic sets of measure 1. There is a notion, called '1-genericity,' that in a restricted version calls for the intersection of all recursively enumerable sets of measure 1. This is obviously related to the approach of Martin-Löf. Forcing was introduced by P. Cohen in 1963 to show the

independence of the continuum hypothesis, and using sets of positive measure as forcing conditions is due to R.M. Solovay soon afterward.

G.J. Chaitin had finished the Bronx High School of Science, and was an eighteen-year-old undergraduate student at the City College of the City University of New York, when he submitted two papers [*J. ACM*, 13(1966), 547–569; *J. ACM*, 16(1969), 145–159] for publication, in October and November 1965, respectively. In the 1966 paper he addresses the 'state × symbol' complexity of algorithms following Shannon's coding concepts, as described in Section 1.12, but does not introduce an invariant notion of complexity. See also G.J. Chaitin's abstracts [*AMS Notices*, 13(1966), 133, 228–229] submitted October 19, 1965, and January 6, 1966, respectively. Continuing this work in the 1969 paper, in the final part, Chaitin puts forward the notion of Kolmogorov complexity, proves the invariance theorem, Theorem 2.1.1, and studies infinite random sequences (in the sense of having maximally random finite initial segments) and their complexity oscillations. As Chaitin [*Scientific American*, 232:5(1975), 47–52] formulates it, "this definition [of Kolmogorov complexity] was independently proposed about 1965 by A.N. Kolmogorov and me. [...] Both Kolmogorov and I were then unaware of related proposals made in 1960 by Ray Solomonoff." A short autobiography appears in [G.J. Chaitin, *Information-Theoretic Incompleteness*, World Scientific, Singapore, 1992].

2

Algorithmic Complexity

The most natural approach to defining the quantity of information is clearly to define it in relation to the individual object (be it Homer's *Odyssey* or a particular type of dodo) rather than in relation to a set of objects from which the individual object may be selected. To do so, one could define the quantity of information in an object in terms of the number of bits required to losslesly describe it. A description of an object is evidently useful in this sense only if we can reconstruct the full object from this description.

We aim at something different from C.E. Shannon's theory of communication, which deals with the specific technological problem of data transmission, that is, with the information that needs to be transmitted in order to select an object from a previously agreed-upon set of alternatives; Section 1.11. Our task is to widen the limited set of alternatives until it is universal. We aim at a notion of absolute information of individual objects, that is, the information that by itself describes the object completely.

Intuition tells us that some objects are complicated and some objects are simple. For instance, a number like 2^{1000} is certainly very simple (we have just expressed it in a few bits); yet evidently there are numbers of a thousand bits for which it is hard to see how we can find a description requiring many fewer than a thousand bits. Such hard-to-describe numbers would be their own shortest descriptions.

We require both an agreed-upon universal description method and an agreed-upon mechanism to produce the object from its alleged description. This would appear to make the information content of an object depend on whether it is particularly favored by the description method

M. Li and P.M.B. Vitányi, *An Introduction to Kolmogorov Complexity and Its Applications*,
DOI: 10.1007/978-0-387-49820-1_2, © Springer Science + Business Media, LLC 2008

we have selected. By 'favor' we mean to produce short descriptions in terms of bits.

For instance, it is well known that certain programming languages favor symbolic computations, while other programming languages favor arithmetic computations, even though all of them are universal. The notion of information content of individual objects can be useful only if the quantity of information is an attribute of the object *alone* and is independent of the means of description. It is a priori by no means obvious that this is possible. Relatively recent advances resulting in the great ideas of computability theory from the 1930s onward have made it possible to design a universal description method that appears to meet our goals.

Denote the set of objects by S, and assume some standard enumeration of objects x by natural numbers $n(x)$. We are interested in the fact that $n(x)$ may not be the most economical way to specify x. To compare methods of specification, we view such a method as a partial function over the nonnegative integers defined by $n = f(p)$. We do not yet assume that f is recursive, but maintain full generality to show to what extent such a theory can also be developed with noneffective notions, and at which point effectiveness is required. With each natural number p associate the length of the finite binary string identified with p as in Equation 1.3. Denote this length by $l(p)$.

For each object x in S, the complexity of object x with respect to the specifying method f is defined as

$$C_f(x) = \min\{l(p) : f(p) = n(x)\},$$

and $C_f(x) = \infty$ if there are no such p. In computer science terminology we would say that p is a program and f a computer, so that $C_f(x)$ is the minimal length of a program for f (without additional input) to compute output x.

Considering distinct methods $f_1, f_2, \ldots, f_r$ of specifying the objects of S, it is easy to construct a new method f that assigns to each object x in S a complexity $C_f(x)$ that exceeds only by c (less than about $\log r$) the minimum of $C_{f_1}(x), C_{f_2}(x), \ldots, C_{f_r}(x)$. The only thing we have to do is to reserve the first $\log r$ bits of p to identify the method f_i that should be followed, using as a program the remaining bits of p.

We say that a method f *minorizes* a method g (additively) if there is a constant c such that for all x,

$$C_f(x) \leq C_g(x) + c.$$

Above we have shown how to construct a method f that minorizes each of the methods $f_1, \ldots, f_r$ with constant $c \approx \log r$. Two methods f and g are called *equivalent* if each of them minorizes the other.

Consider the hierarchy of equivalence classes of methods with respect to minorization. Kolmogorov has remarked that the idea of 'description length' would be useless if the constructed hierarchy did not have certain niceness properties. In particular, we would like such a hierarchy to have a unique minimal element: the equivalence class of description methods that minorize all other description methods. Some sets of description methods do have a unique minimal element, while other sets of description methods don't.

Definition 2.0.1 Let $\mathcal{C}$ be a subclass of the partial functions over the nonnegative integers. A function f is *additively optimal* (a special type of universality) for $\mathcal{C}$ if it belongs to $\mathcal{C}$ and if for every function $g \in \mathcal{C}$ there is a constant $c_{f,g}$ such that $C_f(x) \leq C_g(x) + c_{f,g}$, for all x. (Here $c_{f,g}$ depends on f and g, but not on x.) Replacing x by $\langle x, y \rangle$, with $\langle \cdot \rangle$ the standard recursive bijective pairing function, yields the definition for a class of two-variable functions.

Clearly, all additively optimal methods f, g of specifying objects in S are equivalent in the following way:

$$|C_f(x) - C_g(x)| \leq c_{f,g},$$

for all x, where $c_{f,g}$ is a constant depending only on f and g. Thus, from an asymptotic point of view, the complexity $C(x)$ of an object x, when we restrict ourselves to optimal methods of specification, does not depend on accidental peculiarities of the chosen optimal method.

Example 2.0.1 Consider the class of description methods consisting of *all* partial functions over the nonnegative integers. Every additively optimal function f for this class must be unbounded. Take an infinite sequence $x_1, x_2, \ldots$ such that $C_f(x_i) \geq i$. Define the function g by $g(i) = x_i$. Clearly, $C_g(x_i) = \log i + O(1) \ll C_f(x_i)$. Therefore, f cannot be additively optimal. Thus, there is no additively optimal partial function, and the hierarchy of complexities with respect to the partial functions does not have *any* minimal element.

The development of the theory of Kolmogorov complexity is made possible by the remarkable fact that the class of partial *recursive* functions (defined in Section 1.7) possesses a universal element that is additively optimal. Under this relatively natural restriction on the class of description methods (that is, to partial recursive functions) we obtain a well-behaved hierarchy of complexities. ◇

We begin by worrying about notation. There are several variants of Kolmogorov complexity, with notations that are not used consistently among different authors or even by the same author at different times.

In the main text of this book we shall concentrate on two major variants of Kolmogorov complexity. It seems educationally the right approach to first study Kolmogorov complexity as originally defined by Solomonoff, Kolmogorov, and Chaitin because it is intuitively clearer.

Some mathematical technicalities will naturally lead up to, and justify, the less intuitive version of Kolmogorov complexity. The first type we call *plain* Kolmogorov complexity, and the second type we call *prefix* Kolmogorov complexity. We use C to denote the plain Kolmogorov complexity. We reserve K for the prefix type. Fortunately, the majority of theorems we derive for plain Kolmogorov complexity carry over unchanged and with the same proofs to the prefix version. The difference is that the prefix version is tweaked to have just the right quantitative properties for some desired uses and applications.

2.1 The Invariance Theorem

Identify an object x from a countably infinite sample space S with its index $n(x)$. Consider the class of description methods

$$\{\phi : \phi \text{ is a partial recursive function}\}.$$

Consider the particular problem of describing objects consisting of natural numbers in terms of programs consisting of finite strings of 0's and 1's. Just as in information theory, Section 1.11, where the entropy and information of a message over an alphabet of any size are expressed in the normalized format of bits, the restriction of the programs to a binary alphabet does not imply any loss of generality. In both cases, changing alphabet size leaves all statements invariant up to an appropriate logarithmic multiplicative factor related to the alphabet sizes involved; see Exercise 2.1.9.

The *invariance theorem*, Theorem 2.1.1 below, is the cornerstone for the subsequent development of the theory. In fact, for many later applications it embodies the *entire* theoretical foundation. Recall Definition 2.0.1 of a function that is additively optimal (a special type of universality) for a class of functions. We give the unconditional version as a preliminary lemma.

Lemma 2.1.1 *There is an additively optimal universal partial recursive function.*

Proof. Let ϕ_0 be the function computed by a universal Turing machine U. Machine U expects inputs of the format

$$\langle n, p\rangle = \underbrace{11\ldots1}_{l(n)\text{ times}}\ 0\, np\,.$$

The interpretation is that the total program $\langle n, p\rangle$ is a two-part code of which the first part consists of a self-delimiting encoding of T_n and the

second part is the literally rendered program p. In this way, U can first parse the binary input into the T_n-part and the p-part, and subsequently simulate the computation of T_n started with program p as its input (Section 1.7). That is, $\phi_0(\langle n,p\rangle) = \phi_n(p)$. What happens if U gets the program $0p$? By convention we can set $U = T_0$ and therefore $U(0p) = U(p)$. Altogether, if T_n computes the partial recursive function ϕ_n, then

$$C_{\phi_0}(x) \leq C_{\phi_n}(x) + c_{\phi_n},$$

where c_{ϕ_n} can be set to $2l(n)+1$. □

For many applications we require a generalization to a conditional version, as follows. The difficulty of specifying an object can be facilitated when another object is already specified. We define the complexity of an object x, given an object y. Fix an effective enumeration of Turing machines $T_1, T_2, \ldots$ as in Section 1.7. The Turing machines use a tape alphabet $\{0,1,B\}$, and the input to a Turing machine is a program consisting of a contiguous string of 0's and 1's, delimited by blanks B on both sides. In this way, a Turing machine can detect the end of its program. The effective enumeration of Turing machines induces an effective enumeration of partial recursive functions $\phi_1, \phi_2, \ldots$ such that T_i computes ϕ_i for all i. As above, $\langle\cdot\rangle : \mathcal{N}\times\mathcal{N} \to \mathcal{N}$ is a standard recursive bijective pairing function mapping the pair (x,y) to the singleton $\langle x,y\rangle$. We can iterate this as $(x,y,z) = \langle x,\langle y,z\rangle\rangle$.

Definition 2.1.1 Let x, y, p be natural numbers. Any partial recursive function ϕ, together with p and y, such that $\phi(\langle y,p\rangle) = x$, is a *description* of x. The *complexity* C_ϕ of x *conditional* to y is defined by

$$C_\phi(x|y) = \min\{l(p) : \phi(\langle y,p\rangle) = x\},$$

and $C_\phi(x|y) = \infty$ if there are no such p. We call p a *program* to compute x by ϕ, given y.

Theorem 2.1.1 *There is an additively optimal universal partial recursive function ϕ_0 for the class of partial recursive functions to compute x given y. Therefore, $C_{\phi_0}(x|y) \leq C_\phi(x|y) + c_\phi$ for all partial recursive functions ϕ and all x and y, where c_ϕ is a constant depending on ϕ but not on x or y.*

Proof. Let ϕ_0 be the function computed by a universal Turing machine U such that U started on input $\langle y,\langle n,p\rangle\rangle$ simulates T_n on input $\langle y,p\rangle$ (Section 1.7). That is, if T_n computes the partial recursive function ϕ_n, then $\phi_0(\langle y,\langle n,p\rangle\rangle) = \phi_n(\langle y,p\rangle)$. Hence, for all n,

$$C_{\phi_0}(x|y) \leq C_{\phi_n}(x|y) + c_{\phi_n},$$

where $c_{\phi_n} = 2l(n)+1$. □

The key point is not that the universal description method necessarily gives the shortest description in each case, but that no other description method can improve on it infinitely often by more than a fixed constant. Note also that the optimal complexity $C_{\phi_0}(x|y)$ is defined for all x and y. Namely, for each x and y we can find a Turing machine that computes output x, given y, for some input p (such as the Turing machine that outputs x for all inputs).

For *every* pair ψ, ψ' of additively optimal functions, there is a fixed constant $c_{\psi,\psi'}$, depending only on ψ and ψ', such that for all x, y we have

$$|C_\psi(x|y) - C_{\psi'}(x|y)| \leq c_{\psi,\psi'}.$$

To see this, first substitute $\phi_0 = \psi$ and $\phi = \psi'$ in Theorem 2.1.1, then substitute $\phi = \psi$ and $\phi_0 = \psi'$ in Theorem 2.1.1, and combine the two resulting inequalities. While the complexities according to ψ and ψ' are not exactly equal, they are *equal up to a fixed constant* for all x and y.

Definition 2.1.2 Fix an additively optimal universal ϕ_0 and dispense with the subscript by defining the *conditional Kolmogorov complexity* $C(\cdot|\cdot)$ by

$$C(x|y) = C_{\phi_0}(x|y).$$

This particular ϕ_0 is called the *reference function* for C. We also fix a particular Turing machine U that computes ϕ_0 and call U the *reference machine*. The *unconditional Kolmogorov complexity* $C(\cdot)$ is defined by

$$C(x) = C(x|\epsilon).$$

Example 2.1.1 Programmers are generally aware that programs for symbolic manipulation tend to be shorter when they are expressed in the LISP programming language than if they are expressed in FORTRAN, while for numerical calculations the opposite is the case. Or is it? The invariance theorem in fact shows that to express an algorithm succinctly in a program, it does not matter which programming language we use (up to a fixed additive constant that depends only on the two programming languages).

To see this, as an example consider the lexicographic enumeration of all syntactically correct LISP programs $\lambda_1, \lambda_2, \ldots$ and the lexicographic enumeration of all syntactically correct FORTRAN programs $\pi_1, \pi_2, \ldots$. With proper definitions we can view the programs in both enumerations as computing partial recursive functions from their inputs to their outputs. Choosing reference machines in both enumerations, we can define complexities $C_{\mathrm{LISP}}(x)$ and $C_{\mathrm{FORTRAN}}(x)$ completely analogous to $C(x)$. All of these measures of the descriptional complexity of x coincide up

to a fixed additive constant. Let us show this directly for $C_{\rm LISP}(x)$ and $C_{\rm FORTRAN}(x)$.

It is well known and also easy to see that each enumeration contains a universal program; the LISP enumeration contains a LISP interpreter program that interprets any LISP program. But there is also a LISP program λ_P that is a FORTRAN interpreter in the sense that it interprets any FORTRAN program. Consequently, $C_{\rm LISP}(x) \leq C_{\rm FORTRAN}(x) + l(\lambda_P)$. Similarly, there is a FORTRAN program π_L that is a LISP interpreter, which yields $C_{\rm FORTRAN}(x) \leq C_{\rm LISP}(x) + l(\pi_L)$. Consequently, $|C_{\rm LISP}(x) - C_{\rm FORTRAN}(x)| \leq l(\lambda_P) + l(\pi_L)$ for all x. ◇

Example 2.1.2 In Theorem 2.1.1 we used a special type of universal partial recursive function, called 'additively optimal.' There are other universal partial recursive functions that are not additively optimal and for which the theorem does not hold. For example, let ϕ be the function computed by a universal Turing machine U_ϕ such that U_ϕ started on input $\langle y, \langle n, pp\rangle\rangle$ simulates T_n on input $\langle y, p\rangle$, and ϕ is not defined for inputs that are not of the form $\langle y, \langle n, pp\rangle\rangle$. (That is, if T_n computes the partial recursive function ϕ_n, then $\phi(\langle y, \langle n, pp\rangle\rangle) = \phi_n(\langle y, p\rangle)$.) Then, for all x, y, n, we have $C_\phi(x|y) \geq 2C_{\phi_n}(x|y)$. ◇

2.1.1 Two-Part Codes

It is a deep and useful fact that the shortest effective description of an object x can be expressed in terms of a *two-part code*, the first part describing an appropriate Turing machine and the second part describing the program that interpreted by the Turing machine reconstructs x. The essence of the invariance theorem is as follows: For the fixed reference universal Turing machine U, the length of the shortest program to compute x is $\min\{l(p) : U(p) = x\}$. Looking back at the proof of Lemma 2.1.1, we notice that $U(0p) = U(p)$. From the definitions it therefore follows that

$$C(x) = \min\{l(T) + l(p) : T(p) = x\} + O(1),$$

where $l(T)$ is the length of a self-delimiting encoding for a Turing machine T. This provides an alternative definition of Kolmogorov complexity (similarly, for conditional Kolmogorov complexity). The above expression for Kolmogorov complexity can be rewritten as

$$C(x) = \min\{l(T) + C(x|T) : T \in \{T_0, T_1, \ldots\}\} + O(1), \tag{2.1}$$

which emphasizes the two-part-code nature of Kolmogorov complexity, using the regular aspects of x to maximally compress. In the example

$$x = 10101010101010101010101010$$

we can encode x by a small Turing machine that computes x from the program 13. Intuitively, the Turing machine part of the code squeezes out the *regularities* in x. What is left are irregularities, or *random aspects*, of x relative to that Turing machine. The minimal-length two-part code squeezes out regularity only insofar as the reduction in the length of the description of random aspects is greater than the increase in the regularity description.

The right model is a Turing machine T among those that reach the minimum description length

$$\min_T \{l(T) + C(x|T) : T \in \{T_0, T_1, \ldots\}\}.$$

This T embodies the amount of useful information contained in x. The main remaining question is which such T to select among those that satisfy the requirement. The problem is how to separate a shortest program x^* for x into parts $x^* = pq$ such that p represents an appropriate T. This idea has spawned the 'minimum description length' principle in statistics and inductive reasoning, Section 5.4; Kolmogorov's structure functions and algorithmic (minimal) sufficient statistic, Section 5.5; and the notion of algorithmic entropy in Section 8.6.

2.1.2 Upper Bounds

Theorem 2.1.1 has a wider importance than just showing that the hierarchy of C_ϕ complexity measures contains an additively optimal one. It is also our principal tool in finding upper bounds on $C(x)$. Such upper bounds depend on the choice of reference function, and hence are proved only to within an additive constant.

Intuitively, the Kolmogorov complexity of a binary string cannot exceed its own length, because the string is obviously a (literal) description of itself.

Theorem 2.1.2 *There is a constant c such that for all x and y,*

$$C(x) \leq l(x) + c \text{ and } C(x|y) \leq C(x) + c.$$

Proof. The first inequality is supremely obvious: define a Turing machine T that copies the input to the output. Then for all x, we have $C_T(x) = l(x)$. By Theorem 2.1.1 the result follows.

To prove the second inequality, construct a Turing machine T that for all y, z computes output x on input $\langle z, y\rangle$ iff the universal reference machine U computes output x for input $\langle z, \epsilon\rangle$. Then $C_T(x|y) = C(x)$. By Theorem 2.1.1, there is a constant c such that $C(x|y) \leq C_T(x|y)+c = C(x) + c$. □

Note that the additive constants in these inequalities are fudge terms related to the reference machine U. For example, we need to indicate

to the reference machine that a given description is the object itself, and this information adds a number of bits to the literal description. In Section 3.2 we will calculate the constants explicitly as 8 and 2, respectively. Let us look at some more examples in order to develop our intuition about the notion of complexity of description.

Example 2.1.3 For each finite binary string x we have $C(xx) \leq C(x) + O(1)$. Construct a Turing machine V such that $V(p) = U(p)U(p)$, for all programs p, where U is the reference machine in the proof of Theorem 2.1.1. In particular, if $U(p) = x$, then $V(p) = xx$. Let $V = T_m$ in the standard enumeration of Turing machines $T_1, T_2, \ldots$. With $\overline{m}$ denoting the self-delimiting description $1^{l(m)}0m$ of m, we have $U(\overline{m}p) = T_m(p) = xx$ and $l(\overline{m}p) = l(p) + 2l(m) + 1$. Hence, $C(xx) \leq C(x) + 2l(m) + 1$. From now on we leave the more obvious details of this type of argument for the reader to fill in. ◇

Example 2.1.4 Recall that x^R denotes the reverse of x. Clearly, the complexities of x and x^R can differ by at most a fixed constant c independent of x. That is, $|C(x) - C(x^R)| < c$ holds for all x. We can generalize this example as follows: For every total recursive function ϕ that is one-to-one there is (another) constant c such that $|C(\phi(x)) - C(x)| < c$ for all x.

In fact, if ϕ is computed by Turing machine T_n and $U(p) = x$, then there is a Turing machine V such that $V(\bar{n}p) = \phi(x)$. If $V = T_m$, then $U(\overline{m}\bar{n}p) = \phi(x)$, and therefore $|C(\phi(x)) - C(x)| < 2l(m) + 2l(n) + 2$. Similar relations hold for the conditional complexity $C(x|y)$. ◇

Example 2.1.5 Can the complexity of a pair of strings exceed the sum of the complexities of the individual strings? In other words, is C *subadditive*? Let $\langle\cdot\rangle : \mathcal{N}\times\mathcal{N} \to \mathcal{N}$ be the standard recursive bijection over the natural numbers that encodes x and y as $\langle x, y\rangle$. Define $C(x, y) = C(\langle x, y\rangle)$. That is, up to a fixed constant, $C(x, y)$ is the length of the shortest program such that U computes both x and y and a way to tell them apart. It is seductive to conjecture $C(x, y) \leq C(x) + C(y) + O(1)$, the obvious (but false) argument running as follows: Suppose we have a shortest program p to produce x, and a shortest program q to produce y. Then with $O(1)$ extra bits to account for some Turing machine T that schedules the two programs, we have a program to produce x followed by y. However, any such T will have to know where to divide its input to identify p and q. One way to do this is by using input $\overline{l(p)}pq$ or input $\overline{l(q)}qp$. In this way, we can show that for all x, y, we have

$$C(x, y) \leq C(x) + C(y) + 2\log(\min(C(x), C(y))). \tag{2.2}$$

We cannot eliminate the logarithmic error term for the general case. Namely, in Example 2.2.3 on page 118 we show that there is a constant

c such that for all n there are x and y of length at most n such that

$$C(x, y) \geq C(x) + C(y) + \log n - c.$$

We can eliminate the logarithmic error term at the cost of entering the length of one of the programs in the conditional,

$$C(x, y|C(x)) \leq C(x) + C(y) + O(1).$$

Equation 2.2 also holds if we replace the left-hand side by the complexity $C(xy)$ of the unmarked concatenation xy. In the example already referred to above, it is shown that we cannot eliminate the logarithmic error in this case either. ◇

Example 2.1.6 If we know $C(x)$ and x, then we can run all programs of length $C(x)$ in parallel on the reference machine U in dovetail fashion (in stage k of the overall computation execute the ith computation step of program $k-i$). By definition of $C(\cdot)$, there must be a program of length $C(x)$ that halts with output x. The first such program is the *first shortest program* for x in enumeration order, and is denoted by x^*.

Therefore, a program to compute $C(x)$, given x, can be converted to a program to compute x^*, given x, at the cost of a constant number of extra bits. If we have computed x^*, then $C(x)$ is simply its length, so the converse is trivial. Furthermore, to describe $C(x)$ from scratch takes at least as many bits as to describe $C(x)$ using x. Altogether we have, up to additional constant terms,

$$C(x^*|x) = C(C(x)|x) \leq C(C(x)) \leq \log l(x).$$

◇

The upper bound on $C(x^*|x)$ cannot be improved to $O(1)$. If it could, then one could show that $C(x)$ is a recursive function. However, in Theorem 2.3.2 we shall show that $C(x)$ is not partial recursive. It is a curious fact that for some x, knowledge of x does not help much in computing x^*. In fact, the upper bound is nearly optimal. In Theorem 3.8.1 we shall show that for some x of each length n the quantity $C(C(x)|x)$, and hence also $C(x^*|x)$, is almost $\log n$.

Clearly, the information that an element belongs to a particular set can severely curtail the complexity of that element. The following simple observation, due to Kolmogorov, turns out to be very useful. We show that for every easily describable set the conditional complexity of every one of its elements is at most equal to the logarithm of the cardinality of that set. (We will observe later, in Theorem 2.2.1, that the conditional complexities of the majority of elements in a finite set cannot be significantly less than the logarithm of the cardinality of that set: we will say that they are 'incompressible' and have a small 'randomness deficiency.')

Theorem 2.1.3 *Let $A \subseteq \mathcal{N} \times \mathcal{N}$ be recursively enumerable, and $y \in \mathcal{N}$. Suppose $Y = \{x : (x, y) \in A\}$ is finite. Then, for some constant c depending only on A, for all x in Y, we have $C(x|y) \leq l(d(Y)) + c$.*

Proof. Let A be enumerated without repetition as $(x_1, y_1), (x_2, y_2), \ldots$ by a Turing machine T. Let $(x_{i_1}, y_{i_1}), \ldots, (x_{i_k}, y_{i_k})$ be the subsequence in which the elements of Y are enumerated, $k = d(Y)$. Using the fixed y, modify T to T_y such that T_y, on input $1 \leq p \leq d(Y)$, outputs x_{i_p}, $T_y(p) = x_{i_p}$. Therefore, we have by the invariance theorem, Theorem 2.1.1, that $C(x|y) \leq C_{T_y}(x) + c \leq l(d(Y)) + c$, with c depending only on A. □

Let us illustrate the use of this theorem. Let A be a subset of $\mathcal{N}$. Define $A^{\leq n} = \{x \in A : l(x) \leq n\}$. Let A be recursively enumerable and $d(A^{\leq n}) \leq p(n)$, with p a polynomial. Then, for all $x \in A$ of length at most n we have $C(x|n) \leq l(p(n)) + O(1)$, by Theorem 2.1.3. For all x of length at most n we have $C(x) \leq C(x|n) + 2l(n) + O(1)$. Therefore, for $x \in A^{\leq n}$ we find that $C(x) = O(\log n)$.

2.1.3 Invariance of Kolmogorov Complexity

The complexity $C(x)$ is invariant only up to a constant depending on the reference function ϕ_0. Thus, one may object, for *every* string x there is an additively optimal recursive function ψ_0 such that $C_{\psi_0}(x) = 0$. So how can one claim that $C(x)$ is an objective notion?

A mathematically clean solution to this problem is as follows: Call two complexities C_ϕ and C_ψ *equivalent*, $C_\phi \equiv C_\psi$, if there is a constant c such that for all x,

$$|C_\phi(x) - C_\psi(x)| \leq c.$$

Then the equivalence relation $\equiv$ induces equivalence classes

$$[C_\phi] = \{C_\psi : C_\psi \equiv C_\phi\}.$$

We order the equivalence classes by $[C_\phi] \leq [C_\psi]$ if there is a constant $c \geq 0$ such that $C_\phi(x) \leq C_\psi(x) + c$ for every x. The resulting order on the equivalence classes is a partial order with a single minimal element, namely $[C_{\phi_0}]$, such that for all C_ψ,

$$[C_{\phi_0}] \leq [C_\psi].$$

We have somewhat glibly overlooked the fact that our definition of Kolmogorov complexity is relative to the particular effective enumeration of Turing machines as used in the proof of the invariance theorem, Theorem 2.1.1. We have claimed that the quantity of information in an object depends on itself alone. That is, it should be independent of the particular enumeration of Turing machines.

Consider two different enumerations of all partial recursive functions, say $\phi_1, \phi_2, \ldots$ and $\psi_1, \psi_2, \ldots$. Assume that the ϕ enumeration is the enumeration corresponding to our effective enumeration of Turing machines as used in the proof of the invariance theorem.

Let the standard enumeration $\phi_1, \phi_2, \ldots$ and the other enumeration $\psi_1, \psi_2, \ldots$ be related by $\psi_i = \phi_{f(i)}$ and $\phi_i = \psi_{g(i)}$, $i = 1, 2, \ldots$. If both f and g are partial recursive, then the enumerations are called *recursively isomorphic* and are both *acceptable numberings* (Section 1.7, Exercise 1.7.6 on page 41). Let $C(x)$ be the complexity with respect to the reference function in the ϕ enumeration, and let $C'(x)$ be the complexity with respect to the reference function in the ψ enumeration. It is an easy exercise to show that there is a constant c such that $|C(x) - C'(x)| < c$ for all x. (Hint: use the indexes of f and g in the enumerations.)

Therefore, not only do additively optimal functions in the *same* acceptable numberings yield complexities that are equal up to a fixed constant, but additively optimal functions in two *different* acceptable numberings do so as well. Hence, Kolmogorov complexity is *recursively invariant* between acceptable numberings, even though we have chosen to define it using the specific enumeration of Turing machines of Section 1.7. Using an analogy due to Hartley Rogers, Jr., the fixed choice of effective enumeration of Turing machines can be compared with using a particular coordinate system to establish coordinate-free results in geometry.

A contradiction is possible only if there is no recursive isomorphism between the ϕ enumeration and the ψ enumeration. We give an example of an enumeration of all partial recursive functions for which an additively optimal function yields a complexity $C'(x)$ such that $|C(x) - C'(x)|$ is unbounded. Let $C(x)$ be defined with respect to the ϕ enumeration as in Theorem 2.1.1. Define the ψ enumeration as follows: The even functions ψ_{2i} are defined by $\psi_{2i}(1) := y_i$ for some y_i with $C(y) \geq i^2$ and $\psi_{2i}(x) := \phi_i(x)$ for all $x > 1$. The odd functions ψ_{2i+1} are given by $\psi_{2i+1} := \phi_i$.

Clearly, the ψ enumeration contains all partial recursive functions. By way of contradiction, assume that $C'(\cdot)$ is the Kolmogorov complexity in the ψ-enumeration defined as in Theorem 2.1.1. Then, $C'(y_i) \leq C'_{\psi_{2i}}(y_i) + c_{\psi_{2i}}$. By construction, $C'_{\psi_{2i}}(y_i) = 1$ and $c_{\psi_{2i}} \leq 2 \log 2i + O(1)$. On the other hand, $C(y_i) > i^2$ by construction. Hence, $|C'(y_i) - C(y_i)|$ rises unboundedly with i.

2.1.4 Concrete Kolmogorov Complexity

It is possible to eliminate the indeterminacy of 'equality up to a constant' everywhere by using a fixed domain of objects, a fixed effective enumeration of Turing machines, and a fixed choice of additively optimal function (rather the universal Turing machine that computes it). Start from the enumeration of Turing machines in Section 1.7. Fix any small universal machine, say U, with state–symbol product less than 30. There exists at least one 7×4 universal Turing machine as mentioned in the comment on page 31 following Example 1.7.4. In Section 3.2 we exhibit a universal reference machine U to fix a concrete Kolmogorov complexity with $C(x|y) \leq l(x) + 2$ and $C(x) \leq l(x) + 8$.

For every x it is of course possible to choose a universal Turing machine U' such that $C_{U'}(x) = 0$ (in this notation identifying U' with the function it computes). For every such universal Turing machine U', we have for all x that

$$C(x) \leq C_{U'}(x) + C(U').$$

Here $C(U')$ is at least the length of the shortest program p such that for all programs q we have $U(pq) = U'(q)$. This means that if $C_{U'}(x) = 0$, then $C(U') \geq C(x)$. That is, $C_{U'}(x) = 0$ unavoidably means that the description of U' contains a description of x. Therefore, in order to assign low complexity to a large and complicated object, a universal machine has to be large and complicated as well.

Exercises

2.1.1. [15] (a) Show that $C(0^n|n) \leq c$, where c is a constant independent of n.

(b) Show that $C(\pi_{1:n}|n) \leq c$, where $\pi = 3.1415\ldots$ and c is some constant independent of n.

(c) Show that we can expect $C(a_{1:n}|n) \leq \frac{1}{4}n$, where a_i is the ith bit in Shakespeare's *Romeo and Juliet*.

(d) What is $C(a_{1:n}|n)$, where a_i is the ith bit in the expansion of the fine structure constant $a = e^2/\hbar c$, in physics.

Comments. Hint: for Item (c) use known facts concerning the letter frequencies (entropy) in written English. Source: T.M. Cover, *The Impact of Processing Technique on Communications,* J.K. Skwirzynski, ed., Martinus Nijhof, 1985, pp. 23–33.

2.1.2. [10] Let x be a finite binary string with $C(x) = q$. What is the complexity $C(x^q)$, where x^q denotes the concatenation of q copies of x?

2.1.3. [14] Show that there are infinite binary sequences ω such that the length of the shortest program for reference Turing machine U to compute the consecutive digits of ω one after another can be significantly shorter than the length of the shortest program to compute an initial n-length segment $\omega_{1:n}$ of ω, for any large enough n.

Comments. Hint: choose ω a recursive sequence with shortest program of length $O(1)$. Then $C(\omega_{1:n}) = C(n) + O(1)$, which goes to ∞ with n.

2.1.4. [12] Prove that for every x, there is an additively optimal function ϕ_0 (as in Theorem 2.1.1) such that $C_{\phi_0}(x) = 0$. Prove the analogous statement for x under condition y.

2.1.5. [07] Below, x, y, and z are arbitrary elements of $\mathcal{N}$. Prove the following:

(a) $C(x|y) \leq C(x) + O(1)$.

(b) $C(x|y) \leq C(x, z|y) + O(1)$.

(c) $C(x|y, z) \leq C(x|y) + O(1)$.

(d) $C(x, x) = C(x) + O(1)$.

(e) $C(x, y|z) = C(y, x|z) + O(1)$.

(f) $C(x|y, z) = C(x|z, y) + O(1)$.

(g) $C(x, y|x, z) = C(y|x, z) + O(1)$.

(h) $C(x|x, z) = C(x|x) + O(1) = O(1)$.

2.1.6. [14] Let ϕ_k be any partial recursive function in the effective enumeration $\phi_1, \phi_2, \ldots$. Let x, y, z be arbitrary elements of $\mathcal{N}$. Prove the following:

(a) $C(\phi_k(x)|y) \leq C(x|y) + 2l(k) + O(1)$.

(b) $C(y|\phi_k(x)) \geq C(y|x) - 2l(k) + O(1)$.

Assume that ϕ_k is also one-to-one. Show that

(c) $|C(x) - C(\phi_k(x))| \leq 2l(k) + O(1)$.

(d) $C(x|y, z) \leq C(x|\phi_k(y), z) + 2l(k) + O(1)$.

2.1.7. [12] Let x, y, z, and ϕ_k be as before. Prove the following.

(a) $C(x, y) \leq C(x) + 2l(C(x)) + C(y|x) + O(1)$.

(b) $C(\phi_k(x, y)) \leq C(x) + 2l(C(x)) + C(y|x) + 2l(k) + O(1) \leq C(x) + 2l(C(x)) + C(y) + 2l(k) + O(1)$.

2.1.8. [12] Show that if ϕ is a fixed one-to-one and onto recursive function $\phi : \{0,1\}^* \to \{0,1\}^*$, then for every $x \in \{0,1\}^*$,

$$C(x) - C(x|\phi(x)) = C(x) + O(1) = C(\phi(x)) + O(1).$$

2.1.9. • [19] We investigate the invariance of C under change of program representations from 2-ary to r-ary representations. Let $A_r = \{0, 1, \ldots, r-1\}^*$, $r \geq 2$, and $A = \mathcal{N}^*$ with $\mathcal{N}$ the set of natural numbers. A function $\phi : A_r \times A \to A$ is called an r-ary *decoder*. In order not to hide too much information in the decoder, we want it to be a simple function, a partial recursive one. Analogous to the definitions in the main text, for any binary decoder ϕ and x, y in A,

$$C_\phi(x|y) = \min\{l(p) : \phi(p, y) = x\},$$

or ∞ if such p does not exist.

(a) Prove Theorem 2.1.1 under this definition of C.

(b) Define for each pair of natural numbers $r, s \geq 2$ a standard encoding E of strings x in base r to strings $E(x)$ in base s such that $l(E(x)) \leq l(x) \log r / \log s + 1$.

(c) Prove the invariance theorem, Theorem 2.1.1, for r-ary decoders ϕ. First, let us define $C_\phi(x|y) = \min\{l(p) \log r : \phi(p, y) = x\}$ and $C_\phi(x|y) = \infty$ if such p does not exist. Then prove that there exists an additively optimal (universal) r-ary decoder ϕ_0 such that for all s, for all s-ary decoders ϕ, there exists a constant c_ϕ such that for all $x, y \in A$ we have

$$C_{\phi_0}(x|y) \leq C_\phi(x|y) + c_\phi.$$

(d) Show that for any $x \in A_r$ of length n, we have $C(x) \leq n \log r + 2 \log r + c$ for some fixed c, independent of x and r.

(e) Fix natural numbers $r, s \geq 2$ and choose an additively optimal r-ary decoder and an additively optimal s-ary decoder. Call the associated canonical C measures respectively C_r and C_s. Show that there exists a constant c such that for all x, y in A we have

$$|C_r(x|y) - C_s(x|y)| \leq c,$$

where c is independent of x and y. Conclude that C_2, the C measure treated in the main text, is universal in the sense that neither the restriction to binary objects to be described nor the restriction to binary descriptions (programs) results in any loss of generality.

Comments. In general, if we denote by $C_r(x)$ the analogous complexity of x in terms of programs over alphabets of r letters ($C_2(x) = C(x)$ but for $r > 2$ without the $\log r$ normalizing multiplicative factor as in Item (c)), then by the same analysis as of Item (c) we obtain $C_r(x) \sim C(x)/\log r$. Source: P. Gács, *Lecture Notes on Descriptional Complexity and Randomness*, Manuscript, Boston University, 1987.

2.1.10. [12] (a) Show that $C(x + C(x)) \leq C(x) + O(1)$.

(b) Show that if $m \leq n$, then $m + C(m) \leq n + C(n) + O(1)$.

Comments. Hint for Item (a): if $U(p) = x$ with $l(p) = C(x)$, then p also suffices to reconstruct $x + l(p)$. Hint for Item (b): use Item (a). Source: P. Gács, *Lecture Notes on Descriptional Complexity and Randomness*, Manuscript, Boston University, 1987; result is attributed to C.P. Schnorr.

2.1.11. [13] Let $\phi_1, \phi_2, \ldots$ be the standard enumeration of the partial recursive functions, and let a be a fixed natural number such that the set $A = \{x : \phi_k(y) = \langle a, x \rangle \text{ for some } y \in \mathcal{N}\}$ is finite. Show that for each x in A we have $C(x|a) \leq l(d(A)) + 2l(k) + O(1)$.

2.1.12. [18] Define the *function complexity* of a function $f : \mathcal{N} \to \mathcal{N}$, restricted to a finite domain D, as

$$C(f|D) = \min\{l(p) : \forall_{x \in D}[U(p, x) = f(x)]\}.$$

(a) Show that for all recursive functions f, there exists a constant c_f such that for all finite $D \subseteq \mathcal{N}$, we have $C(f|D) \leq c_f$.

(b) Show that for all partial recursive functions, for all $D = \{i : i \leq n\}$, we have $C(f|D) \leq \log n + c_f$, where c_f depends on f but not on D.

Comments. Compare Theorem 2.7.2. Source: J.M. Barzdins, *Soviet Math. Dokl.*, 9(1968), 1251–1254.

2.2 Incompressibility

It is easy to see that there are strings that can be described by programs much shorter than themselves. For instance, the function defined by $f(1) = 2$ and $f(i) = 2^{f(i-1)}$ for $i > 1$ grows very fast, $f(k)$ is a stack of k twos. Yet for each k it is clear that the string $x = 1^{f(k)}$, or the integer $y = 2^{f(k)}$, has at most complexity $C(k) + c$ for some constant c independent of k.

Trivially, this simple argument can be generalized to prove the following fact: for every recursive function ϕ, no matter how fast it grows, there is a constant c such that for each value of n there is a string x such that $l(x) = \phi(n)$ but $C(x) \leq n + c$. That is, for an appropriate sequence of strings, the ratio of string length to description length can increase as fast as any recursive function—some strings are very *compressible*.

What about incompressibility? By a simple counting argument one can show that whereas some strings can be greatly compressed, the majority of strings cannot be compressed at all.

For each n there are 2^n binary strings of length n, but only $\sum_{i=0}^{n-1} 2^i = 2^n - 1$ possible shorter descriptions. Therefore, there is at least one binary string x of length n such that $C(x) \geq n$. We call such strings *incompressible*. It also follows that for any length n and any binary string y, there is a binary string x of length n such that $C(x|y) \geq n$.

Definition 2.2.1 For each constant c we say that a string x is *c-incompressible* if $C(x) \geq l(x) - c$.

Strings that are incompressible (say, c-incompressible with small c) are patternless, since a pattern could be used to reduce the description length. Intuitively, we think of such patternless sequences as being random, and we use 'random sequence' synonymously with 'incompressible sequence.' Later we give a formalization of the intuitive notion of a random sequence as a sequence that passes all effective tests for randomness.

How many strings of length n are c-incompressible? By the same counting argument we find that the number of strings of length n that are c-incompressible is at least $2^n - 2^{n-c} + 1$. Hence there is at least one 0-incompressible string of length n, at least one-half of all strings of length n are 1-incompressible, at least three-fourths of all strings of length n are 2-incompressible, ..., and at least the $(1 - 1/2^c)$th part of all 2^n strings of length n are c-incompressible. This means that for each constant $c > 1$ the majority of all strings of length n with $n > c$ are c-incompressible. We generalize this to the following simple but extremely useful *incompressibility theorem*.

Theorem 2.2.1 *Let c be a positive integer. For each fixed y, every finite set A of cardinality m has at least $m(1-2^{-c})+1$ elements x with $C(x|y) \geq \log m - c$.*

Proof. The number of programs of length less than $\log m - c$ is

$$\sum_{i=0}^{\log m - c - 1} 2^i = 2^{\log m - c} - 1.$$

Hence, there are at least $m - m2^{-c} + 1$ elements in A that have no program of length less than $\log m - c$. □

As an example, set $A = \{x : l(x) = n\}$. Then the cardinality of A is $m = 2^n$. Since Theorem 2.1.2 asserts that $C(x) \leq n + c$ for some fixed c and all x in A, Theorem 2.2.1 demonstrates that this trivial estimate is quite sharp. The deeper reason is that since there are few short programs, there can be only few objects of low complexity.

It is important to realize that Theorem 2.1.1 and Theorem 2.2.1, together with the trivial upper bound of Theorem 2.1.2, give us already all we need for most applications.

Example 2.2.1 Are all substrings of incompressible strings also incompressible? A string $x = uvw$ of length n can be specified by a short program p for v and the string uw itself. Additionally, we need information on how to tell these items apart. For instance, $q = \overline{l(p)}p\overline{l(u)}uw$ is a program for x. There exists a machine T that starting on the left end of q, first determines $l(p)$, then uses $l(p)$ to delimit p, and computes v from p. Continuing on its input, T determines $l(u)$ and uses this to delimit u on the remainder of its input. Subsequently, T reassembles x from the three pieces v, u, and w it has determined. It follows that $C(x) \leq C_T(x) + O(1) \leq l(q) + O(1)$, since $l(q) \leq C(v) + 2l(C(v)) + 2l(n) + n - l(v) + 2$. Therefore,

$$C(x) \leq C(v) + n - l(v) + 4\log n + O(1).$$

Hence, for c-incompressible strings x with $C(x) \geq n - c$ we obtain

$$C(v) \geq l(v) - O(\log n).$$

Thus, we have shown that v is incompressible up to an additive term logarithmic in n.

Can we hope to prove $C(v) \geq l(v) - O(1)$ for all x and v? If this were true, then x could not contain long regular subsequences, for instance, a subsequence of k zeros has complexity $O(\log k)$ and not $k - O(1)$. However, the very restriction on x of not having long regular subsequences imposes some regularity on x by making it a member of a relatively small set. Namely, we can describe x by stating that it does not contain certain subsequences, followed by x's index in the set that is determined by these constraints. But the set of which x is a member is so small that $C(x)$ drops below $n - c$, and x is compressible. Hence, the very incompressibility of x requires that it have compressible substrings. This corresponds to a fact we know from probability theory: a random sequence must contain long runs of zeros. ◇

Example 2.2.2 If p is a shortest program for x, so that $C(x) = l(p)$, then we would like to assert that p is incompressible. This time, our intuition corresponds with the truth. There is a constant $c > 0$ such that for all strings x we have $C(p) \geq l(p) - c$. For suppose the contrary, and for every constant c there is an x and a shortest program q that generates p with $l(q) < l(p) - c$. Define a universal machine V that works just like the reference machine U, except that V first simulates U on its input to obtain the output, and then uses this output as input on which to simulate U once more. Let $V = T_i$, the ith Turing machine in the standard enumeration. Then, U with input $1^i 0 q$ computes x, and therefore $C(x) < l(p) - c + i + 1$. But this contradicts $l(p) = C(x)$ for $c \geq i + 1$. ◇

Example 2.2.3 We continue Example 2.1.5 on page 109 that $C(x, y)$ is not *subadditive* since the logarithmic term in Equation 2.2 cannot be eliminated. Namely, there are $(n + 1)2^n$ pairs (x, y) of binary strings whose sum of lengths is n. By Theorem 2.2.1 there is a pair (x, y) with $l(x) + l(y) = n$ such that $C(x, y) \geq n + \log n - 1$. But Theorem 2.1.2 on page 108 states that $C(x) + C(y) \leq l(x) + l(y) + c$ for some constant c independent of x and y. Hence, for all n there are x and y of length at most n such that

$$C(x, y) > C(x) + C(y) + \log n - c,$$

where c is a constant independent of x and y. For the unmarked concatenation xy with $l(xy) = n$, if $C(xy) \geq n$, then xy contains a block of 0's or 1's of length at least $\log n - 2 \log \log n - O(1)$ (follows from Example 2.2.1 but is more precisely derived in Corollary 2.6.2 om page 172). We can choose the concatenation xy so that x ends with this longest run of 0's or 1's. This means that $C(x) \leq l(x) - \log n + 2 \log \log n$. Then, $C(xy) \geq C(x) + C(y) + \log n - 2 \log \log n - c$. ◇

There is a particular use we had in mind in defining conditional Kolmogorov complexity. Namely, we often want to speak about the complexity of x given its length n. This is because a string x of length n carries in a sense *two* quantities of information, one associated with the irregularity of the pattern of 0's and 1's in x, and one associated with the length n of x.

Example 2.2.4 One effect of the information quantity associated with the length of strings is that $C(x)$ is *nonmonotonic on prefixes*. This can be due to the information contained in the length of x. That is, for $m < n$ we can still have $C(m) > C(n)$. But then $C(y) > C(x)$ for $x = 1^n$ and $y = 1^m$, notwithstanding that y is a proper prefix of x. For example, if $n = 2^k$, then $C(1^n) \leq \log\log n + O(1)$, while Theorem 2.2.1 shows that there exist m with $\frac{1}{2}n \leq m < n$ such that $C(1^m) \geq \log n - O(1)$. Therefore, the complexity of a part can turn out to be bigger than the complexity of the whole. In an initial attempt to solve this problem we may try to eliminate the effect of the length of the string on the complexity measure by treating the length as given. ◇

Definition 2.2.2 The *length-conditional* Kolmogorov complexity of x is $C(x|l(x))$.

Roughly speaking, this means that the length of the shortest program for x may save up to $\log l(x)$ bits in comparison with the shortest program in the unconditional case. Clearly, there is a constant c such that for all x,

$$C(x|l(x)) \leq C(x) + c.$$

While on the face of it the measure $C(x|l(x))$ gives a pure estimate of the quantity of information in solely the pattern of 0's and 1's of x, this is not always true. Namely, sometimes the information contained in $l(x)$ can be used to determine the pattern of zeros and ones of x. This effect is noticeable especially in the low-complexity region.

Example 2.2.5 For each integer n, the *n-string* is defined by $n0^{n-l(n)}$ (using the binary string n). There is a constant c such that for all n, if x is the n-string, then $C(x|n) \leq c$. Namely, given n we can find the nth binary string according to Equation 1.3 and pad the string with zeros up to overall length n. We use n-strings to show that unfortunately, like the original $C(x)$, the complexity measure $C(x|l(x))$ is *not monotonic over the prefixes*. Namely, if we choose n such that its pattern of 0's and 1's is very irregular, $C(n) \geq l(n)$, then for $x = n0^{n-l(n)}$, we still obtain $C(x|l(x)) \leq c$. But clearly $C(n|l(n)) \geq C(n) - C(l(n)) \geq \log n - 2\log\log n$. ◇

Example 2.2.6 Consider the complexity of a string x, with x an element of a given set A. Clearly, the information that an element belongs to a particular set severely curtails the complexity of that element if that set is small or sparse. The following is a simple application of the very useful Theorem 2.1.3. Let A be a subset of $\mathcal{N}$ and $A^{\leq n} = \{x \in A : l(x) \leq n\}$ We call A *meager* if $\lim d(A^{\leq n})/2^n = 0$ for $n \to \infty$. For example, the set of all finite strings that have twice as many 0's as 1's is meager. We show that meagerness may imply that almost all strings in the meager set have short descriptions.

Claim 2.2.1 If A is recursive and meager, then for each constant c there are only finitely many x in A that are c-incompressible ($C(x) \geq l(x) - c$).

Proof. Consider the lexicographic enumeration of all elements of A. Because A is recursive, there is a total recursive function ϕ_i that enumerates A in increasing order. Hence, for the jth element x of A we have $C(x) \leq C(j) + 2l(i) + 1$. If x has length n, then the meagerness of A implies that for each constant c', no matter how large, $n - C(j) > c'$ from some n onward. Hence, $C(x) < n - c' + 2l(i)$. The proof is completed by setting $c' = c + 2l(i)$. □ ◇

2.2.1 Randomness Deficiency

If we know that x belongs to a subset A of the natural numbers, then we can consider its complexity $C(x|A)$. For instance, $C(x) = C(x|\mathcal{N})$, because it is understood that x is a natural number. If x is an element of a finite set A, then Theorem 2.1.3 asserts that $C(x|A) \leq l(d(A)) + c$ for some c independent of x but possibly dependent on A. For instance, the infinite meager sets of Example 2.2.6 contain finitely many incompressible strings only.

Definition 2.2.3 The *randomness deficiency* of x *relative* to A is defined as $\delta(x|A) = l(d(A)) - C(x|A)$. It follows that $\delta(x|A) \geq -c$ for some fixed constant c independent of x.

If $\delta(x|A)$ is large, then this means that there is a description of x with the help of A that is considerably shorter than just giving x's serial number in A. There are comparatively few objects with large randomness deficiency—this is the substance of Martin-Löf's notion of a statistical test in Section 2.4. Quantitatively this is expressed as follows:

Theorem 2.2.2 *Assume the discussion above. Then,* $d(\{x : \delta(x|A) \geq k\}) \leq d(A)/2^{k-1}$.

Proof. There are fewer than 2^{l+1} descriptions of length at most l. □

By Theorem 2.1.3, the complexity of a string x in a given finite section of a recursively enumerable set is bounded above by the logarithm of the cardinality of that finite section. Let $\langle\cdot\rangle : \mathcal{N}^2 \to \mathcal{N}$ be the standard recursive bijective pairing function. Let $R = \{(x, y) : \phi(i) = \langle x, y\rangle, i \geq 1\}$ with ϕ a partial recursive function, say $\phi = \phi_r$ in the standard enumeration of partial recursive functions. Then R is recursively enumerable. Let the set $A = \{x : (x, y) \in R\}$ be finite. We can assume that A is enumerated without repetition, and that $j \leq d(A)$ is the position of x in this enumeration. Clearly,

$$C(x|y) \leq \log d(A) + \log r + 2 \log\log r + O(1).$$

As above, define $C(x|A) = C(x|y)$ with the obvious interpretation. The randomness deficiency of x relative to y is

$$\delta(x|y) = \log d(A) - C(x|y).$$

The randomness deficiency measures the difference between the maximal complexity of a string in A and the complexity of x in A. Now, the defect of randomness is positive up to a fixed constant independent of x and A (but dependent on r). We may consider x to be random in the set A iff $\delta(x|y) = O(1)$. If A is the set of binary strings of length n, or equivalently, R is the set $\{(x, n) : l(x) = n\}$ and $A = \{x : l(x) = n\}$, then we note that

$$\delta(x|n) = n - C(x|n) + O(1).$$

That is, x is a random finite string in our informal sense iff $\delta(x|n) = O(1)$. It will turn out that this coincides with Martin-Löf's notion of randomness in Section 2.4.

Exercises

2.2.1. [08] Prove the following continuity property of $C(x)$. For all natural numbers x, y we have $|C(x + y) - C(x)| \leq 2l(y) + O(1)$.

2.2.2. [15] Let x satisfy $C(x) \geq n - O(1)$, where $n = l(x)$.

(a) Show that $C(y), C(z) \geq \frac{1}{2}n - O(1)$ for $x = yz$ and $l(y) = l(z)$.

(b) Show that $C(y) \geq n/3 - O(1)$ and $C(z) \geq 2n/3 - O(1)$ for $x = yz$ and $l(z) = 2l(y)$.

(c) Let $x = x_1 \ldots x_{\log n}$ with $l(x_i) = n/\log n$ for all $1 \leq i \leq \log n$. Show that $C(x_i) \geq n/\log n - O(\log\log n)$ for all $1 \leq i \leq \log n$.

2.2.3. [21] Let x satisfy $C(x) \geq n - O(1)$, where $n = l(x)$. Show that for all divisions $x = yz$ we have $n - \log n - 2\log\log n \leq C(y) + C(z)$ and for some divisions we have $C(y) + C(z) \leq n - \log n + \log\log n$.

2.2.4. [23] Assume that the elements of $\{1, \ldots, n\}$ are uniformly distributed with probability $1/n$. Compute the expected value of $C(x)$ for $1 \leq x \leq n$.

Comments. Hint: $\sum_{x=1}^{n} \frac{C(x)}{n} \geq \sum_{i=1}^{\log n} 2^{-i}(1 - \frac{i}{\log n}) = \log n + O(1)$.

2.2.5. [14]We call x an *n-string* if x has length n and $x = n00\ldots0$.

(a) Show that there is a constant c such that for all n, every n-string x has complexity $C(x|n) \leq c$. (Of course, c depends on the reference Turing machine U used to define C.)

(b) Show there is a constant c such that for all n, $C(x|n) \leq c$ for every x in the form of the n-length prefix of $nn\ldots n$.

(c) Let c be as in Item (a). Consider some n and some string x of length n with $C(x|n) \gg c$. Prove that the extension of x to a string $y = x00\ldots0$ of length x has complexity $C(y|x) \leq c$. Conclude that there is a constant c such that each string x, no matter how high its $C(x|l(x))$ complexity, can be extended to a string y with $C(y|l(y)) \leq c$.

Comments. The $C(x)$ measure contains the information about the *pattern* of 0's and 1's in x and information about the *length* n of x. For random such n, the complexity $C(n) = l(n) + O(1)$ is about $\log n$. In this case, about $\log n$ bits of the shortest program p for x will be used to account for x's length. For n's that are easy to compute, this is much less. This seems a minor problem at high complexities, but becomes an issue at low complexities, as follows. If the quantities of information related to the *pattern only* is low, say less than $\log n$, for two strings x and y of length n, then distinctions between these quantities for x and y may get blurred in the comparison between $C(x)$ and $C(y)$ if the quantity of information related to length n dominates in both. The $C(x|l(x))$ complexity was meant to measure the information content of x apart from its length. However, as the present exercise shows, in that case $l(x)$ may contain already the complete description of x up to a constant number of bits. Source: D.W. Loveland, *Inform. Contr.*, 15(1969), 510–526.

2.2.6. [19] (a) Show that there is a constant $d > 0$ such that for every n there are at least $\lfloor 2^n/d \rfloor$ strings x of length n with $C(x|n) \geq n$ and $C(x) \geq n$.

(b) Show that there are constants $c, d' > 0$ such that for every large enough n there are at least $\lfloor 2^n/d' \rfloor$ strings x of length $n - c \leq l(x) \leq n$ with $C(x|n) > n$ and $C(x) > n$.

(c) Assume that we have fixed a reference universal turing machine such that for every n, we have $C(x), C(x|n) \leq n + 1$ for all strings x of length n. Show that in this case Item (b) holds with $l(x) = n$.

Comments. Hint for Item (a): There is a constant $c > 0$ such that for every n and every x of length $l(x) \leq n - c$ we have $C(x|n) \leq n$ by Theorem 2.1.2. Therefore, there are at most $2^n - 2^{n-c+1}$ programs of length $< n$ available as shortest programs for the strings of length n. Hence there is at least one x of length n with $C(x|n) \geq n$. Let there be $m \geq 1$ such strings. Given m and n we can enumerate all $2^n - m$ strings x of length n and complexity $C(x|n) < n$ by dovetailing the running of all programs of length $< n$. The lexicographic first string of length n not in the list satisfies $\log m + O(1) \geq C(x|n) \geq n$. The unconditional result follows similarly by padding the description of x up to length n. Hint for Item (b): For every n there are equally many strings of length $\leq n$ to be described and potential programs of length $\leq n$ to describe them. Since some programs do not halt (Lemma 1.7.5 on page 34) for every large enough n, there exists a string x of length at most n that has $C(x|n), C(x) > n$ (and $C(x|n), C(x) \leq l(x) + c$). The remaining argument is similar to that of Item (a). Source: H. Buhrman, T. Jiang, M. Li, P.M.B. Vitányi, *Theoret. Comput. Sci.*, 235:1(2000), 59–70. Also reported by M. Kummer and L. Fortnow. Compare with the similar Exercise 3.3.1 for prefix Kolmogorov complexity on page 213. In the source of that exercise, some form of the result of the current exercise is attributed to G.J. Chaitin in the early 1970s.

2.2.7. [14] We can extend the notion of c-incompressibility as follows (all strings are binary): Let $g : \mathcal{N} \to \mathcal{N}$ be unbounded. Call a string x of length n *g-incompressible* if $C(x) \geq n - g(n)$. Let $I(n)$ be the number of strings x of length at most n that are g-incompressible. Show that $\lim_{n\to\infty} I(n)/2^{n+1} = 1$.

Comments. Thus, the g-incompressible finite strings have uniform probability going to 1 in the set of strings of length n for $n \to \infty$.

2.2.8. [19] Prove that for each binary string x of length n there is a y equal to x except for one bit such that $C(y|n) \leq n - \log n + O(1)$.

Comments. Hint: the set of binary strings of length n constituting a Hamming code has $2^n/n$ elements and is recursive. Source: personal communication, I. Csiszár, May 8, 1993.

2.2.9. [12] A Turing machine T computes an infinite sequence ω if there is a program p such that $T(p, n) = \omega_{1:n}$ for all n. Define $C(\omega) = \min\{l(p) : U(p, n) = \omega_{1:n} \text{ for all } n\}$, or ∞ if such a p does not exist. Obviously, for all ω either $C(\omega) < \infty$ or $C(\omega) = \infty$.

(a) Show that $C(\omega) < \infty$ iff $0.\omega$ is a *recursive real number* as in Exercise 1.7.22 on page 47. For the mathematical constants π and e, $C(\pi) < \infty$ and $C(e) < \infty$.

(b) Show that the reals $0.\omega$ with $C(\omega) < \infty$ form a countably infinite set and that the reals $0.\omega$ with $C(\omega) = \infty$ have uniform measure one in the total set of reals in the interval $[0, 1)$.

2.2.10. [27] We consider how information about x can be dispersed. Let $x \in \mathcal{N}$ with $l(x) = n$ and $C(x) = n + O(1)$. Show that there are $u, v, w \in \mathcal{N}$ such that

(i) $l(u) = l(v) = l(w) = \frac{1}{2}n$, $C(u) = C(v) = C(w) = \frac{1}{2}n$ $(+O(1))$, and they are pairwise independent: $C(y|z) = \frac{1}{2}n + O(1)$ for $y, z \in \{u, v, w\}$ and $y \neq z$;

(ii) x can be reconstructed from any two of them: $C(x|y, z) = O(1)$, where $y, z \in \{u, v, w\}$ and $y \neq z$.

Can you give a general solution for finding $m+k$ elements of $\mathcal{N}$ such that each of them has length and complexity n/m, and x can be reconstructed from any m distinct elements?

Comments. It is surprising that x can be reconstructed from any two out of three elements, each of one-half the complexity of x. This shows that the identity of the individual bits is not preserved in the division. Hint: assume $n = 2m$ and $x = x_1 \dots x_{2m}$, $u = u_1 \dots u_m$, $v = v_1 \dots v_m$, and $w = w_1 \dots w_m$ with $u_i = x_{2i-1}$, $v_i = x_{2i}$, and $w_i = v_i \oplus u_i$. (Recall that $a \oplus b = 1$ iff $a \neq b$.) This solution apparently does not generalize. A general solution to distribute x over $m+k$ elements such that any group of m elements determines x can be given as follows: Compute the least integer $y \geq x^{1/m}$. Let p_i be the ith prime, with $p_1 = 2$. Distribute x over $u_1, \dots, u_{m+k}$, where $u_i \equiv x \bmod p_i^{\alpha(i)}$, with $\alpha(i) = \lceil y \log_{p_i} 2 \rceil$. Using the Chinese remainder theorem we find that we can reconstruct x from any subset of m elements u_i. Source: A. Shamir, *Comm. ACM*, 22:11(1979), 612–613; M.O. Rabin, *J. ACM*, 36:2(1989), 335–348.

2.2.11. [26] Show that there are strings x, y, z such that $C(x|y) + C(x|z) > C(x) + C(x|y, z) + O(1)$. For convenience prove this first for strings of the same length n; but it also holds for some strings x, y, z with $l(x) = \log n$ and $l(y) = l(z) = n$. *Comments.* This is a counterintuitive result. Hint: prove there are pairwise random strings x, y, z such that each string results from $\oplus$-ing the other two.

2.2.12. [18] Let A be the set of binary strings of length n. An element x in A is *δ-random* if $\delta(x|A) \leq \delta$, where $\delta(x|A) = n - C(x|A)$ is the randomness deficiency. Show that if $x \in B \subseteq A$, then

$$\log \frac{d(A)}{d(B)} - C(B|A) \leq \delta(x|A) + O(\log n).$$

Comments. That is, no random elements of A can belong to any subset B of A that is simultaneously pure (which means that $C(B|A)$ is small)

and not large (which means that $d(A)/d(B)$ is large). Source: A.N. Kolmogorov and V.A. Uspensky, *Theory Probab. Appl.*, 32(1987), 389–412.

2.2.13. [27] Let $x \in A$, with $d(A) < \infty$. Then in Section 2.2 the *randomness deficiency* of x *relative* to A is defined as $\delta(x|A) = l(d(A)) - C(x|A)$. (Here $C(x|A)$ is defined as $C(x|\chi)$ with χ the characteristic sequence of A and $l(\chi) < \infty$.) If $\delta(x|A)$ is large, this means that there is a description of x with the help of A that is considerably shorter than just giving x's serial number in A. Clearly, the randomness deficiency of x with respect to sets A and B can be vastly different. But then it is natural to ask whether there exist *absolutely nonrandom* objects, objects having large randomness deficiency with respect to any appropriate set.

Prove the following: Let a and b be arbitrary constants; for every sufficiently large n, there exists a binary string x of length n such that $\delta(x|A) \geq b \log n$ for any set A containing x for which $C(A) \leq a \log n$.

Comments. Source: A.K. Shen, *Soviet Math. Dokl.,* 28(1983), 295–299. Compare with Kamae's theorem, Exercise 2.7.5. Let us give some interpretation of such results bearing on statistical inference. Given an experimental result, the statistician wants to infer a statistical hypothesis under which the result is *typical*. Mathematically, given x we want to find a simple set A that contains x as a typical element. The above shows that there are outcomes x such that *no* simple statistical model of the kind described is possible. The question remains whether such objects occur in the real world.

2.2.14. [31] Consider two complexity measures for infinite binary sequences ω. Let $C_\infty(\omega)$ be the minimal length of a program p such that $p(n) = \omega_{1:n}$ for all sufficiently large n. Let $\hat{C}_\infty(\omega)$ be defined as $\limsup_{n\to\infty} C(\omega_{1:n}|n)$. Prove that $C_\infty(\omega) \leq 2\hat{C}_\infty(\omega) + O(1)$, and that this bound is tight (the constant 2 cannot be replaced by a smaller one).

Comments. Source: B. Durand, A.K. Shen, N.K. Vereshchagin. *Theoret. Comput. Sci.*, 171(2001), 47–58.

2.2.15. [37] Consider $C_{\lim}(x) = \min\{l(p) : p(n) = x \text{ for all but finitely many } n\}$ and $C_{\limsup}(x) = \min\{m : \text{for all but finitely many } n \text{ there exists a } p \text{ with } l(p) \leq m \text{ and } p(n) = x\}$. Let $C'(x)$ denote the plain Kolmogorov complexity relativized to $0'$ (that is, the program is allowed to ask an oracle whether a given Turing machine terminates on given input).

(a) Prove that $C_{\lim}(x) = C'(x) + O(1)$.

(b) Prove that $C_{\limsup}(x) = C'(x) + O(1)$.

Comments. Source: N.K. Vereshchagin *Theoret. Comput. Sci.*, 271(2002), 59–67. Item (b) is the more difficult one; Item (a) is attributed to An.A. Muchnik, S.Y. Positselsky.

2.3 C as an Integer Function

We consider C as an integer function $C : \mathcal{N} \to \mathcal{N}$, and study its behavior, Figure 2.1. First we observe that Theorem 2.1.2 gives an *upper bound* on C: there exists a constant c such that for all x in $\mathcal{N}$ we have $C(x) \leq l(x) + c$, and by Theorem 2.2.1 this estimate is almost exact for the majority of x's. This is a computable monotonic increasing upper bound that grows to infinity. It is also the least such upper bound. It turns out that the greatest monotonic nondecreasing lower bound also grows to infinity but does so incomputably slowly.

Theorem 2.3.1 (i) *The function $C(x)$ is unbounded.*

(ii) *Define a function m by $m(x) = \min\{C(y) : y \geq x\}$. That is, m is the greatest monotonic increasing function bounding C from below. The function $m(x)$ is unbounded.*

(iii) *For any partial recursive function $\phi(x)$ that goes monotonically to infinity from some x_0 onward, we have $m(x) < \phi(x)$ except for finitely many x. In other words, although $m(x)$ goes to infinity, it does so more slowly than any unbounded partial recursive function.*

Proof. (i) This follows immediately from (ii).

(ii) For each i there is a least x_i such that for all $x > x_i$, the smallest program p printing x has length greater than or equal to i. This follows immediately from the fact that there are only a finite number of programs of each length i. Clearly, for all i we have $x_{i+1} \geq x_i$. Now observe that the function m has the property that $m(x) = i$ for $x_i < x \leq x_{i+1}$.

(iii) Assume the contrary: there is a monotonic nondecreasing unbounded partial recursive function $\phi(x) \leq m(x)$ for infinitely many x. The domain $A = \{x : \phi(x) < \infty\}$ of ϕ is an infinite recursively enumerable set. By Lemma 1.7.4, A contains an infinite recursive subset B. Define

$$\psi(x) = \begin{cases} \phi(x) & \text{for } x \in B, \\ \phi(y) & \text{with } y = \max\{z : z \in B, z < x\}, \text{ otherwise.} \end{cases}$$

This ψ is total recursive and goes monotonically to infinity, and $\psi(x) \leq m(x)$ for infinitely many x.

Now define $M(a) = \max\{x : C(x) \leq a\}$. Then, $M(a) + 1 = \min\{x : m(x) > a\}$. It is easy to verify that

$$\max\{x : \psi(x) \leq a + 1\} \geq \min\{x : m(x) > a\} > M(a),$$

for infinitely many a's, and the function $F(a) = \max\{x : \psi(x) \leq a + 1\}$ is obviously total recursive. Therefore, $F(a) > M(a)$ for infinitely many a's. In other words, $C(F(a)) > a$ for infinitely many a's. But by Theorem 2.1.1,

$$C(F(a)) \leq C_F(F(a)) + O(1) \leq l(a) + O(1).$$

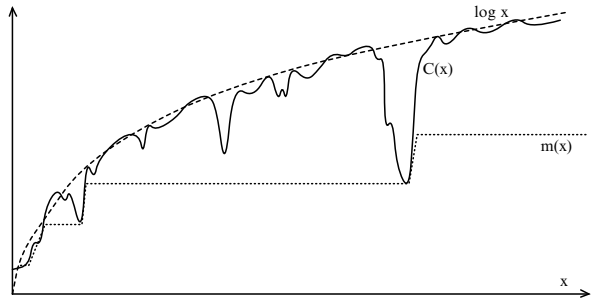


FIGURE 2.1. The graph of the integer function $C(x)$

This implies that there exists a constant c such that $l(a) + c \geq a$ for infinitely many a, which is impossible. □

Notice that Items (ii) and (iii) of Theorem 2.3.1 do not hold for the length-conditional complexity $C(x|l(x))$. Namely, although $C(x|l(x))$ is unbounded, it drops infinitely often to constant level. In other words, there is no unbounded monotonic function that is a lower bound on $C(x|l(x))$ by Example 2.2.5. This phenomenon is further explored in the exercises.

The second cornerstone of the theory (millstone around its neck is probably more apt) is the *incomputability theorem*.

Theorem 2.3.2 *The function $C(x)$ is not recursive. Moreover, no partial recursive function $\phi(x)$ defined on an infinite set of points can coincide with $C(x)$ over the whole of its domain of definition.*

Proof. This proof is related to that of Theorem 2.3.1, Item (iii). We prove that there is no partial recursive ϕ as in the statement of the theorem. Every infinite recursively enumerable set contains an infinite recursive subset, Lemma 1.7.4. Select an infinite recursive subset A in the domain of definition of ϕ. The function $\psi(m) = \min\{x : C(x) \geq m, x \in A\}$ is (total) recursive (since $C(x) = \phi(x)$ on A), and takes arbitrarily large values, Theorem 2.3.1. Also, by definition of ψ, we have $C(\psi(m)) \geq m$. On the other hand, $C(\psi(m)) \leq C_\psi(\psi(m)) + c_\psi$ by definition of C, and obviously $C_\psi(\psi(m)) \leq l(m)$. Hence, $m \leq \log m$ up to a constant independent of m, which is false from some m onward. □

That was the bad news; the good news is that we can approximate $C(x)$.

Theorem 2.3.3 *There is a total recursive function $\phi(t, x)$, monotonic decreasing in t, such that $\lim_{t\to\infty} \phi(t, x) = C(x)$.*

Proof. We define $\phi(t,x)$ as follows: For each x, we know that the shortest program for x has length at most $l(x)+c$, Theorem 2.1.2. Run the reference Turing machine U in the proof of Theorem 2.1.1 for t steps on *each* program p of length at most $l(x)+c$. If for any such input p the computation halts with output x, then define the value of $\phi(t,x)$ as the length of the shortest such p, otherwise equal to $l(x)+c$. Clearly, $\phi(t,x)$ is recursive, total, and monotonically nonincreasing with t (for all x, $\phi(t',x) \leq \phi(t,x)$ if $t' > t$). The limit exists, since for each x there exists a t such that U halts with output x after computing t steps starting with input p with $l(p) = C(x)$. □

One cannot decide, given x and t, whether $\phi(t,x) = C(x)$. Since $\phi(t,x)$ is nondecreasing and goes to the limit $C(x)$ for $t \to \infty$, if there were a decision procedure to test $\phi(t,x) = C(x)$, given x and t, then we could compute $C(x)$. But Theorem 2.3.2 tells us that C is not recursive.

Let $g_1, g_2, \ldots$ be a sequence of functions. We call f the *limit* of this sequence if $f(x) = \lim_{t\to\infty} g_t(x)$ for all x. The limit is *recursively uniform* if for every rational $\epsilon > 0$ there exists a $t(\epsilon)$, where t is a total recursive function, such that $|f(x) - g_{t(\epsilon)}(x)| \leq \epsilon$, for all x. Let the sequence of one-argument functions $\psi_1, \psi_2, \ldots$ be defined by $\psi_t(x) = \phi(t,x)$, for each t for all x. Clearly, C is the limit of the sequence of ψ's. However, by Theorem 2.3.2, the limit is not recursively uniform. In fact, by the halting problem in Section 1.7, for each $\epsilon > 0$ and $t > 0$ there exist infinitely many x such that $|C(x) - \psi_t(x)| > \epsilon$. This means that for each $\epsilon > 0$, for each t there are many x's such that our estimate $\phi(t,x)$ overestimates $C(x)$ by an *error* of at least ϵ.

We describe some other characteristics of the function C.

Continuity: The function C is continuous in the sense that there is a constant c such that $|C(x) - C(x \pm h)| \leq 2l(h) + c$ for all x and h. (Hint: given a program that computes x we can change it into another program that adds (or subtracts) h from the output.)

Logarithmic: The function $C(x)$ mostly hugs $\log x$. It is bounded above by $\log x + c$, Theorem 2.1.2, page 108. On the other hand, by Theorem 2.2.1, page 117, for each constant k, the number of x of length n (about $\log x$) such that $C(x) < \log x - k$ is at most 2^{n-k}.

Fluctuation: The function $C(x)$ fluctuates rapidly. Namely, for each x there exist two integers x_1, x_2 within distance $\sqrt{x}$ of x (that is, $|x - x_i| \leq \sqrt{x}$ for $i = 1, 2$) such that $C(x_1) \geq l(x)/2 - c$ and $C(x_2) \leq l(x)/2 + c$. (Hint: change the low-order half of the bits of x to some incompressible string to obtain x_1, and change these bits to a very compressible string (such as all zeros) to obtain x_2.) Therefore, if x is incompressible with $C(x) = l(x) - O(1)$, then there is an x_2 nearby where $C(x_2)$ equals about $C(x)/2$, and if x is compressible with $C(x) = o(l(x))$, then there is an x_1 nearby where

$C(x_1)$ equals about $l(x)/2$. These facts imply many fluctuations within small intervals because, for instance, $C(x)$, $C(x + \log x)$, $C(x+\sqrt{x})$, $C(cx)$, $C(x^2)$, and $C(2^x)$ all have about the same value.

Long high-complexity runs: For each c there is a d such that there are no runs of d consecutive c-incompressible numbers. However, conversely, for each d there is a c such that there are runs of d consecutive c-incompressible numbers. (Hint: for the nonexistence part use numbers x of the form $i2^j$ for which $C(i2^j) \leq l(i) + l(j) + c < l(i2^j) - d$; for the existence part use the continuity property and the nearly logarithmic property above.)

Example 2.3.1 It is not difficult to see that Theorems 2.3.1, Item (i), 2.3.2, and 2.3.3, Theorem 2.1.2, and the above properties hold for the length-conditional complexity measure $C(x|l(x))$. By the existence of n-strings, Example 2.2.5, the greatest monotonic lower bound on $C(x|l(x))$ is a fixed constant, and therefore Items (ii) and (iii) of Theorem 2.3.1 do not hold for this complexity measure. Theorems 2.1.1, 2.2.1 are already proved for $C(x|l(x))$ in their original versions. Namely, either they were proved for the conditional complexity in general, or the proof goes through as given for the length-conditional complexity. Thus, the general contour of the graph of $C(x|l(x))$ looks *very roughly* similar to that of $C(x)$, except that there are dips below a fixed constant infinitely often, Figure 2.2.

Let us make an estimate of how often the dips occur. Consider the n-strings of Example 2.2.5. For each integer n there is an extension of the corresponding binary string with $n - l(n)$ many 0's such that the resulting string x has complexity $C(x|l(x)) \leq c$ for a fixed constant c. It is easy to see that $\log x \approx n$, and that for all $n' < n$ the corresponding x' is less than x. Hence, the number of $x' < x$ such that $C(x'|l(x')) \leq c$ is at least $\log x$. ◇

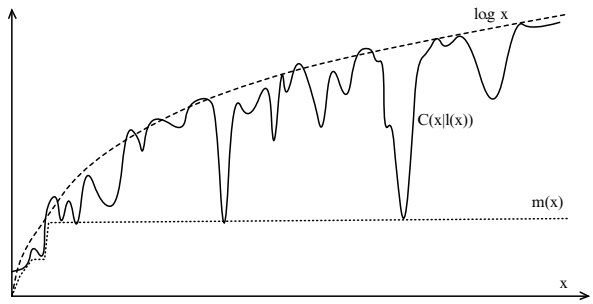


FIGURE 2.2. The graph of the integer function $C(x|l(x))$

Exercises

2.3.1. [15] Let $\phi(t,x)$ be a recursive function and $\lim_{t\to\infty}\phi(t,x) = C(x)$, for all x. For each t define $\psi_t(x) = \phi(t,x)$ for all x. Then C is the *limit* of the sequence of functions $\psi_1, \psi_2, \ldots$. Show that for each *error* ϵ and all t there are infinitely many x such that $\psi_t(x) - C(x) > \epsilon$.

Comments. $C(x)$ is the *uniform limit* of the approximation above if for each $\epsilon > 0$, there exists a t such that for all x, $\psi_t(x) - C(x) \leq \epsilon$. Item (a) implies that $C(x)$ is not the uniform limit of the above sequence of functions.

2.3.2. [23] Let $\phi_1, \phi_2, \ldots$ be the effective enumeration of partial recursive functions in Section 1.7. Define the *uniform complexity* of a finite string x of length n with respect to ϕ (occurring in the above enumeration) as $C_\phi(x;n) = \min\{l(p) : \phi(m,p) = x_{1:m} \text{ for all } m \leq n\}$ if such a p exists, and ∞ otherwise. We can prove an invariance theorem to the effect that there exists a universal partial recursive function ϕ_0 such that for all ϕ there is a constant c such that for all x, n we have $C_{\phi_0}(x;n) \leq C_\phi(x;n) + c$. We choose a reference universal function ϕ_0 and define the *uniform Kolmogorov complexity* as $C(x;n) = C_{\phi_0}(x;n)$.

(a) Show that for all finite binary strings x we have $C(x) \geq C(x;l(x)) \geq C(x|l(x))$ up to additive constants independent of x.

(b) Prove Theorems 2.1.1 to 2.3.3, with $C(x)$ replaced by $C(x;l(x))$.

(c) Show that in contrast to the measure $C(x|l(x))$, no constant c exists such that $C(x;l(x)) \leq c$ for all n-strings (Example 2.2.5).

(d) Show that in contrast to $C(x|l(x))$, the uniform complexity is *monotonic in the prefixes*: if $m \leq n$, then $C(x_{1:m};m) \leq C(x_{1:n};n)$, for all x.

(e) Show that there exists an infinite binary sequence ω and a constant c such that for infinitely many n, $C(\omega_{1:n};n) - C(\omega_{1:n}|n) > \log n - c$.

Comments. Item (b) shows that the uniform Kolmogorov complexity satisfies the major properties of the plain Kolmogorov complexity. Items (c) and (d) show that at least two of the objections to the length-conditional measure $C(x|l(x))$ do not hold for the uniform complexity $C(x;l(x))$. Hint for Item (c): this is implied by the proof of Theorem 2.3.1 and Item (a). Item (e) shows as strong a divergence between the measures concerned as one could possibly expect. Source: D.W. Loveland, *Inform. Contr.*, 15(1969), 510–526.

2.3.3. [27] Let BB' be a variant of the busy beaver function defined in Exercise 1.7.19, page 45, where $BB'(n)$ is defined as the maximal number of steps in a halting computation of the reference universal Turing machine when started on an n-bit input.

Show that $C(BB'(n)) = n + O(\log n)$. Use this to provide an alternative proof for Theorem 2.3.1, Item (iii).

Comments. Hint: Knowing n and the index $j \leq 2^n$ of the input that achieves $BB'(n)$, we can compute $BB'(n)$. Hence, $C(BB'(n) \mid n) \leq n + O(1)$. On the other hand, Knowing n and $BB'(n)$, we can run all programs of n bits for at most $BB'(n)$ steps; the programs that have not halted after $BB'(n)$ steps will never halt. This resolves the halting problem for all programs of n bits, and yields the halting sequence $\chi_1 \ldots \chi_{2^n}$ for the first 2^n programs. By an application of the later Theorem 2.7.2, known as Barzdins's lemma, Item (ii), we conclude that $C(BB'(n), n) \geq C(\chi_1 \ldots \chi_{2^n}) - O(1) \geq n - O(1)$.

2.3.4. • [35] Let ω be an infinite binary string. We call ω *recursive* if there exists a recursive function ϕ such that $\phi(i) = \omega_i$ for all $i > 0$. Prove the following:

(a) If ω is recursive, then there is a constant c such that for all n,

$$C(\omega_{1:n}; n) < c,$$
$$C(\omega_{1:n}|n) < c,$$
$$C(\omega_{1:n}) - C(n) < c.$$

This is easy. The converses also hold but are less easy to show. They follow from Items (b), (e), and (f).

(b) For each constant c, there are only finitely many ω such that for all n, $C(\omega_{1:n}; n) \leq c$, and each such ω is recursive.

(c) For each constant c, there are only finitely many ω such that for infinitely many n, $C(\omega_{1:n}; n) \leq c$, and each such ω is recursive.

(d) There exists a constant c such that the set of infinite ω that satisfy $C(\omega_{1:n}|n) \leq c$ for infinitely many n, has the cardinality of the continuum.

(e) For each constant c, there are only finitely many ω such that for all n, $C(\omega_{1:n}|n) \leq c$, and each such ω is recursive.

(f) For each constant c, there are only finitely many ω with $C(\omega_{1:n}) \leq C(n) + c$ for all n, and each such ω is recursive.

(g) For each constant c, there are only finitely many ω with $C(\omega_{1:n}) \leq l(n) + c$ for all n, and each such ω is recursive.

(h) There exist nonrecursive ω for which there exists a constant c such that $C(\omega_{1:n}) \leq C(n) + c$ for infinitely many n.

Comments. Clearly Item (c) implies Item (b). In Item (d) conclude that not all such ω are recursive. In particular, the analogue of Item (c) for $C(\omega_{1:n}|n)$ does not hold. Namely, there exist nonrecursive ω for which there exists a constant c such that for infinitely many n we have $C(\omega_{1:n}|n) \leq c$. Hint for Item (d): exhibit a one-to-one coding of subsets of $\mathcal{N}$ into the set of infinite binary strings of which infinitely many

prefixes are n-strings—in the sense of Example 2.2.5. Item (e) means that in contrast to the differences between the measures $C(\cdot; l(\cdot))$ and $C(\cdot|l(\cdot))$ exposed by the contrast between Items (c) and (d), Item (b) holds also for $C(\cdot|l(\cdot))$. Items (f) and (g) show a complexity gap, because $C(n)$ can be much lower than $l(n)$. Hint for Item (h): use Item (d). Source: for Items (b) through (e), and (h), D.W. Loveland, *Inform. Contr.*, 15(1969), 510–526. Loveland attributes Item (e) to A.R. Meyer. The equivalence between bounded length-conditional complexity and bounded uniform complexity for prefixes of infinite strings is stated by A.K. Zvonkin and L.A. Levin, *Russ. Math. Surv.*, 25:6(1970), 83–124. Source of Items (f) and (g) is G.J. Chaitin, *Theoret. Comput. Sci.*, 2(1976), 45–48. For the prefix complexity K introduced in Chapter 3, there are nonrecursive ω such that $K(\omega_{1:n}) \leq K(n) + O(1)$ for all n by a result of R.M. Solovay in Exercise 3.6.9 on page 231.

2.3.5. [HM35] We want to show in some precise sense that the real line is computationally a fractal. (Actually, one is probably most interested in Item (a), which can be proved easily and elementarily from the following definition.) The required framework is as follows: Each infinite binary sequence $\omega = \omega_1\omega_2\ldots$ corresponds to a real number $0 \leq 0.\omega < 1$. Define the normalized complexity $Cn(\omega) = \lim_{n\to\infty} C(\omega_{1:n})/n$. If the limit does not exist, we set $Cn(\omega)$ to half the sum of the upper and lower limits.

(a) Show that for all real ω in $[0,1)$, for every $\epsilon > 0$ and all real r, $0 \leq r \leq 1$, there exist real ζ in $[0,1)$ such that $|\omega - \zeta| < \epsilon$ and $Cn(\zeta) = r$. (For each real r, $0 \leq r \leq 1$, the set of ω's with $Cn(\omega) = r$ is dense on the real line $[0,1)$.)

(b) Show that for all real ω, all rational r and s, we have $Cn(r\omega + s) = Cn(\omega)$ (both ω and $r\omega + s$ in $[0,1)$). Similarly, show that $Cn(f(\omega)) = Cn(\omega)$ for all recursive functions f.

B. Mandelbrot defined a set to be a *fractal* if its Hausdorff dimension is greater than its topological dimension [B. Mandelbrot, *The Fractal Geometry of Nature*, W.H. Freeman, 1983; for definitions of the dimensions see W. Hurewicz and H. Wallman, *Dimension Theory*, Princeton Univ. Press, 1974].

(c) Show that for any real numbers $0 \leq a < b \leq 1$, the Hausdorff dimension of the set $\{(\omega, Cn(\omega))\} \bigcap ([0,1) \times [a,b])$ is $1 + b$.

(d) Show that the set $G = \{(\omega, Cn(\omega)) : \omega \in [0,1)\}$ has Hausdorff dimension 2 and topological dimension 1. (That is, G is a fractal.)

Comments. Source: J.-Y. Cai and J. Hartmanis, *J. Comput. System Sci.*, 49:3(1994), 605–619. Other relationships among the Hausdorff dimension, Lutz's constructive dimension, and Kolmogorov complexity have been investigated by L. Staiger in [*Inform. Comput.*, 102(1993), 159-194; *Theor. Comput. Syst.* 31(1998), 215-229], B.Ya. Ryabko in [*J.*

Complexity, 10(1994) 281–295]; J.H. Lutz in [*Proc. 27th Int. Colloq. Aut. Lang. Prog.*, 2000, pp. 902–913; *Inform. Comput.*, 187(2003), pp. 49-79; *SIAM J. Comput.* 32(2003), 1236-1259], and E. Mayordomo in [*Inform. Process. Lett.*, 84:1(2002), 247–356].

2.3.6. [M34] To investigate repeating patterns in the graph of $C(x)$ we define the notion of a 'shape match.' Every function from the integers to the integers is a shape. A *shape* f *matches* the graph of C at j with *span* c if for all x with $j-c \le x \le j+c$ we have $C(x) = C(j)+f(x-j)$.

(a) Show that every matching shape has $f(0) = 0$. Thus, a matching shape is a template of which we align the center $f(0)$ with j to see to what extent it matches C's graph around the point of interest. We wish to investigate shapes that can be made to match arbitrarily far in each direction.

(b) A shape f is *recurrent* if for all c there exists a j such that f matches the graph of C at j with span c. Show that there exists a recurrent shape.

(c) Show that there exists a constant c such that there are no runs $C(n) = C(n+1) = \cdots = C(n+c)$ for any n.

(d) Prove that no recurrent shape is a recursive function.

Comments. The notion of 'shape match' is different from that of 'following the shape' in Definition 5.5.8 on page 407. Hints: for Item (b) use König's infinity lemma. Item (c) means that the graph of C has no arbitrarily long flat spots. For Item (c), prove for sufficiently large c that for each integer i, for all n with $C(n) = i$, the run $C(n), C(n+1), \ldots, C(n+c)$ contains an element less than i. For Item (d) use a case analysis, and use in one case the proof of Item (c) and in the other cases the recursion theorem, Exercises 1.7.20, page 46. Source: H.P. Katseff and M. Sipser, *Theoret. Comput. Sci.*, 15(1981), 291–309.

2.4 Random Finite Sequences

One can consider those objects as nonrandom in which one can find sufficiently many regularities. In other words, we would like to identify incompressibility with randomness. This is proper if the sequences that are incompressible can be shown to possess the various properties of randomness (stochasticity) known from the theory of probability. That this is possible is the substance of the celebrated theory developed by the Swedish mathematician Per Martin-Löf.

There are many properties known that probability theory attributes to random objects. To give an example, consider sequences of n tosses with a fair coin. Each sequence of n zeros and ones is equiprobable as an outcome: its probability is 2^{-n}. If such a sequence is to be random in the sense of a proposed new definition, then the number of ones in x

should be near to $\frac{1}{2}n$, the number of occurrences of blocks 00 should be close to $\frac{1}{4}n$, and so on.

It is not difficult to show that each such single property separately holds for all incompressible binary strings. But we want to demonstrate that incompressibility implies all conceivable effectively testable properties of randomness (both the known ones and the as yet unknown ones). In this way, the various theorems in probability theory about random sequences carry over automatically to incompressible sequences.

In the case of finite strings we cannot hope to distinguish sharply between random and nonrandom strings. For instance, considering the set of binary strings of a fixed length, it would not be natural to fix an m and call a string with m zeros random and a string with $m+1$ zeros nonrandom.

Let us borrow some ideas from statistics. We are given a certain sample space S with an associated distribution P. Given an element x of the sample space, we want to test the hypothesis "x is a typical outcome." Practically speaking, the property of being typical is the property of belonging to any reasonable majority. In choosing an object at random, we have confidence that this object will fall precisely in the intersection of all such majorities. The latter condition we identify with x being random.

To ascertain whether a given element of the sample space belongs to a particular reasonable majority, we introduce the notion of a test. Generally, a test is given by a prescription that for every level of significance ϵ, tells us for what elements x of S the hypothesis "x belongs to majority M in S" should be rejected, where $\epsilon = 1 - P(M)$. Taking $\epsilon = 2^{-m}$, $m = 1, 2, \ldots$, we achieve this by saying that we have a description of the set $V \subseteq \mathcal{N} \times S$ of nested *critical regions*

$$V_m = \{x : (m, x) \in V\},$$
$$V_m \supseteq V_{m+1}, \quad m = 1, 2, \ldots,$$

while the condition that V_m be a critical region on the *significance level* $\epsilon = 2^{-m}$ amounts to requiring, for all n,

$$\sum_x \{P(x|l(x) = n) : x \in V_m\} \leq \epsilon.$$

The complement of a critical region V_m is called the $(1 - \epsilon)$ *confidence interval*. If $x \in V_m$, then the hypothesis "x belongs to majority M," and therefore the stronger hypothesis "x is random," is rejected with significance level ϵ. We can say that x fails the test at the level of critical region V_m.

Example 2.4.1 A string $x_1 x_2 \ldots x_n$ with many initial zeros is not very random. We can test this aspect as follows. The special test V has critical regions

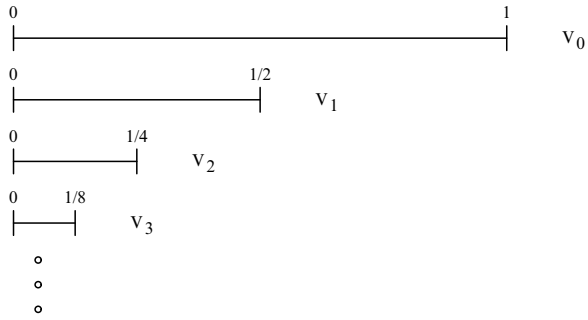


FIGURE 2.3. Test of Example 2.4.1

$V_1, V_2, \ldots$. Consider $x = 0.x_1x_2 \ldots x_n$ as a rational number, and each critical region as a half-open interval $V_m = [0, 2^{-m})$ in $[0, 1)$, $m = 1, 2, \ldots$. Then the subsequent critical regions test the hypothesis "x is random" by considering the subsequent digits in the binary expansion of x. We reject the hypothesis on the significance level $\epsilon = 2^{-m}$ provided $x_1 = x_2 = \cdots = x_m = 0$, Figure 2.3. ◇

Example 2.4.2 Another test for randomness of finite binary strings rejects when the relative frequency of ones differs too much from $\frac{1}{2}$. This particular test can be implemented by rejecting the hypothesis of randomness of $x = x_1x_2 \ldots x_n$ at level $\epsilon = 2^{-m}$ provided $|2f_n - n| > g(n, m)$, where $f_n = \sum_{i=1}^{n} x_i$, and $g(n, m)$ is the least number determined by the requirement that the number of binary strings x of length n for which this inequality holds be at most 2^{n-m}. Thus, in this case the critical region V_m is $\{x \in \{0, 1\}^n : |2f_n - n| > g(n, m)\}$. ◇

2.4.1 Randomness Tests

In practice, statistical tests are *effective* prescriptions such that we can compute, at each level of significance, for what strings the associated hypothesis should be rejected. It would be hard to imagine what use it would be in statistics to have tests that are not effective in the sense of computability theory (Section 1.7).

Definition 2.4.1 Let P be a recursive probability distribution on sample space $\mathcal{N}$. A total function $\delta : \mathcal{N} \to \mathcal{N}$ is a *P-test* (Martin-Löf test for randomness) if

1. δ is lower semicomputable (the set $V = \{(m, x) : \delta(x) \geq m\}$ is recursively enumerable); and
2. $\sum_x \{P(x|l(x) = n, \delta(x) \geq m\} \leq 2^{-m}$, for all n.

The critical regions associated with the common statistical tests are present in the form of the sequence $V_1 \supseteq V_2 \supseteq \cdots$, where $V_m = \{x : \delta(x) \geq m\}$, for $m \geq 1$. Nesting is ensured, since $\delta(x) \geq m+1$ implies $\delta(x) \geq m$. Each set V_m is recursively enumerable because of Item 1.

Consider the important case of the uniform distribution, defined by $L(x) = 2^{-2l(x)-1}$. The restriction of L to strings of length n is defined by $L_n(x) = 2^{-n}$ for $l(x) = n$ and 0 otherwise. (By definition, $L_n(x) = L(x|l(x) = n)$.) Then Item 2 can be rewritten as $\sum_{x \in V_m} L_n(x) \leq 2^{-m}$, which is the same as

$$d(\{x : l(x) = n,\ x \in V_m\}) \leq 2^{n-m}.$$

In this case we often speak simply of a *test*, with the uniform distribution L understood.

In Definition 2.4.1, the integer function δ is total and the set of points V of its graph is recursively enumerable. But the totality of δ implies that the recursively enumerable set V is actually recursive, and therefore we can require δ to be a total recursive function without changing the notion of P-test.

In statistical tests, membership of (m, x) in V can usually be determined in time polynomial in $l(m) + l(x)$.

Note that

$$\sum_x P(x)\delta(x) = \sum_m P\{x : \delta(x) \geq m\} \leq \sum_m 2^{-m} = 2.$$

Therefore, $\delta'(x) = \log \delta(x)$ is almost a sum-P test, Definition 4.3.8 on page 278.

Example 2.4.3 The previous test examples can be rephrased in terms of Martin-Löf tests. Let us try a more subtle example. A real number such that all bits in odd positions in its binary representation are 1's is not random with respect to the uniform distribution. To show this we need a test that detects sequences of the form $x = 1x_2 1x_4 1x_6 1x_8 \ldots$. Define a test δ by

$$\delta(x) = \max\{i : x_1 = x_3 = \cdots = x_{2i-1} = 1\},$$

and $\delta(x) = 0$ if $x_1 = 0$. For example: $\delta(01111) = 0$; $\delta(10011) = 1$; $\delta(11011) = 1$; $\delta(10100) = 2$; $\delta(11111) = 3$. To show that δ is a test we have to show that δ satisfies the definition of a test. Clearly, δ is lower semicomputable (even recursive). If $\delta(x) \geq m$ where $l(x) = n \geq 2m-1$, then there are 2^{m-1} possibilities for the $(2m-1)$-length prefix of x, and $2^{n-(2m-1)}$ possibilities for the remainder of x. Therefore, $d\{x : \delta(x) \geq m,\ l(x) = n\} \leq 2^{n-m}$. ◇

Definition 2.4.2 A universal Martin-Löf test for randomness with respect to a distribution P, a *universal P-test* for short, is a test $\delta_0(\cdot|P)$ such that for each P-test δ, there is a constant c such that for all x we have $\delta_0(x|P) \geq \delta(x) - c$.

We say that $\delta_0(\cdot|P)$ (additively) majorizes δ. Intuitively, $\delta_0(\cdot|P)$ constitutes a test for randomness that incorporates all particular tests δ in a single test. No test for randomness δ other than $\delta_0(\cdot|P)$ can discover more than a constant amount of greater deficiency of randomness in any string x. In terms of critical regions, a universal test is a test such that if a binary sequence is random with respect to that test, then it is random with respect to any conceivable test, neglecting a change in significance level. Namely, with $\delta_0(\cdot|P)$ a universal P-test, let $U = \{(m, x) : \delta_0(x|P) \geq m\}$, and for any test δ, let $V = \{(m, x) : \delta(x) \geq m\}$. Then, defining the associated critical zones as before, we obtain

$$V_{m+c} \subseteq U_m, \ m = 1, 2, \ldots,$$

where c is a constant (dependent only on U and V).

It is a major result that there exists a universal P-test. The proof goes by first showing that the set of all tests is enumerable. This involves the first example of a type of construction we shall use over and over again in different contexts in Chapters 2, 3, and 4. The idea is as follows:

Lemma 2.4.1 *We can effectively enumerate all P-tests.*

Proof. We start with the standard enumeration $\phi_1, \phi_2, \ldots$ of partial recursive functions from $\mathcal{N}$ into $\mathcal{N} \times \mathcal{N}$, and turn this into an enumeration $\delta_1, \delta_2, \ldots$ of all and only P-tests. The list $\phi_1, \phi_2, \ldots$ enumerates all and only recursively enumerable sets of pairs of integers as $\{\phi_i(x) : x \geq 1\}$ for $i = 1, 2, \ldots$. In particular, for any P-test δ, the set $\{(m, x) : \delta(x) \geq m\}$ occurs in this list. The *only* thing we have to do is to eliminate those ϕ_i whose range does not correspond to a P-test.

First, we effectively modify each ϕ (we drop the subscript for convenience) to a function ψ such that range ϕ equals range ψ, and ψ has the special property that if $\psi(n)$ is defined, then $\psi(1), \psi(2), \ldots, \psi(n-1)$ are also defined. This can be done by dovetailing the computations of ϕ on the different arguments: in the first phase, do one step of the computation of $\phi(1)$; in the second phase, do the second step of the computation of $\phi(1)$ and the first step of the computation of $\phi(2)$. In general, in the nth phase we execute the n_1th step of the computation of $\phi(n_2)$, for all n_1, n_2 satisfying $n_1 + n_2 = n$. We now define ψ as follows. If the first computation that halts is that of $\phi(i)$, then set $\psi(1) := \phi(i)$. If the second computation that halts is that of $\phi(j)$, then set $\psi(2) := \phi(j)$, and so on.

Secondly, use each ψ to construct a test δ by approximation from below. In the algorithm, at each stage of the computation the local variable array $\delta(1 : \infty)$ contains the current approximation to the list of function values $\delta(1), \delta(2), \ldots$. This is doable because the nonzero part of the approximation is always finite.

Step 1. Initialize δ by setting $\delta(x) := 0$ for all x; and set $i := 0$. **{**If the range of ψ is empty, then this assignment will not be changed in the remainder of the procedure, that is, δ stays identically zero and it is trivially a test**}**

Step 2. Set $i := i + 1$; compute $\psi(i)$ and let its value be (m, x).

Step 3. If $\delta(x) \geq m$ **then go to** Step 2 **else** set $\delta(x) := m$.

Step 4. If $\sum\{P(y|l(y) = l(x)) : \delta(y) \geq k\} > 2^{-k}$ for some k, $k = 1, \ldots, m$ **{**since P is a recursive function we can effectively test whether the new value of $\delta(x)$ violates Definition 2.4.1 on page 135**}** **then** set $\delta(x) := 0$ and terminate **{**the computation of δ is finished**}** **else go to** Step 2.

With P the uniform distribution, for $i = 1$ the conditional in Step 4 simplifies to $m > l(x)$. In case the range of ψ is already a test, then the algorithm never finishes but forever approximates δ from below. If ψ diverges for some argument then the computation goes on forever and does not change δ any more. The resulting δ is a lower semicomputable test. If the range of ψ is not a test, then at some point the conditional in Step 4 is violated and the approximation of δ terminates. The resulting δ is a test, even a recursive one. Executing this procedure on all functions in the list $\phi_1, \phi_2, \ldots$, we obtain an effective enumeration $\delta_1, \delta_2, \ldots$ of all P-tests (and only P-tests). We are now in a position to define a universal P-test. □

Theorem 2.4.1 *Let $\delta_1, \delta_2, \ldots$ be an enumeration of the above P-tests. Then $\delta_0(x|P) = \max\{\delta_y(x) - y : y \geq 1\}$ is a universal P-test.*

Proof. Note first that $\delta_0(\cdot|P)$ is a total function on $\mathcal{N}$ because of Item 2 in Definition 2.4.1, page 135.

(1) The enumeration $\delta_1, \delta_2, \ldots$ in Lemma 2.4.1 yields an enumeration of recursively enumerable sets:

$$\{(m, x) : \delta_1(x) \geq m\},\ \{(m, x) : \delta_2(x) \geq m\}, \ldots\ .$$

Therefore, $V = \{(m, x) : \delta_0(x|P) \geq m\}$ is recursively enumerable.

(2) Let us verify that the critical regions are small enough: for each n,

$$\begin{aligned}
&\sum_{l(x)=n} \{P(x|l(x) = n) : \delta_0(x|P) \geq m\} \\
&\qquad \leq \sum_{y=1}^{\infty} \sum_{l(x)=n} \{P(x|l(x) = n) : \delta_y(x) \geq m + y\} \\
&\qquad \leq \sum_{y=1}^{\infty} 2^{-m-y} = 2^{-m}.
\end{aligned}$$

(3) By its definition, $\delta_0(\cdot|P)$ majorizes each δ additively. Hence, it is universal. □

By definition of $\delta_0(\cdot|P)$ as a universal P-test, any particular P-test δ can discover at most a constant amount more regularity in a sequence x than does $\delta_0(\cdot|P)$, in the sense that for each δ_y we have $\delta_y(x) \leq \delta_0(x|P) + y$ for all x.

For any two universal P-tests $\delta_0(\cdot|P)$ and ${\delta'}_0(\cdot|P)$, there is a constant $c \geq 0$ such that for all x we have $|\delta_0(x|P) - {\delta'}_0(x|P)| \leq c$.

2.4.2 Explicit Universal Randomness Test

We started out with the objective to establish in what sense incompressible strings may be called random. In Section 2.2.1 we considered the randomness deficiency $\delta(x|A)$ of a string x relative to a finite set A. With A the set of strings of length n and $x \in A$ we find that $\delta(x|A) = \delta(x|n) = n - C(x|n)$.

Theorem 2.4.2 *The function $\delta_0(x|L) = l(x) - C(x|l(x)) - 1$ is a universal L-test with L the uniform distribution.*

Proof. (1) We first show that $f(x) = \delta_0(x|L)$ is a test with respect to the uniform distribution. The set $\{(m, x) : f(x) \geq m\}$ is recursively enumerable by Theorem 2.3.3.

(2) We verify the condition on the critical regions. Since the number of x's with $C(x|l(x)) \leq l(x) - m - 1$ cannot exceed the number of programs of length at most $l(x) - m - 1$, we have $d(\{x : f(x) \geq m\}) \leq 2^{l(x)-m} - 1$.

(3) We show that for each test δ, there is a constant c such that $f(x) \geq \delta(x) - c$. The main idea is to bound $C(x|l(x))$ by exhibiting a description of x, given $l(x)$. Fix x. Let the set A be defined as

$$A = \{z : \delta(z) \geq \delta(x), l(z) = l(x)\}.$$

We have defined A such that $x \in A$ and $d(A) \leq 2^{l(x)-\delta(x)}$. Let $\delta = \delta_y$ in the standard enumeration $\delta_1, \delta_2, \ldots$ of tests. Given y, $l(x)$, and $\delta(x)$, we have an algorithm to enumerate all elements of A. Together with the index j of x in enumeration order of A, this suffices to find x. We pad the standard binary representation of j with nonsignificant zeros to a string $s = 00\ldots0j$ of length $l(x) - \delta(x)$. This is possible since $l(s) \geq l(d(A))$. The purpose of changing j to s is that now the number $\delta(x)$ can be deduced from $l(s)$ and $l(x)$. In particular, there is a Turing machine that computes x from input $\bar{y}s$, when $l(x)$ is given for free. Consequently, by Theorem 2.1.1, $C(x|l(x)) \leq l(x) - \delta(x) + 2l(y) + 1$. Since y is a constant depending only on δ, we can set $c = 2l(y) + 2$. □

Definition 2.4.3 Let us fix $\delta_0(x|L) = l(x) - C(x|l(x)) - 1$ as the *reference universal test* with respect to the *uniform distribution* L. A string x is called *c-random* if $\delta_0(x|L) \leq c$.

Randomness of a string is related to its incompressibility. It is easy to see that

$$C(x|l(x)) \leq C(x) \leq C(x|l(x)) + 2C(l(x) - C(x|l(x))),$$

up to fixed additive constants. (We can reconstruct $l(x)$ from the length of a shortest program p for x and the quantity $l(x) - l(p)$.) This makes $C(x)$ and $C(x|l(x))$ about equal for the special case of x being incompressible. (However, for compressible x, such as $x = 0^n$, the difference between $C(x)$ and $C(x|n)$ can rise to logarithmic in n.) Together with Theorem 2.4.2, this provides the relation between the outcome of the reference universal L-test and incompressibility. Fix a constant c. If string x is c-incompressible, then $\delta_0(x|L) \leq c'$, where c' is a constant depending on c but not on x. Similarly, if $\delta_0(x|L) \leq c$, then x is c'-incompressible, where c' is a constant depending on c but not on x. Roughly, x is *random*, or *incompressible*, if $l(x) - C(x|l(x))$ is small with respect to $l(x)$.

Example 2.4.4 It is possible to directly demonstrate a property of random binary strings $x = x_1x_2 \ldots x_n$ related to Example 2.4.2: the number of ones, $f_n = x_1 + \cdots + x_n$, must satisfy $|2f_n - n| = O(\sqrt{n})$. Assume that x is a binary string of length n that is random in the sense of Martin-Löf. Then, by Theorem 2.4.2, x is also c-incompressible for some fixed constant c. Let $f_n = k$. The number of strings satisfying this equality is $\binom{n}{k}$. By simply giving the index of x in the lexicographic order of such strings, together with n, and the excess of ones, $d = |2k - n|$, we can give a description of x. Therefore, using a short self-delimiting program for d, we have

$$C(x|n) \leq \log \binom{n}{k} + C(d) + 2l(C(d)).$$

For x given n to be c-incompressible for a constant c, we need $C(x|n) \geq n - c$. Then,

$$n - \log \binom{n}{k} - C(d) - 2l(C(d)) \leq c,$$

which can be satisfied only if $d = O(\sqrt{n})$ (estimate the binomial coefficient by Stirling's formula, Exercise 1.5.4 on page 17). Curiously, if d given n is easily describable (for example $d = 0$ or $d = \sqrt{n}$), then x given n is not random, since it is not c-incompressible. ◇

Randomness in the sense of Martin-Löf means randomness insofar as it can be effectively certified. In other words, it is a negative definition. We look at

objects through a special filter, which highlights some features but obscures others. We can perceive some qualities of nonrandomness through the limited sight of effective tests. Everything else we then call by definition random. This is a matter of a pragmatic, expedient, approach. It has no bearing on the deeper question about what properties real randomness, physically or mathematically, should have. It just tells us that an object is random as far as we will ever be able to tell, *in principle* and not just as a matter of *practicality*.

Exercises

2.4.1. [20] For a binary string x of length n, let $f(x)$ be the number of ones in x. Show that $\delta(x) = \log(2n^{-1/2}|f(x) - \frac{1}{2}n|)$ is a P-test with P the uniform measure.

Comments. Use Markov's inequality to derive that for each positive λ, the probability of $2n^{-1/2}|f(x) - \frac{1}{2}n| > \lambda$ is at most $1/\lambda$. Source: T.M. Cover, P. Gács, and R.M. Gray, *Ann. Probab.*, 17(1989), 840–865.

2.4.2. [23] Let $x_1x_2 \ldots x_n$ be a random sequence with $C(x|n) \geq n$.

(a) Use a Martin-Löf test to show that $x_10x_20 \ldots 0x_n$ is not random with respect to the uniform distribution.

(b) Use a Martin-Löf test to show that the ternary sequence $y_1y_2 \ldots y_n$ with $y_1 = x_n + x_1$ and $y_i = x_{i-1} + x_i$ for $1 < i \leq n$ is not random with respect to the uniform distribution.

Comments. Hint: in Item (b) in the y-string the blocks 02 and 20 do not occur. Extend the definition of random sequences from binary to ternary. Source: R. von Mises, *Probability, Statistics and Truth*, Dover, 1981.

2.4.3. [35] Let x be a finite binary sequence of length n with $f_j = x_1 + x_2 + \cdots + x_j$ for $1 \leq j \leq n$. Show that there exists a constant $c > 0$ such that for all $m \in \mathcal{N}$, all $\epsilon > 0$, and all x,

$$C(x|n, f_n) > \log \binom{n}{f_n} - \log(m\epsilon^4) + c$$

implies

$$\max_{m \leq j \leq n} \left| \frac{f_j}{j} - \frac{f_n}{n} \right| < \epsilon.$$

Comments. This result is called *Fine's theorem*. This is an instance of the general principle that high probability of a computable property translates into the fact that high complexity implies that property. (For infinite sequences this principle is put in a precise and rigorous form in Theorem 2.5.5.) Fine's theorem shows that for finite binary sequences with high Kolmogorov complexity, given the length and the number of

ones, the fluctuations of the relative frequencies in the initial segments is small. Since we deal with finite sequences, this is called *apparent convergence.* Since virtually all finite binary strings have high complexity (Theorem 2.1.3), this explains why in a typical sequence produced by random coin throws the relative frequencies appear to converge or stabilize. Apparent convergence occurs because of, and not in spite of, the high irregularity (randomness or complexity) of a data sequence. Conversely, the failure of convergence forces the complexity to be less than maximal. Source: T.L. Fine, *IEEE Trans. Inform. Theory,* IT-16(1970), 251–257; also R. Heim, *IEEE Trans. Inform. Theory,* IT-25(1979), 557–566.

2.4.4. [36] (a) Consider a finite sequence of zeros and ones generated by independent tosses of a coin with probability p $(0 < p < 1)$ for 1. Let $x = x_1 x_2 \dots x_n$ be a sequence of outcomes of length n, and let $f_n = x_1 + x_2 + \cdots + x_n$. The probability of such an x is $p^{f_n}(1-p)^{n-f_n}$. If p is a recursive number, then the methods in this section can be used to obtain a proper definition of finite *Bernoulli sequences*, sequences that are random for this distribution. There is, however, no reason to suppose that in physical coins p is a recursive real. This prompts another approach whereby we disregard the actual probability distribution, but follow more closely the combinatorial spirit of Kolmogorov complexity. Define a finite Bernoulli sequence as a binary sequence x of length n whose only regularities are given by f_n and n. That is, x is a Bernoulli sequence iff $C(x|n, f_n) = \log \binom{n}{f_n}$ up to a constant independent of x. Define a *Bernoulli test* as a test with the condition that the number of sequences with f_n ones and $n - f_n$ zeros in V_m be $\leq 2^{-m}\binom{n}{f_n}$ for all m, n, and f_n. Show that there exists a universal Bernoulli test δ_0. Finite Bernoulli sequences are those sequences x such that $\delta_0(x)$ is low. Show that up to a constant independent of x,

$$\delta_0(x) = \log \binom{n}{f_n} - C(x|n, f_n),$$

for all x (with n and f_n as above).

(b) We continue Item (a). In the current interpretation of probability, not only should the relative frequency of an event in a large number of experiments be close to the probability, but there is an obscure secondary stipulation. If the probability of success is very small, we should be practically sure that the event should not occur in a single trial [A.N. Kolmogorov, *Grundbegriffe der Wahrscheinlichkeitsrechnung*, Berlin, 1933; English translation: Chelsea, 1956]. If x is a Bernoulli sequence (result of n independent coin tosses) with a very low relative success frequency f_n/n (the coin is heavily biased), then, almost necessarily, $x_1 = 0$. That is, the assumption that 1 occurs as the very first element implies substantial regularity of the overall sequence.

Show that there is a constant c such that $\delta_0(x) \leq \log n/f_n - c$ implies $x_1 = 0$.

Comments. Hint for Item (b): construct the test that rejects at level 2^{-m} when $x_1 = 1$ and $f_n \leq n2^{-m}$. Show that this is a Bernoulli test. Compare this test with δ_0 in Item (a). For sequential Bernoulli tests for infinite sequences see Exercise 2.5.17. Source: P. Martin-Löf, *Inform. Contr.*, 9(1966), 602–619.

2.5 *Random Infinite Sequences

Consider the question of how C behaves in terms of increasingly long initial segments of a fixed infinite binary sequence (or string) ω. Is it monotone in the sense that $C(\omega_{1:m}) \leq C(\omega_{1:n})$, or $C(\omega_{1:m}|m) \leq C(\omega_{1:n}|n)$, for all infinite binary sequences ω and all $m \leq n$? We have already seen that the answer is negative in both cases. A similar effect arises when we try to use Kolmogorov complexity to solve the problem of finding a proper definition of random infinite sequences (*collectives*) according to the task already set by von Mises in 1919, Section 1.9.

2.5.1 Complexity Oscillations

It is seductive to call an infinite binary sequence ω random if there is a constant c such that for all n, the n-length prefix $\omega_{1:n}$ has $C(\omega_{1:n}) \geq n - c$. However, such sequences do not exist. We shall show that for high-complexity sequences, with $C(\omega_{1:n}) \geq n - \log n - 2\log\log n$ for all n, this results in so-called *complexity oscillations*, where for every $\epsilon > 0$,

$$\frac{n - C(\omega_{1:n})}{\log n}$$

oscillates between 0 and $1+\epsilon$ for large enough n. First, we show that the C complexity of prefixes of each infinite binary sequence drops infinitely often unboundedly far below its own length.

Theorem 2.5.1 *Let $f : \mathcal{N}^+ \to \mathcal{N}$ be a total recursive function satisfying $\sum_{n=1}^{\infty} 2^{-f(n)} = \infty$ (such as $f(n) = \log n$). Then for all infinite binary sequences ω, we have $C(\omega_{1:n}|n) \leq n - f(n)$ infinitely often.*

Proof. In order to get rid of an $O(1)$ term in the final argument, we first change f into something that gets arbitrarily larger yet still diverges in the same way as f. Define

$$F(n) = \left\lfloor \log \left(\sum_{i=1}^{n} 2^{-f(i)} \right) \right\rfloor .$$

Note that $\sum_{F(n)=m} 2^{-f(n)} \geq 2^m - 1$. Now let g, also total recursive, be defined as $g(n) = f(n) + F(n)$. It follows that

$$\begin{aligned}\sum_{n=1}^{\infty} 2^{-g(n)} &= \sum_{m=1}^{\infty} \sum_{F(n)=m} 2^{-f(n)-m} \\ &\geq \sum_{m=1}^{\infty} 2^{-m}(2^m - 1) = \infty.\end{aligned}$$

Now we come to the real argument. Consider the unit interval $[0, 1]$ laid out in a circle so that 0 and 1 are identified. The partial sums $G_n = \sum_{i=1}^{n} 2^{-g(i)}$ mark off successive intervals $I_n \equiv [G_{n-1}, G_n) \bmod 1$ on this circle. We exploit the fact that a point on the circle will be contained in many of these intervals.

For each $x \in \{0, 1\}^*$, the associated cylinder is the set

$$\Gamma_x = \{\omega \in \{0, 1\}^\infty : \omega_{1:l(x)} = x\}.$$

The geometric interpretation is $\Gamma_x = [0.x, 0.x + 2^{-l(x)})$. Let

$$A_n = \left\{x \in \{0, 1\}^n : \Gamma_x \bigcap I_n \neq \varnothing\right\}.$$

It follows from the divergence of G_n that for any ω there is an infinite set $N \subseteq \mathcal{N}$ consisting of the infinitely many n such that prefixes $\omega_{1:n}$ belong to A_n. Describing a prefix $\omega_{1:n} \in A_n$ by its index in the set, we have

$$\begin{aligned}C(\omega_{1:n}|n) &\leq \log |A_n| + O(1) \leq \log \frac{G_n - G_{n-1}}{2^{-n}} + O(1) \\ &= n - g(n) + O(1) \leq n - f(n).\end{aligned}$$

□

Corollary 2.5.1 Let $f(n)$ be as in Theorem 2.5.1. Then $C(\omega_{1:n}) \leq n - f(n)$ infinitely often, provided $C(n|n - f(n)) = O(1)$ (such as $f(n) = \log n$).

Proof. This is a slightly stronger statement than Theorem 2.5.1. Let p be a description of $\omega_{1:n}$, given n, of length $C(\omega_{1:n}|n)$. Let $f(), g()$ be as in the proof of Theorem 2.5.1, and let q be an $O(1)$-length program that retrieves n from $n - f(n)$. Then $l(\bar{q}p) \leq n - f(n)$, since $l(p) \leq n - g(n)$ and $g(n) - f(n)$ rises unboundedly. We pad $\bar{q}p$ to length $n - f(n)$, obtaining $\bar{q}1^{n-f(n)-l(\bar{q}p)-1}0p$. This is a description for $\omega_{1:n}$. Namely, we first determine $\bar{q}$ to find q. The length of the total description is $n - f(n)$. By assumption, q computes n from this. Given n we can retrieve $\omega_{1:n}$ from p. □

In [P. Martin-Löf, *Z. Wahrsch. Verw. Geb.*, 19(1971), 225–230], Corollary 2.5.1 is stated to hold without the additional condition of n being retrievable from $n - f(n)$ by an $O(1)$-bit program. The proof of this fact is attributed to [P. Martin-Löf, On the oscillation of the complexity of infinite binary sequences (Russian), unpublished, 1965].

There is a simple proof that yields a result only slightly weaker than this. We first prove the result with the particular function $\log n$ substituted for $f(n)$ in the statement of the theorem. We then iterate the construction to prove a version of the theorem with a particular function $g(n)$ substituted for $f(n)$. Our result is almost tight, since for functions $h(n)$ that are only slightly larger than $g(n)$ the sum $\sum 2^{-h(n)}$ is finite. Moreover, the proof is explicit in that we exhibit $g(n)$.

Let ω be an infinite binary sequence, and $\omega_{1:m}$ an m-length prefix of ω. If $\omega_{1:m}$ is the nth binary string in the lexicographic order $0, 1, 00, \ldots$, that is, $n = \omega_{1:m}$, $m = l(n)$, then $C(\omega_{1:n}) \leq C(\omega_{m+1:n}) + c$, with c a constant independent of n and m. Namely, with $O(1)$ additional bits of information, we can trivially reconstruct the nth binary string $\omega_{1:m}$ from the length $n - l(n)$ of $\omega_{m+1:n}$. By Theorem 2.1.2, we find that $C(\omega_{m+1:n}) \leq n - l(n) + c$ for some constant c independent of n, whence the claimed result with $f(n) = \log n$ follows.

It is easy to see that we get a stronger result by iteration of the above argument. There are infinitely many n such that the initial segment $y = \omega_{1:n}$ can be divided as $y = y_1 y_2 \ldots y_k$, where $l(y_1) = 2$, $y_1 = l(y_2)$, $y_2 = l(y_3)$, $\ldots$, $y_{k-1} = l(y_k)$. Use the usual pairing between natural numbers and binary strings. Clearly, given y_k we can easily compute all of $\omega_{1:n}$, by determining y_{k-1} as the binary representation of $l(y_k)$, y_{k-2} as the binary representation of $l(y_{k-1})$, and so on until we obtain $l(y_1) = 2$. (If $\omega_{1:24} = 010011101100000110100001$, then $y_1 = 01$, which corresponds to natural number 4, so $y_2 = 0011$, which corresponds to the natural number 18, and finally $y_3 = 101100000110100001$. Hence, given y_3, we can easily determine all of $\omega_{1:24}$.) Then, for infinitely many n,

$$C(\omega_{1:n}) \leq \kappa + c,$$

for κ determined by $n = \kappa + l(\kappa) + l(l(\kappa)) + \cdots + 2$, all terms greater than or equal to 2. If we put $g(n) = n - \kappa$, then it can be shown that $\sum 2^{-g(n)} = \infty$ but that for only slightly larger functions $h(n) > g(n)$ the sum converges (for example $h(n) = g(n) +$ the number of terms in $g(n)$). There is an interesting connection with prefix codes, Section 1.11.1.

Our approach in this proof makes it easy to say something about the frequency of these complexity oscillations. Define a wave function w by $w(1) = 2$ and $w(i) = 2^{w(i-1)}$. Then the above argument guarantees that there are at least k values $n_1, n_2, \ldots, n_k$ less than $n = w(k)$ such that $C(\omega_{1:n_i}) \leq n_i - g(n_i) + c$ for all $i = 1, 2, \ldots, k$. Obviously, this can be improved.

In Figure 2.4 we display the complexity oscillations of initial segments of high-complexity sequences as they must look according to Theorems 2.5.1, 2.5.4, 2.5.5. The upper bound on the oscillations, $C(\omega_{1:n}) = n + O(1)$, is reached infinitely often for almost every high-complexity

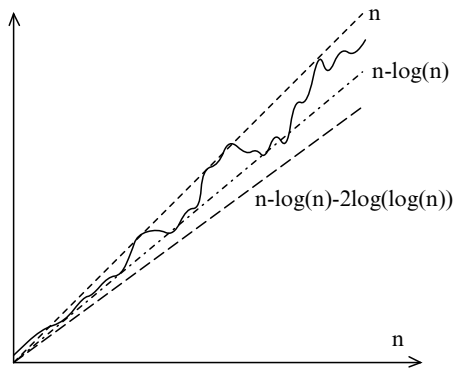


FIGURE 2.4. Complexity oscillations of initial segments of high–complexity infinite sequences

sequence. Furthermore, the oscillations of all high-complexity sequences stay above $n - \log n - 2\log\log n$, but dip infinitely often below $n - \log n$.

Having shown that the complexity of prefixes of each infinite sequence drops infinitely often unboundedly below the maximum value, we now want to show that Theorem 2.5.1 is optimal. But let's first discuss what this means. Clearly, Theorem 2.5.1 is nontrivial only for very irregular sequences x. It holds trivially for regular sequences such as $\omega = 0^\infty$, where the complexity of the initial segments $\omega_{1:n}$ is about $\log n$. We will prove that it is sharp for those ω that are maximally random. To make this precise, we must define rigorously what we mean by a random infinite sequence. It is of major significance that in so doing we also succeed in completing in a satisfactory way the program outlined by von Mises.

Due to the complexity oscillations, the idea of identifying random infinite sequences with those such that $C(\omega_{1:n}) \geq n - c$, for all n, is trivially infeasible. That is the bad news. In contrast, a similar approach in Section 2.4 for finite binary sequences turned out to work just fine. Its justification was found in Martin-Löf's important insight that to justify any proposed definition of randomness one has to show that the sequences that are random in the stated sense satisfy the several properties of stochasticity we know from the theory of probability. Instead of proving each such property separately, one may be able to show, once and for all, that the random sequences introduced possess, in an appropriate sense, all possible properties of stochasticity.

The naive execution of the above ideas in classical mathematics is infeasible as shown by the following example: Consider as sample space S the set of all one-way infinite binary sequences. The cylinder $\Gamma_x = \{\omega : \omega = x \ldots\}$ consists of all infinite binary sequences starting with the finite binary sequence x. For

instance, $\Gamma_\epsilon = S$. The uniform distribution λ on the sample space is defined by $\lambda(\Gamma_x) = 2^{-l(x)}$. That is, the probability of an infinite binary sequence ω starting with a finite initial segment x is $2^{-l(x)}$. In probability theory it is general practice that if a certain property, such as the law of large numbers, or the law of the iterated logarithm, has been shown to have probability one, then one calls this a *law of randomness*. For example, in our sample space the law of large numbers says that $\lim_{n\to\infty}(\omega_1 + \cdots + \omega_n)/n = \frac{1}{2}$. If A is the set of elements of S that satisfy the law of large numbers, then it can be shown that $\lambda(A) = 1$.

Generalizing this idea for S with measure μ, one may identify *any* set $B \subseteq S$ such that $\mu(B) = 1$ with a law of randomness, namely, "to be an element of B." Elements of S that do not satisfy the law "to be an element of B" form a set of measure zero, a *null set.* It is natural to call an element of the sample space 'random' if it satisfies all laws of randomness. Now we are in trouble. For each element $\omega \in S$, the set $B_\omega = S - \{\omega\}$ forms a law of randomness. But the intersection of all these sets B_ω of probability one is empty. Thus, no sequence would be random if we require that all laws of randomness that exist be satisfied by a random sequence.

It turns out that a constructive viewpoint enables us to carry out this program mathematically without such pitfalls. In practice, all laws that are proved in probability theory to hold with probability one are effective in the sense of Section 1.7. A straightforward formalization of this viewpoint is to require a law of probability to be partial recursive in the sense that we can effectively test whether it is violated. This suggests that the set of random infinite sequences should not be defined as the intersection of all sets of measure one, but as the intersection of all sets of measure one with a recursively enumerable complement. The latter intersection is again a set of measure one with a recursively enumerable complement. Hence, there is a single effective law of randomness that can be stated as the property "to satisfy all effective laws of randomness," and the infinite sequences have this property with probability one.

2.5.2 Sequential Randomness Tests

As in Section 2.4, we define a test for randomness. However, this time the test will not be defined on the entire sequence (which is impossible for an effective test and an infinite sequence), but for each finite binary string. The value of the test for an infinite sequence is then defined as the maximum of the values of the test on all prefixes. Since this suggests an effective process of sequential approximations, we call it a sequential test. Below, we need to use notions of continuous sample spaces and measures as treated in Section 1.6.

Definition 2.5.1 Let μ be a recursive probability measure on the sample space $\{0,1\}^\infty$. A total function $\delta : \{0,1\}^\infty \to \mathcal{N} \bigcup \{\infty\}$ is a *sequential μ-test* (sequential Martin-Löf μ-test for randomness) if

1. $\delta(\omega) = \sup_{n\in\mathcal{N}}\{\gamma(\omega_{1:n})\}$, where $\gamma : \mathcal{N} \to \mathcal{N}$ is a (total) lower semicomputable function ($V = \{(m, y) : \gamma(y) \geq m\}$ is a recursively enumerable set); and

2. $\mu\{\omega : \delta(\omega) \geq m\} \leq 2^{-m}$, for each $m \geq 0$.

If μ is the uniform measure λ, then we often use simply the term *sequential test.*

We can require γ to be a recursive function without changing the notion of a sequential μ-test. By definition, for each lower semicomputable function γ there exists a recursive function ϕ with $\phi(x, k)$ nondecreasing in k such that $\lim_{k\to\infty} \phi(x, k) = \gamma(x)$. Define a recursive function γ' by $\gamma'(\omega_{1:n}) = \phi(\omega_{1:m}, k)$ with $\langle m, k\rangle = n$. Then, $\sup_{n\in\mathcal{N}}\{\gamma'(\omega_{1:n})\} = \sup_{n\in\mathcal{N}}\{\gamma(\omega_{1:n})\}$.

Example 2.5.1 Consider $\{0, 1\}^\infty$ with the uniform measure $\lambda(x) = 2^{-l(x)}$. An example of a sequential λ-test is to test whether there is some 1 in an even position of $\omega \in \{0, 1\}^\infty$. Let

$$\gamma(\omega_{1:n}) = \begin{cases} \frac{1}{2}n & \text{if } \sum_{i=1}^{n/2} \omega_{2i} = 0, \\ 0 & \text{otherwise.} \end{cases}$$

The number of x's of length n such that $\gamma(x) \geq m$ is at most $2^{n/2}$ for any $m \geq 1$. Therefore, $\lambda\{\omega : \delta(\omega) \geq m\} \leq 2^{-m}$ for $m > 0$. For $m = 0$, $\lambda\{\omega : \delta(\omega) \geq m\} \leq 2^{-m}$ holds trivially. A sequence ω is random with respect to this test if $\delta(\omega) < \infty$. Thus, a sequence ζ with 0's in all even locations will have $\delta(\zeta) = \infty$ and it will fail the test, and hence ζ is not random with respect to this test. Notice that this is not a very strong test of randomness. For example, a sequence $\eta = 010^\infty$ will pass δ and be considered random with respect to this test. This test filters out only some nonrandom sequences with all 0's at the even locations and cannot detect other kinds of regularities. ◇

We continue the general theory of sequential testing. If $\delta(\omega) = \infty$, then we say that ω *fails* δ, or that δ *rejects* ω. Otherwise, ω *passes* δ. By definition, the set of ω's that are rejected by δ has μ-measure zero, and conversely, the set of ω's that pass δ has μ-measure one.

Suppose that for a test δ we have $\delta(\omega) = m$. Then there is a prefix y of ω, with $l(y)$ minimal, such that $\gamma(y) = m$ for the γ used to define δ. Then obviously, *each* infinite sequence ζ that starts with y has $\delta(\zeta) \geq m$. The set of such ζ is $\Gamma_y = \{\zeta : \zeta = y\rho, \rho \in \{0, 1\}^\infty\}$, the cylinder generated by y. Geometrically speaking, Γ_y can be viewed as the set of all real numbers $0.y\ldots$ corresponding to the half-open *interval* $I_y = [0.y, 0.y + 2^{-l(y)})$. For the uniform measure, $\lambda(\Gamma_y)$ is equal to $2^{-l(y)}$, the common length of I_y.

In terms of common statistical tests, the critical regions are formed by the nested sequence $V_1 \supseteq V_2 \supseteq \cdots$, where V_m is defined as $V_m = \{\omega : \delta(\omega) \geq m\}$, for $m \geq 1$. We can formulate the definition of V_m as

$$V_m = \bigcup \{\Gamma_y : (m, y) \in V\}.$$

In geometric terms, V_m is the union of a set of subintervals of $[0, 1)$. Since V is recursively enumerable, so is the set of intervals whose union is V_m. For each critical section we have $\mu(V_m) \leq 2^{-m}$ (in the measure we count overlapping intervals only once).

Now we can reformulate the notion of passing a sequential test δ with associated set V:

$$\delta(\omega) < \infty \text{ iff } \omega \notin \bigcap_{m=1}^{\infty} V_m.$$

Definition 2.5.2 Let $\mathbf{V}$ be the set of all sequential μ-tests. An infinite binary sequence ω, or the binary-represented real number $0.\omega$, is called *μ-random* if it passes *all* sequential μ-tests:

$$\omega \notin \bigcup_{V \in \mathbf{V}} \bigcap_{m=1}^{\infty} V_m.$$

For each sequential μ-test V, we have $\mu(\bigcap_{m=1}^{\infty} V_m) = 0$, by Definition 2.5.1. We call $\bigcap_{m=1}^{\infty} V_m$ a *constructive μ-null set* . Since there are only countably infinitely many sequential μ-tests V, it follows from standard measure theory that

$$\mu\left(\bigcup_{V \in \mathbf{V}} \bigcap_{m=1}^{\infty} V_m\right) = 0,$$

and we call the set $U = \bigcup_{V \in \mathbf{V}} \bigcap_{m=1}^{\infty} V_m$ the *maximal constructive μ-null set*.

In analogy to Section 2.4, we construct a lower semicomputable function $\delta_0(\omega|\mu)$, the universal sequential μ-test that incorporates (majorizes) all sequential μ-tests $\delta_1, \delta_2, \ldots$ and that corresponds to U.

Definition 2.5.3 A *universal sequential μ-test* f is a sequential μ-test such that for each sequential μ-test δ_i there is a constant $c \geq 0$ such that for all $\omega \in \{0, 1\}^\infty$, we have $f(\omega) \geq \delta_i(\omega) - c$.

Theorem 2.5.2 *There is a universal sequential μ-test (denoted by $\delta_0(\cdot|\mu)$).*

Proof. Start with the standard enumeration $\phi_1, \phi_2, \ldots$ of partial recursive functions from $\mathcal{N}$ into $\mathcal{N} \times \mathcal{N}$. In this way, we enumerate all recursively enumerable sets of pairs of integers in the form of the ranges of the ϕ_i's. In particular, we have included *any* recursively enumerable set V associated with a sequential μ-test. The *only* thing we have to do is to eliminate those ϕ_i's that do not correspond to a sequential μ-test.

First, effectively modify each ϕ (we drop the subscript for convenience) to a function ψ such that range ϕ equals range ψ, and ψ has the special property that if $\psi(n) < \infty$, then also $\psi(1), \psi(2), \ldots, \psi(n-1) < \infty$.

Second, use each ψ to construct a function $\delta : \{0,1\}^\infty \to \mathcal{N}$ by approximation from below. In the algorithm, at each stage of the computation the arrays γ and δ contain the current approximation to the function values of γ and δ. This is doable because the nonzero part of the approximation is always finite.

Step 1. Initialize γ and δ by setting $\delta(\omega) := \gamma(\omega_{1:n}) := 0$, for all $\omega \in \{0,1\}^\infty, n \in \mathcal{N}$, and set $i := 0$.

Step 2. Set $i := i + 1$; compute $\psi(i)$ and let its value be (y, m).

Step 3. **If** $\gamma(y) \geq m$ **then go to** Step 2 **else** set $\gamma(y) := m$.

Step 4. **If** $\mu(\bigcup_{\gamma(z)\geq k} \Gamma_z) > 2^{-k}$ for some k, $k = 1, \ldots, m$ **{**since μ is a recursive function we can effectively test whether the new assignment violates Definition 2.5.1**}** **then** terminate **{**the computation of δ is finished**}** **else** set $\delta(\omega) := \max\{m, \delta(\omega)\}$, for all $\omega \in \Gamma_y$, and **go to** Step 2.

For the uniform measure $\lambda(\Gamma_x) = 2^{-l(x)}$, the conditional in Step 4 simplifies for $i = 1$ to $m > l(y)$. In case the range of ψ is already a sequential μ-test, then the algorithm never finishes but approximates δ from below. If the range of ψ is not a sequential μ-test, then at some point the conditional in Step 4 is violated and the computation of δ terminates. The resulting δ is a sequential μ-test, even a recursive one. If the conditional in Step 4 is never violated, but the computation of ψ diverges for some argument, then δ is trivially a lower semicomputable sequential μ-test.

Executing this procedure on all functions in the list $\phi_1, \phi_2, \ldots$, we obtain an effective enumeration $\delta_1, \delta_2, \ldots$ of all sequential μ-tests (and only sequential μ-tests). The function $\delta_0(\cdot|\mu)$ defined by

$$\delta_0(\omega|\mu) = \sup_{i\in\mathcal{N}}\{\delta_i(\omega) - i\}$$

is a universal sequential μ-test.

First, $\delta_0(\omega|\mu)$ is a lower semicomputable function, since the set $\{(m,\omega) : \delta_0(\omega|\mu) \geq m\}$ is recursively enumerable. The proof that $\delta_0(\cdot|\mu)$ is a sequential μ-test, and majorizes all other sequential μ-tests additively, is completely analogous to the proof for the similarly defined universal P-test in Section 2.4. □

Any other sequential μ-test δ_i can discover at most a constant additional amount randomness in a sequence ω than does $\delta_0(\cdot|\mu)$. That is, $\delta_i(\omega) \leq \delta_0(\omega|\mu) + i$, for all ω.

The difference between any two universal sequential μ-tests $\delta_0(\cdot|\mu)$ and $\delta'_0(\cdot|\mu)$ is bounded by a constant: $|\delta_0(\omega|\mu) - \delta'_0(\omega|\mu)| \leq c$, with c independent of ω. We are now ready to separate the random infinite sequences from the nonrandom ones.

Definition 2.5.4 Let the sample space $\{0,1\}^\infty$ be distributed according to μ, and let $\delta_0(\cdot|\mu)$ be a universal sequential μ-test. An infinite binary sequence ω is *μ-random in the sense of Martin-Löf* if $\delta_0(\omega|\mu) < \infty$. We call such a sequence simply *random*, where both μ and Martin-Löf are understood. (This is particularly interesting for μ is the uniform measure.)

Note that *this definition does not depend on the choice of the particular universal sequential μ-test* with respect to which the level is defined. Hence, the line between random and nonrandom infinite sequences is drawn sharply without dependence on a reference μ-test. Clearly, the set of infinite sequences that are not random in the sense of Martin-Löf forms precisely the maximal constructive μ-null set of μ-measure zero we have constructed above. Therefore, we have the following result.

Theorem 2.5.3 *Let μ be a recursive measure. The set of μ-random infinite binary sequences has μ-measure one.*

We say that the universal sequential μ-test $\delta_0(\cdot|\mu)$ rejects an infinite sequence with probability zero, and we conclude that a randomly selected infinite sequence passes all effectively testable laws of randomness with probability one.

The main question remaining is the following: Let λ be the uniform measure. Can we formulate a universal sequential λ-test in terms of complexity? In Theorem 2.4.2 the universal (nonsequential) test is expressed in that way. The most obvious candidate for the universal sequential test would be $f(\omega) = \sup_{n\in\mathcal{N}}\{n - C(\omega_{1:n})\}$, but it is improper. To see this, it is simplest to notice that $f(\omega)$ would declare *all* infinite ω to be nonrandom, since $f(\omega) = \infty$, for all ω, by Theorem 2.5.1. The same would be the case for $f(\omega) = \sup_{n\in\mathcal{N}}\{n - C(\omega_{1:n}|n)\}$, by about the same proof. It is difficult to express a universal sequential test precisely in terms of C-complexity. But in Chapter 3 we show that it is easy to separate the random infinite sequences from the nonrandom ones in terms of prefix complexity.

2.5.3 Characterization of Random Sequences

How accurately can we characterize the set of infinite random sequences in complexity terms? It turns out that we can sandwich them between a proper superset in Theorem 2.5.4, page 152, and a proper subset in Theorem 2.5.5, page 153. First we bound the amplitude of the oscillations of random sequences.

Definition 2.5.5 The infinite series $\sum 2^{-f(n)}$ is *recursively convergent* if there is a recursive sequence $n_1, n_2, \ldots$ such that

$$\sum_{n=n_m}^{\infty} 2^{-f(n)} \leq 2^{-m}, \ m = 1, 2, \ldots \ .$$

Theorem 2.5.4 *Let $f(n)$ be a recursive function such that $\sum_{n=1}^{\infty} 2^{-f(n)} < \infty$ is recursively convergent. If an infinite binary sequence ω is random with respect to the uniform measure, then $C(\omega_{1:n}|n) \geq n - f(n)$, from some n onward.*

Proof. We define a sequential test that is passed only by the ω's satisfying the conditions in the theorem. For each m, let the critical section V_m consist of all infinite binary sequences ω such that there exists an $n \geq n_m$ for which $C(\omega_{1:n}|n) < n - f(n)$. In other words, V_m consists of the union of all intervals $[0.\omega_{1:n}, 0.\omega_{1:n} + 2^{-n})$ satisfying these conditions. We have to show that this is a sequential test. We can recursively enumerate the intervals that constitute V_m, and therefore $V_1, V_2, \ldots$ is a sequence of recursively enumerable sets. Obviously, the sequence is nested. For every large enough n, at most $2^{n-f(n)}$ strings y of length n satisfy $C(y|n) < n - f(n)$ (Theorem 2.2.1). Hence, with λ the uniform measure, we have, for all m,

$$\lambda(V_m) \leq \sum_{n=n_m}^{\infty} 2^{n-f(n)} 2^{-n} \leq 2^{-m}.$$

Therefore, the sequence of critical regions forms a sequential test, and $\lambda(\bigcap_{m=1}^{\infty} V_m) = 0$. That is, $\bigcap_{m=1}^{\infty} V_m$ is a constructive λ-null set associated with a sequential test. Consequently, it is contained in the maximal constructive λ-null set, which consists precisely of the sequences that are not random according to Martin-Löf. □

We can say fairly precisely which functions f satisfy the condition in Theorem 2.5.4. Examples are $f(n) = 2 \log n$ and $f(n) = \log n + 2 \log \log n$. In fact, the function $g(n)$ used in the proof of Theorem 2.5.1 is about the borderline. It is *almost* sufficient that $f(n) - g(n)$ be unbounded for a recursive function $f(n)$ to satisfy the condition in Theorem 2.5.4. A precise form of the borderline function is given in Exercise 3.6.7 on page 231.

With Theorem 2.5.4 we have shown that Theorem 2.5.1 is optimal in the sense that it gives the deepest complexity dips to be expected from sequences that are random with respect to the uniform measure (in the sense of Martin-Löf). But also, we have found a property of infinite sequences in terms of C that is implied by randomness with respect to the uniform measure. Is there also a property of infinite sequences in terms of complexity C that implies randomness with respect to the uniform measure?

Theorem 2.5.5 *Let ω be an infinite binary sequence.*

(i) *If there exists a constant c such that $C(\omega_{1:n}) \geq n - c$, for infinitely many n, then ω is random in the sense of Martin-Löf with respect to the uniform measure.*

(ii) *The set of ω for which there exists a constant c and infinitely many n such that $C(\omega_{1:n}) \geq n - c$ has uniform measure one.*

Proof. We first prove the following claim:

Claim 2.5.1 Let $\omega \in \{0,1\}^\infty$. There exists a positive constant c such that $C(\omega_{1:n}|n) \geq n - c$ for infinitely many n iff there exists a positive constant c such that $C(\omega_{1:n}) \geq n - c$ for infinitely many n.

Proof. (Only if) This is the easy direction, since conditional information does not increase complexity. Hence, for all ω, n, we have $C(\omega_{1:n}|n) \leq C(\omega_{1:n})$ up to a fixed additive constant.

(If) For some fixed constant c_1, we have for all ω, n that $C(\omega_{1:n}) \leq C(\omega_{1:n}|n) + 2l(n - C(\omega_{1:n}|n)) + c_1$. (The right-hand side of the inequality is the length of a description of $\omega_{1:n}$.) Since in the 'If' direction we assume that $C(\omega_{1:n}) \geq n - c$ for some c and infinitely many n, we obtain $n - C(\omega_{1:n}|n) \leq c + c_1 + 2l(n - C(\omega_{1:n}|n))$ for this infinite sequence of n's. But that is possible only if there is a constant c_2 such that $n - C(\omega_{1:n}|n) \leq c_2$ for the same infinite sequence of n's, which finishes the proof. □

(i) Below, a 'test' is a 'λ-test' with λ the uniform measure. We denote sequential tests by δ's, and (nonsequential) tests of Section 2.4 by γ's. Let $\delta_0(\cdot|\lambda)$ denote the universal sequential test with respect to the uniform measure λ, and let $\gamma_0(\cdot|L)$ denote the universal test with respect to the uniform distribution L.

Since a sequential test is a fortiori a test, there is a constant c such that $\delta_0(\omega_{1:n}|\lambda) \leq \gamma_0(\omega_{1:n}|L) + c$, for all ω and n. By choosing the specific universal test of Theorem 2.4.2, we have $\delta_0(\omega_{1:n}|\lambda) \leq n - C(\omega_{1:n}|n)$ up to a constant. Since δ_0 is monotonic nondecreasing,

$$\lim_{n\to\infty} \delta_0(\omega_{1:n}|\lambda) \leq \liminf_{n\to\infty} (n - C(\omega_{1:n}|n)) + O(1).$$

For those ω's satisfying the assumption in the statement of the theorem, by Claim 2.5.1 the right-hand side of the inequality is finite. By Theorem 2.4.2, therefore, such ω's are random with respect to the uniform measure.

(ii) For each c and n, let $V_{c,n}$ denote the union of the set of intervals associated with prefixes $\omega_{1:n}$ of infinite binary sequences ω such that $C(\omega_{1:n}|n) \geq n - c$. Let λ be the uniform measure. There are at most 2^{n-c} strings of length less than $n - c$ and therefore at least $2^n - 2^{n-c}$ strings x of length n satisfying $C(x|n) \geq n - c$. Hence, for each m and c we have

$$\lambda\left(\bigcup_{n=m}^{\infty} V_{c,n}\right) \geq \lambda\left(V_{c,m}\right) \geq (2^m - 2^{m-c})2^{-m} = 1 - 2^{-c}.$$

Since the right-hand term is independent of m, we also have

$$\lambda\left(\bigcap_{m=1}^{\infty} \bigcup_{n=m}^{\infty} V_{c,n}\right) \geq 1 - 2^{-c}.$$

Since

$$\bigcap_{m=1}^{\infty} \bigcup_{n=m}^{\infty} V_{c,n} \subseteq \bigcap_{m=1}^{\infty} \bigcup_{n=m}^{\infty} V_{c+1,n}$$

for all positive integers c, we obtain

$$\begin{aligned}\lambda\left(\bigcup_{c=1}^{\infty} \bigcap_{m=1}^{\infty} \bigcup_{n=m}^{\infty} V_{c,n}\right) &= \lim_{c\to\infty} \lambda\left(\bigcap_{m=1}^{\infty} \bigcup_{n=m}^{\infty} V_{c,n}\right) \\ &\geq \lim_{c\to\infty} (1 - 2^{-c}) = 1.\end{aligned}$$

The formula $\bigcup_{c=1}^{\infty} \bigcap_{m=1}^{\infty} \bigcup_{n=m}^{\infty} V_{c,n}$ denotes precisely the set of infinite sequences ω for which there exists a positive integer constant c such that for infinitely many n, $C(\omega_{1:n}|n) \geq n - c$ holds. A fortiori, the same holds without the conditional up to an additive constant. □

We conclude that the set of sequences satisfying the condition in Theorem 2.5.4 contains the set of sequences that are random with respect to the uniform measure (in Martin-Löf's sense), and the latter contains the set of sequences satisfying the condition in Theorem 2.5.5; see Figure 2.5. There, the inner oval is the set of sequences satisfying Theorem 2.5.5; the middle oval is the set of Martin-Löf random sequences; the outer oval is the set of sequences satisfying Theorem 2.5.4. Containment is always proper. The outer oval in its turn is properly contained in the set defined by Exercise 2.5.5 on page 159. Although the differences between

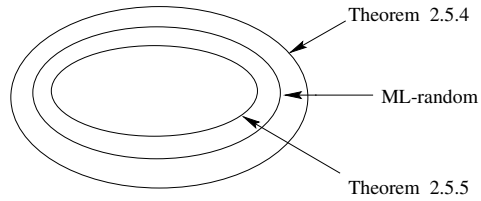


FIGURE 2.5. Three notions of 'chaotic' infinite sequences

each pair of the three sets are nonempty, they are not large, since all three sets have uniform measure one. For instance, the set of random sequences not satisfying the condition of Theorem 2.5.5 has uniform measure zero. In Example 3.6.19 on page 237 it is shown that the condition involved precisely characterizes a stronger notion of randomness than Martin-Löf randomness.

The combination of Theorems 2.5.4 and 2.5.5 enables us to give a relation between the upward and downward oscillations of the complexity of prefixes of the random sequences satisfying the property in Theorem 2.5.5 as follows:

Corollary 2.5.2 If f is a recursive function such that $\sum 2^{-f(n)}$ converges recursively and $C(\omega_{1:n}) \geq n - c$ for some constant c and for infinitely many n, then $C(\omega_{1:n}) \geq n - f(n)$ from some n onward.

The universal sequential μ-test characterizes the set of infinite random sequences. There are other ways to do so. We give an explicit characterization of infinite random sequences with respect to the uniform measure in Theorem 3.6.1 and its corollary, page 222. This characterization is an exact expression in terms of the prefix complexity developed in Chapter 3.

Apart from sequential tests as developed above there are other types of tests for randomness of individual infinite sequences. The extended theory of randomness tests can be given only after we have treated lower semicomputable semimeasures in Sections 4.5.7 and 4.5.6. There we give exact expressions for μ-tests for randomness, for arbitrary recursive μ.

We recall von Mises's classic approach to obtaining infinite random sequences ω as treated in Section 1.9, which formed a primary inspiration to the work reported in this section. It is of great interest whether one can, in his type of formulation, capture the intuitively and mathematically satisfying notion of infinite random sequence in the sense of Martin-Löf. According to von Mises, an infinite binary sequence ω is random (a collective) if

1. ω has the property of frequency stability with limit p; that is, if $f_n = \omega_1 + \omega_2 + \cdots + \omega_n$, then the limit of f_n/n exists and equals p.

2. Any subsequence of ω chosen according to an admissible place-selection rule has frequency stability with the same limit p as in condition 1.

One major problem was how to define 'admissible,' and one choice was to identify it with Church's notion of selecting a subsequence $\zeta_1\zeta_2\ldots$ of $\omega_1\omega_2\ldots$ by a partial recursive function ϕ by $\zeta_n = \omega_m$ if $\phi(\omega_{1:r}) = 0$ for precisely $n-1$ instances of r with $r < m-1$ and $\phi(\omega_{1:m-1}) = 0$. We called these ϕ 'place-selection rules according to von Mises–Wald–Church,' and the resulting sequences ζ Mises–Wald–Church random.

Definition 2.5.6 *Mises–Wald–Church stochastic sequences* are Mises–Wald–Church random sequences with limiting frequency $\frac{1}{2}$.

In Section 1.9 we stated that there are Mises–Wald–Church stochastic sequences that do not satisfy effectively testable properties of randomness such as the law of the iterated logarithm or the infinite recurrence property. (Such properties are by definition satisfied by sequences that are Martin-Löf random.) In fact, the distinction between the Mises–Wald–Church stochastic sequences and the Martin-Löf random ones is quite large, since there are Mises–Wald–Church stochastic sequences ω such that $C(\omega_{1:n}) = O(f(n)\log n)$ for every unbounded, nondecreasing, total recursive function f; see also Exercise 2.5.13 on page 161. Such Mises–Wald–Church stochastic sequences are very nonrandom sequences from the viewpoint of Martin-Löf randomness, where one requires that $C(\omega_{1:n})$ be asymptotic to n. See R.P. Daley, *Math. Systems Theory*, 9(1975), 83–94. Note, that although a Mises–Wald–Church stochastic sequence may have very low Kolmogorov complexity, it in fact has very high time-bounded Kolmogorov complexity. See Exercise 7.1.7 on page 546.

If we consider also sequences with limiting frequencies different from $\frac{1}{2}$, then it is obvious that there are sequences that are random according to Mises–Wald–Church, but not according to Martin-Löf. Namely, *any* sequence ω with limiting relative frequency p has complexity $C(\omega_{1:n}) \leq H(p)n + o(n)$, where $H(p) = p\log 1/p + (1-p)\log 1/(1-p)$ ($H(p)$ is Shannon's binary entropy). This means that for each $\epsilon > 0$ there are Mises–Wald–Church random sequences ω with $C(\omega_{1:n}) < \epsilon n$ for all but finitely many n.

On the other hand, clearly all Martin-Löf random sequences are also Mises–Wald–Church stochastic (each admissible selection rule is an effective sequential test).

This suggests that we have to liberate our notion of admissible selection rule somewhat in order to capture the proper notion of an infinite random sequence using von Mises's approach. A proposal in this direction was given by A.N. Kolmogorov [*Sankhyā*, Ser. A, 25(1963), 369–376] and D.W. Loveland [*Z. Math. Logik Grundl. Math.* 12(1966), 279–294].

Definition 2.5.7 A *Kolmogorov–Loveland admissible selection function* to select an infinite subsequence $\zeta_1\zeta_2\ldots$ from $\omega = \omega_1\omega_2\ldots$ is a partial recursive function $\phi : \{0,1\}^* \to \mathcal{N}\times\{0,1\}$ from binary strings to (index, bit) pairs (not necessarily defined on all of $\{0,1\}^*$). The subsequence selection is a two-phase process. First we select an intermediate sequence z of elements of ω. Initially, $z = \epsilon$. If $z = z_1z_2\ldots z_m$ is the intermediate sequence selected after m steps, and $\phi(z) = (i,a)$ $(a \in \{0,1\})$,

then $z\omega_i$ is the intermediate sequence selected after $m+1$ steps. However, ϕ is partial, so $\phi(z)$ may not be defined. Moreover, we are not allowed to select the same bit position more than once. If for some z either $\phi(z)$ is undefined, or $\phi(z) = (i, \cdot)$ while $\phi(z') = (i, \cdot)$ for some proper initial segment z' of z, then the process terminates with finite z; otherwise z is infinite. If $z = \omega_{i_1}\omega_{i_2}\ldots$, selected by the sequence of associated ϕ-values $\phi(\epsilon) = (i_1, a_1)$, $\phi(\omega_{i_1}) = (i_2, a_2)$, ..., then we obtain the target selected subsequence ζ by erasing every ω_{i_j} in z with associated $a_j = 0$. The resulting ζ may be finite or infinite, but it is only the infinite ζ in which we are interested. If ω is such that for every Kolmogorov–Loveland admissible selection function the selected sequence $\zeta_1\zeta_2\ldots$—if infinite—has the same limiting frequency as the original ω, and we assume the limiting frequency $\frac{1}{2}$, then ω is called a *Kolmogorov–Loveland stochastic sequence*.

The term 'Kolmogorov–Loveland random sequence' is currently used for infinite binary sequences for which there is no computable nonmonotonic betting strategy that succeeds on it. The strategy has success if it obtains unbounded gain in the limit while betting successively on the nonmonotonically selected bits of the sequence. This is in contrast to the above 'stochastic' definition, which expresses the weaker requirement that there be no computable nonmonotonic selection rule that selects an infinite biased sequence from the original sequence. For more details, see [W. Merkle, J.S. Miller, A. Nies, J. Reimann, F. Stephan, *Ann. Pure Appl. Logic*, 138(2006), 183–210].

As compared to the Mises–Wald–Church approach, the liberation of the selection rule mechanism is contained in the fact that the order of succession of the terms in the subsequence chosen is not necessarily the same as that of the original sequence. Thus, the Kolmogorov–Loveland selection rules are called *nonmonotonic*. In comparison, it is not obvious whether a subsequence $\zeta_1\zeta_2\ldots$ selected from a Kolmogorov–Loveland stochastic sequence $\omega_1\omega_2\ldots$ by a Kolmogorov–Loveland place-selection rule is itself a Kolmogorov–Loveland stochastic sequence. Note that the analogous property necessarily holds for Mises–Wald–Church stochastic sequences. This matter was resolved in [W. Merkle, *J. Symbol. Logic*, 68(2003), 1362–1376], where it was shown that there is a Kolmogorov–Loveland stochastic sequence from which one can select effectively (and in fact monotonically) a subsequence that is no longer Kolmogorov–Loveland stochastic.

Clearly, the set of Kolmogorov–Loveland stochastic sequences is contained in the set of Mises–Wald–Church stochastic sequences. In turn, the set of Kolmogorov–Loveland stochastic sequences contains the set of Martin-Löf random sequences. If $\omega_1\omega_2\ldots$ is Kolmogorov–Loveland stochastic, then clearly $\zeta_1\zeta_2\ldots.$ defined by $\zeta_i = \omega_{\sigma(i)}$, with σ being a recursive permutation, is also Kolmogorov–Loveland stochastic. The Mises–Wald–Church notion of stochasticity does not have this important property of randomness of staying invariant under recursive permutation. Loveland gave the required counterexample in the cited reference. Hence, the containment of the set of Kolmogorov–Loveland stochastic sequences in the set of Mises–Wald–Church stochastic sequences is proper. This follows also from the cited result that the Kolmogorov–Loveland stochastic sequences are not closed under monotonic effective selection rules, earlier observed by A.K. Shen; see the acknowledgments in [W. Merkle, *Ibid.*].

This leaves the question whether the containment of the set of Martin-Löf random sequences in the set of Kolmogorov–Loveland stochastic sequences is proper. Kolmogorov has stated in [*Problems Inform. Transmission*, 5(1969), 3–4] without proof that there exists a Kolmogorov–Loveland stochastic sequence ω such that $C(\omega_{1:n}) = O(\log n)$. But An.A. Muchnik (1958–2007)—not to be confused with his father A.A. Muchnik—showed that this is false, since no ω with $C(\omega_{1:n}) \leq cn + O(1)$ for a constant $c < 1$ can be Kolmogorov–Loveland stochastic. Nonetheless, containment is proper, since A.K. Shen [*Soviet Math. Dokl.*, 38:2(1989), 316–319] has shown that there exists a Kolmogorov–Loveland stochastic sequence that is not random in Martin-Löf's sense. Therefore, the problem of giving a satisfactory definition of infinite Martin-Löf random sequences in the form proposed by von Mises has not yet been solved. See also [A.N. Kolmogorov and V.A. Uspensky, *Theory Probab. Appl.*, 32(1987), 389–412; V.A. Uspensky, A.L. Semenov, and A.K. Shen, *Russ. Math. Surveys*, 45:1(1990), 121–189; An.A. Muchnik, A.L. Semenov, V.A. Uspensky, *Theoret. Comput. Sci.*, 2:207(1998), 1362–1376].

Exercises

2.5.1. [13] Consider $\{0,1\}^\infty$ under the uniform measure. Let $\omega = \omega_1\omega_2\ldots \in \{0,1\}^\infty$ be random in the sense of Martin-Löf.

(a) Show that $\zeta = \omega_n\omega_{n+1}\ldots$ is Martin-Löf random for each n.

(b) Show that $\zeta = x\omega$ is Martin-Löf random for each finite string x.

Comments. Source: C. Calude, I. Chitescu, *Bolletino U.M.I.*, (7) 3-B(1989), 229–240.

2.5.2. [21] Consider $\{0,1\}^\infty$ under the uniform measure. Let $\omega = \omega_1\omega_2\ldots \in \{0,1\}^\infty$.

(a) Show that if there is an infinite recursive set I such that either for all $i \in I$ we have $\omega_i = 0$ or for all $i \in I$ we have $\omega_i = 1$, then ω is not random in the sense of Martin-Löf.

(b) Show that if the set $\{i : \omega_i = 0\}$ contains an infinite recursively enumerable subset, then ω is not random in the sense of Martin-Löf.

Comments. Source: C. Calude and I. Chitescu, *Ibid.*

2.5.3. [21] Let $\omega = \omega_1\omega_2\ldots$ be any infinite binary sequence. Define $\zeta = \zeta_1\zeta_2\ldots$ by $\zeta_i = \omega_i + \omega_{i+1}$, $i \geq 1$. This gives a sequence over the alphabet $\{0,1,2\}$. Show that ζ is not random in the sense of Martin-Löf under the uniform measure (extend the definition from binary to ternary sequences).

Comments. Hint: the blocks 02 and 20 do not occur in ζ. Source: R. von Mises, *Probability, Statistics and Truth*, Dover, 1981.

2.5.4. [23] Let ω be any infinite binary sequence. Show that for all constants c there are infinitely many m such that for all n with $m \leq n \leq 2m$, $C(\omega_{1:n}) \leq n - c$.

Comments. We are guaranteed to find long complexity oscillations (of length m) in an infinite binary sequence ω relatively near the beginning (namely $\omega_{m:2m}$), even if ω is Martin-Löf random. Source: H.P. Katseff and M. Sipser, *Theoret. Comput. Sci.*, 15(1981), 291–309.

2.5.5. [M19] Let f be such that $\sum 2^{-f(n)} < \infty$. Show that the set of infinite binary sequences ω satisfying $C(\omega_{1:n}|n) \geq n - f(n)$ for all but finitely many n has uniform measure 1.

Comments. Hint: The number of y with $l(y) = n$ such that $C(y) < n - f(n)$ is less than $2^{n-f(n)}$. This implies that the probability that this inequality is satisfied is less than $2^{-f(n)}$, and the result follows by the Borel–Cantelli lemmas; see Exercise 1.10.2 on page 64. This set of ω's properly contains the set defined by Theorem 2.5.4. Source: P. Martin-Löf, *Z. Wahrsch. Verw. Geb.*, 19(1971), 225–230.

2.5.6. [09] Consider infinite binary sequences ω with respect to the uniform measure. Show that with probability one there exists a constant c such that $C(\omega_{1:n}|n) \geq n - c$ for infinitely many n.

Comments. Hint: use Theorem 2.5.5, Item (ii), and Claim 2.5.1. Source: P. Martin-Löf, *Z. Wahrsch. Verw. Geb.*, 19(1971), 225–230. .

2.5.7. [19] Consider infinite binary sequences ω with respect to the uniform measure. Show that if f is a recursive function and $\sum 2^{-f(n)}$ converges recursively and $C(\omega_{1:n}) \geq n - c$ for some constant c and infinitely many n, then $C(\omega_{1:n}) \geq n - f(n)$ for all but finitely many n.

Comments. This formulation establishes a connection between upward and downward oscillations of the complexity of prefixes of almost all (random) infinite binary sequences. For f we can take $f(n) = 2 \log n$, or $f(n) = \log n + 2 \log \log n$, and so on. Hint: combine Theorems 2.5.4, 2.5.5. Source: P. Martin-Löf, *Z. Wahrsch. Verw. Geb.*, 19(1971), 225–230.

2.5.8. [19] Show that there exists an infinite binary sequence ω and a constant $c > 0$ such that $\liminf_{n\to\infty} C(\omega_{1:n}|n) \leq c$, but for any unbounded function f we have $\limsup_{n\to\infty} C(\omega_{1:n}|n) \geq n - f(n)$.

Comments. The oscillations can have amplitude $\Omega(n)$. Hint: use the n-strings defined above. Construct $\omega = y_1 y_2 \ldots$ from finite strings y_i. Let c be a fixed independent constant. For odd i choose y_i such that $y_1 \ldots y_i$ is an n-string, which implies that $C(y_{1:i}|l(y_{1:i})) < c$. For even i choose y_i as a long enough random string so that $C(y_{1:i}|l(y_{1:n})) \geq l(y_{1:n}) - f(l(y_{1:n}))$. Source: H.P. Katseff and M. Sipser, *Theoret. Comput. Sci.*, 15(1981), 291–309.

2.5.9. [39] Consider the Lebesgue measure λ on the set of intervals contained in $[0, 1)$ defined by $\lambda(\Gamma_y) = 2^{-l(y)}$. (Recall that for each finite

binary string y the cylinder Γ_y is the set of all infinite strings ω starting with y.) Let ω be an infinite binary sequence such that for every recursively enumerable sequence $A_1, A_2, \ldots$ of sets of intervals with the property that the series $\sum_i \lambda(A_i) < \infty$ converges, ω is contained in only finitely many A_i. Show that for the ω's defined this way, the *Solovay random sequences* are precisely the infinite binary sequences that are random in the sense of Martin-Löf with respect to the uniform measure.

Comments. In Martin-Löf's definition of randomness (with respect to the uniform measure) of infinite binary sequences, he required that $\lambda(A_i) \leq 2^{-i}$. That definition is equivalent to stipulating the existence of some regulator of convergence $f(i) \to \infty$ that is recursive and nondecreasing such that $\lambda(A_i) \leq 2^{-f(i)}$. Solovay's definition has the advantage that it does not require such a regulator. Source: R.M. Solovay, *Lecture Notes*, 1975, unpublished; and G.J. Chaitin, *Algorithmic Information Theory*, Cambridge University Press, 1987; A. K. Shen, *Soviet Math. Dokl.*, 38:2(1989), 316–319.

2.5.10. [35] (a) Show that for every positive constant c there is a positive constant c' such that $\{\omega_{1:n} : C(\omega_{1:n}; n) \geq n - c\} \subseteq \{\omega_{1:n} : C(\omega_{1:n}|n) \geq n - c'\}$.

(b) Use the observation in Item (a) to show that Theorem 2.5.4 holds for the uniform complexity measure $C(\cdot; l(\cdot))$.

(c) Show that if f is a recursive function and $\sum 2^{-f(n)} = \infty$, then for all infinite ω we have $C(\omega_{1:n}; n) \leq n - f(n)$ for infinitely many n. Hence, Theorem 2.5.1 holds for uniform complexity.

Comments. Hint for Item (a): define the notion of a (universal) uniform test as a special case of Martin-Löf's (universal) test. Compare this result with the other exercises to conclude that whereas the uniform complexity tends to be higher in the low-complexity region, the length-conditional complexity tends to be higher in the high-complexity region. Source: D.W. Loveland, *Inform. Contr.*, 15(1969), 510–526.

2.5.11. • [31] Show that the following statements are equivalent for an infinite binary sequence ω: For some constant c and infinitely many n, possibly different in each statement,

$$C(\omega_{1:n}|n) \geq n - c,$$
$$C(\omega_{1:n}; n) \geq n - c,$$
$$C(\omega_{1:n}) \geq n - c.$$

Comments. Hint: use Claim 2.5.1 and Exercise 2.5.10. In view of Theorem 2.5.5 these conditions equivalently imply that ω is random in the sense of Martin-Löf. Source: R.P. Daley, pp. 113–122 in: *Computational Complexity*, ed. R. Rustin, *Courant Comput. Sci. Symp.* 7(1971).

2.5.12. [30] (a) Show that the following statements are equivalent for an infinite binary sequence ω: There exists a c such that for infinitely many n, possibly different in each statement,

$$C(\omega_{1:n}|n) \leq c,$$
$$C(\omega_{1:n}; n) \leq l(n) + c,$$
$$C(\omega_{1:n}) \leq l(n) + c.$$

The sequences thus defined are called *pararecursive sequences*.

(b) Show that no pararecursive sequence is random in the sense of Martin-Löf.

Comments. Comparison with the other exercises shows that the recursive sequences are contained in the pararecursive sequences. It also shows that the pararecursive sequences have the cardinality of the continuum, so this containment is proper. R.P. Daley [*J. Symb. Logic,* 41(1976), 626–638] has shown that the lower semicomputable sequences sequences (which properly include the characteristc sequences of recursively enumerable sets) are pararecursive, and this containment is proper by the same argument as before. Hint for Item (b): use Theorem 2.5.4. Despite Item (b), there are pararecursive sequences that are close to being random. For any unbounded function f, there is a pararecursive sequence ω such that for infinitely many n we have $C(\omega_{1:n}|n) \geq n - f(n)$; see Exercise 2.5.8 on page 159. Source: H.P. Katseff and M. Sipser, *Theoret. Comput. Sci.*, 15(1981), 291–309.

2.5.13. [43] Let A be the set of Mises–Wald–Church stochastic sequences (with $p = \frac{1}{2}$). The admissible place-selection rules are the partial recursive functions.

(a) Show that there is an $\omega \in A$ such that for each unbounded, nondecreasing, total recursive function f, we have $C(\omega_{1:n}; n) \leq f(n) \log n$ from some n onward.

(b) Show that for all $\omega \in A$, there is a constant c such that $C(\omega_{1:n}; n) \geq \log n - c$ from some n onward.

Consider the larger class $B \supset A$ that is defined just like A but with the admissible place-selection rules restricted to the total recursive functions.

(c) Show that there is an $\omega \in B$ such that $C(\omega_{1:n}; n) \leq f(n)$ from some n onward, for each f as in Item (a).

(d) Show that for each $\omega \in B$, for each constant c, we have $C(\omega_{1:n}; n) \geq c$ from some n onward.

Comments. Compare this with the text following Definition 2.5.6 on page 156. This shows that there are Mises–Wald–Church stochastic sequences of quite low complexity, and that it makes a difference whether

the admissible place-selection rules are partial recursive or total recursive. Source: R.P. Daley, *Math. Systems Theory*, 9 (1975), 83–94. Item (a) is proved using Item (c), which is attributed to D.W. Loveland, and uses a construction (LMS algorithm) in D.W. Loveland, *Z. Math. Logik*, 12(1966), 279–294. Compare with Exercise 7.1.7, page 546, to see what happens when we impose a total recursive time bound on the decoding process.

2.5.14. [35] Show that there is no Mises–Wald–Church stochastic sequence ω (with limiting frequency $\frac{1}{2}$) and with $C(\omega_{1:n}) = O(\log n)$.

Comments. This exercise was open in the second edition of this book, solved in [W. Merkle, *J. Comput. Syst. Sci.*, 74:3(2008), 350–357]. Compare Exercise 2.5.13.

2.5.15. [33] (a) Show that there exists an infinite binary sequence ω that is random with respect to the uniform measure, but for each constant c there are only finitely many n such that $C(\omega_{1:n}|n) > n - c$ (the condition of Theorem 2.5.5 does not hold).

(b) Show that there exists an infinite binary sequence ω satisfying (i) $C(\omega_{1:n}) > n - f(n)$ from some n onward and $\sum 2^{-f(n)}$ converges recursively (the condition in Theorem 2.5.4 holds), and (ii) ω is not random with respect to the uniform measure.

Comments. Thus, each containment in the nested sequence of sets of infinite binary sequences that satisfy the condition in Theorem 2.5.4, randomness according to Martin-Löf, and the condition in Theorem 2.5.5, as in Figure 2.5, is proper. Source, C.P. Schnorr, *Math. Systems Theory*, 5(1971), 246–258.

2.5.16. [21] (a) Show that none of the variants of algorithmic complexity, such as $C(x)$, $C(x|l(x))$, and $C(x; l(x))$, is invariant with respect to cyclic shifts of the strings.

(b) Show that all these variants coincide to within the logarithm of the minimum of all these measures.

Comments. Hint: use the idea in the proof of Theorem 2.5.4. This invariance cannot be expected from any complexity measure in this book at all. Source: C.P. Schnorr, pp. 193–211 in: R.E. Butts and J. Hintikka, eds., *Basic Problems in Methodology and Linguistics*, D. Reidel, 1977.

2.5.17. [36] (a) Consider an infinite sequence of zeros and ones generated by independent tosses of a coin with probability p $(0 < p < 1)$ for 1. Define sequential Bernoulli tests (in analogy with Section 2.5 and Exercise 2.4.4 on page 142). Show that there exists a universal sequential Bernoulli test δ_0. An infinite binary sequence ω is a Bernoulli sequence if $\delta_0(\omega) < \infty$. Show that the set of Bernoulli sequences has measure

one with respect to the measure induced in the set of infinite binary sequences interpreted as independent $(p, 1-p)$ Bernoulli trials.

(b) In our definition of infinite Bernoulli sequences no restrictions were laid on the limiting behavior of the relative frequency, such as, for instance, required in Condition 1 of Definition 1.9.1 of an infinite random sequence (collective) ω, where we require that $\lim_{n\to\infty} f_n/n = p$ for some p $(0 < p < 1)$ (Section 1.9). The relative frequency f_n/n of ones in increasingly longer prefixes $\omega_{1:n}$ of a collective ω does not oscillate indefinitely, but converges to a definite limit. Show that remarkably, this is also the case for an infinite Bernoulli sequence ω.

(c) By the law of large numbers, *all* real numbers p in $[0, 1]$ occur as limit frequencies $\lim_{n\to\infty} f_n/n$ for infinite random binary sequences ω, and not only the recursive ones. Show that in contrast, for infinite Bernoulli sequences ω, the limit relative frequency cannot vanish, $\lim_{n\to\infty} f_n/n = 0$, unless $\omega_n = 0$ for all n.

Comments. Hint for Item (b): for an arbitrary rational $\epsilon > 0$ construct a sequential Bernoulli test that rejects at level 2^{-m} if $|f_i/i - f_j/j| > \epsilon$, for some $i, j \geq h(m)$, for some suitable nondecreasing total recursive function. Compare with the universal Bernoulli test of Exercise 2.4.4 on page 142. Hint for Item (c): this is the infinite analogue of the phenomenon in Item (b) of Exercise 2.4.4 on page 142. From Item (c) we conclude that an event with vanishing limit frequency is actually impossible. This is in stark contrast with von Mises's explicit statement of the opposite for his conception of random sequences (collectives) [R. von Mises, *Probability, Statistics and Truth*, Dover, 1981 (Reprinted)]. Source: P. Martin-Löf, *Inform. Contr.*, 9(1966), 602–619. Additionally we mention the following result [L.A. Levin, *Sov. Math. Dokl.* , 14(1973), 1413–1416]. Suppose we are given a constructively closed family $\mathcal{M}$ of measures (this notion is defined naturally on the space of measures). Let a test f be called *uniform* for $\mathcal{M}$ if for all measures in $\mathcal{M}$, for all positive integers k, the measure of outcomes ω where $f(\omega) > k$ is at most 2^{-k}. There exists a universal uniform test.

2.5.18. [37] Let μ be a recursive measure on the sample space $\{0, 1\}^\infty$. Recall from Section 2.5 Martin-Löf's construction of a constructive μ-null set using a notion of sequential test V with associated critical regions $V_1 \supseteq V_2 \supseteq \cdots$ of measures $\mu(V_i) \leq 2^{-i}$, for $i \geq 1$. A constructive μ-null set is a *total recursive μ-null set* if additionally, $f(i) = \mu(V_i)$ is a total recursive function. Call an infinite sequence *μ-random in the sense of Schnorr* if it is not contained in any total recursive μ-null set.

(a) Show that there is no universal total recursive μ-null set that contains all others.

(b) Show that the set of sequences that are μ-random in the sense of Martin-Löf is a subset of the set of sequences that are μ-random in the sense of Schnorr.

(c) An $\omega \in \{0,1\}^\infty$ is an *atom* with respect to μ if $\mu(\omega) > 0$. A measure μ is called *discrete* if the set of atoms of μ has μ-measure one. (Obviously all atoms of recursive μ are recursive sequences.) Show that the Schnorr-μ-random sequences coincide with the Martin-Löf-μ-random sequences iff μ is discrete.

Comments. Item (c) implies that for continuous μ (such as the uniform measure), Schnorr randomness is weaker than Martin-Löf randomness. The notion of total recursive null sets is the recursive analogue of the intuitionistic notion of sets of measure zero by L.E.J. Brouwer [A. Heyting, *Intuitionism, an Introduction*, North-Holland, 1956]. Sometimes the following statement is called *Schnorr's thesis:* "A sequence behaves within all effective procedures like a μ-random sequence iff it is μ-random in the sense of Schnorr." Source: C.P. Schnorr, pp. 193–211 in: R.E. Butts and J. Hintikka, eds., *Basic Problems in Methodology and Linguistics*, D. Reidel, 1977.

2.5.19. [M42] We abstract away from levels of significance and concentrate on the arithmetic structure of statistical tests. Statistical tests are just Π^0_n null sets, for some n (Exercise 1.7.21, page 46). The corresponding definition of randomness is defined as, "an infinite binary sequence is Π^0_n-random with respect to a recursive measure μ if it is not contained in any Π^0_n set V with μ-measure zero." Has the set of Π^0_n-random sequences μ-measure one?

Comments. Source: H. Gaifman and M. Snir, *J. Symb. Logic*, 47(1982), 495–548.

2.5.20. [M43] We assume familiarity with the unexplained notions below. We leave the arithmetic hierarchy of Exercise 2.5.19 and consider hyperarithmetic sets. Define an infinite binary sequence to be *hyperarithmetically random* if it belongs to the intersection of all hyperarithmetic sets of measure one. (A hyperarithmetic set can be regarded as a constructive version of the restriction to Borel sets that is usually accepted in probability theory—Section 1.6. The specific Borel sets considered there are always obtained by applying the Borelian operations to recursive sequences of previously defined sets, which means precisely that they are hyperarithmetical.)

(a) Show that the set of sequences that are hyperarithmetically random is a Σ^1_1 set of measure one (Σ^1_1 in the *analytic hierarchy*).

(b) Show that a hyperarithmetic sequence is not hyperarithmetically random.

(c) Show that the set of hyperarithmetically random sequences is not hyperarithmetical.

Comments. Already A. Wald proposed to sharpen von Mises's notion of randomness by defining a sequence to be random if it possesses all properties of probability one that are expressible within a certain formalized logic such as that of *Principia Mathematica*. This exercise is a variation on this idea. Just as here, Wald's proposal can be expected to define a set of random strings that is no longer expressible in the language with which we started. (This does not happen for Martin-Löf random sequences as defined in Section 2.5 because of the existence of a universal sequential test.) However, with the present proposal, the resulting class of random strings, while escaping the hyperarithmetic hierarchy, does not escape us completely but belongs to a class of sets that can still be handled constructively. Source: P. Martin-Löf, pp. 73–78 in: *Intuitionism and Proof Theory*, A. Kino et al., eds., North-Holland, 1970. For related work more in the direction of Wald's ideas, see [P.A. Benioff, *J. Math. Phys.*, 17:5(1976), 618–628, 629–640; L. Longpré and V. Kreinovich, "Randomness as incompressibility: a non-algorithmic analogue," Tech. Rept. UTEP-CS-96-19, Univ. Texas El Paso, 1996]

2.6 Statistical Properties of Finite Sequences

Each individual infinite sequence generated by a $(\frac{1}{2}, \frac{1}{2})$ Bernoulli process (flipping a fair coin) has (with probability 1) the property that the relative frequency of zeros in an initial n-length segment goes to $\frac{1}{2}$ as n goes to infinity. Such randomness-related statistical properties of individual (high)-complexity finite binary sequences are often required in applications of incompressibility arguments. The situation for infinite random sequences is better studied, and therefore we look there first.

Definition 2.6.1 E. Borel has called an infinite sequence of zeros and ones *normal* in the scale of two if for each k, the frequency of occurrences of each block y of length k in the initial segment of length n goes to limit 2^{-k} as n grows unboundedly.

It is known that normality is not sufficient for randomness, since Champernowne's sequence 123456789101112 . . . is normal in the scale of ten. On the other hand, it is universally agreed that a random infinite sequence must be normal. (If not, then some blocks occur more frequently than others, which can be used to obtain better than fair odds for prediction.)

We know from Section 2.5 that each infinite sequence that is random with respect to the uniform measure satisfies all effectively testable properties of randomness: it is normal, it satisfies the so-called law of the iterated logarithm, the number of 1's minus the number of 0's in an

initial n-length segment is positive for infinitely many n and negative for another infinitely many n, and so on. While the statistical properties of infinite sequences are simple corollaries of the theory of Martin-Löf randomness, for finite sequences the situation is less simple. Here, we determine the frequencies of occurrence of substrings in strings of high Kolmogorov complexity. In Section 6.4.1 the similar question is treated for subgraphs of high-Kolmogorov-complexity graphs.

2.6.1 Statistics of 0's and 1's

In the finite case, randomness is a matter of degree, because it would be clearly unreasonable to say that a sequence x of length n is random and to say that a sequence y obtained by flipping the first bit 1 in x is nonrandom. What we can do is to express the degree of incompressibility of a finite sequence in the form of its Kolmogorov complexity, and then analyze the statistical properties of the sequence—for example, the number of 0's and 1's in it.

Since almost all finite sequences have about maximal Kolmogorov complexity, each individual maximal-complexity sequence must possess approximately the expected (average) statistical properties of the overall set. For example, we can a priori state that each high-complexity finite binary sequence is normal in the sense that each binary block of length k occurs about equally frequently for k relatively small. In particular, this holds for $k = 1$. However, in many applications we need to know exactly what 'about' and the 'relatively small' in this statement mean. In other words, we are interested in the extent to which Borel normality holds in relation to the complexity of a finite sequence.

Let x have length n. By Example 2.4.4, if $C(x|n) = n + O(1)$, then the number of zeros it contains is

$$\frac{n}{2} + O(\sqrt{n}).$$

Notation 2.6.1 The quantity $K(x|y)$ in this section satisfies

$$C(x|y) \leq K(x|y) \leq C(x|y) + 2\log C(x|y) + 1.$$

We can think of it as roughly the length of a self-delimiting version of a program p of length $l(p) = C(x|y)$. In Chapter 3 it is defined as 'prefix complexity.'

Definition 2.6.2 The class of *deficiency* functions is the set of functions $\delta : \mathcal{N} \to \mathcal{N}$ satisfying $K(n, \delta(n)|n - \delta(n)) \leq c_1$ for all n. (Hence, $C(n, \delta(n)|n - \delta(n)) \leq c_1$ for all n.)

In this way, we can retrieve n and $\delta(n)$ from $n - \delta(n)$ by a self-delimiting program of at most c_1 bits. We choose c_1 so large that each monotone

sublinear recursive function that we are interested in, such as $\log n$, $\sqrt{n}$, $\log\log n$, is such a deficiency function. The constant c_1 is a benchmark that stays fixed throughout this section.

We denote the number of 1's in a binary string $x \in \{0,1\}^*$ by $\#\text{ones}(x)$.

Lemma 2.6.1 *There is a constant c such that for all deficiency functions δ, for each n and $x \in \{0,1\}^n$, if $C(x) \geq n - \delta(n)$, then*

$$\left|\#\text{ones}(x) - \frac{n}{2}\right| \leq \sqrt{\frac{3}{2}(\delta(n) + c)n/\log e}. \tag{2.3}$$

Proof. A general estimate of the tail probability of the binomial distribution, with s_n the number of successful outcomes in n experiments with probability of success $0 < p < 1$, is given by Chernoff's bounds, Lemma 1.10.1 on page 61:

$$\Pr(|s_n - pn| > m) \leq 2e^{-m^2/3pn}. \tag{2.4}$$

Let s_n be the number of 1's in the outcome of n fair coin flips, which means that $p = \frac{1}{2}$. Define $A = \{x \in \{0,1\}^n : |\#\text{ones}(x) - \frac{1}{2}n| > m\}$ and apply Equation 2.4 to obtain

$$d(A) \leq 2^{n+1}e^{-2m^2/3n}.$$

We choose m such that for some constant c to be determined later,

$$\frac{2m^2 \log e}{3n} = \delta(n) + c.$$

We can compress any $x \in A$ in the following way:

1. Let s be a self-delimiting program to retrieve n and $\delta(n)$ from $n - \delta(n)$, of length at most c_1.
2. Given n and $\delta(n)$, we can effectively enumerate A. Let i be the index of x in such an effective enumeration of A. The length of the (not necessarily self-delimiting) description of i satisfies

$$\begin{aligned} l(i) \leq \log d(A) &\leq n + 1 - (2m^2 \log e)/3n \\ &= n + 1 - \delta(n) - c. \end{aligned}$$

The string si is padded to length $n + 1 - \delta(n) - c + c_1$. From si we can reconstruct x by first using $l(si)$ to compute $n - \delta(n)$, then computing n and $\delta(n)$ from s and $n - \delta(n)$, and subsequently enumerating A to obtain the ith element. Let T be the Turing machine embodying the procedure for reconstructing x. Then by Theorem 2.1.1,

$$C(x) \leq C_T(x) + c_T \leq n + 1 - \delta(n) - c + c_1 + c_T.$$

Choosing $c = c_1 + c_T + 2$, we obtain $C(x) < n - \delta(n)$, which contradicts the condition of the theorem. Hence, $|\#\text{ones}(x) - \frac{1}{2}n| \leq m$. □

It may be surprising at first glance, but there are no maximally complex sequences with about an equal number of zeros and ones. Equal numbers of zeros and ones is a form of regularity, and therefore lack of complexity. That is, for $x \in \{0,1\}^n$, if $|\#\text{ones}(x) - \frac{1}{2}n| = O(1)$, then the randomness deficiency $\delta(n) = n - C(x)$ is nonconstant (order $\log n$).

Lemma 2.6.2 *There is a constant c such that for all n and all $x \in \{0,1\}^n$, if*

$$\left|\#\text{ones}(x) - \frac{n}{2}\right| \leq 2^{-\delta(n)-c}\sqrt{n},$$

then $C(x) \leq n - \delta(n)$.

Proof. Let $m = 2^{-\delta(n)-c}\sqrt{n}$, with c a constant to be determined later. Let $A = \{x \in \{0,1\}^n : |\#\text{ones}(x) - \frac{1}{2}n| \leq m\}$. There is a constant c_2 such that there are only

$$d(A) \leq (2m+1)\binom{n}{n/2} \leq c_2 \frac{2^n m}{\sqrt{n}} \tag{2.5}$$

elements in A (use Stirling's approximation, Exercise 1.5.4 on page 17). Thus, for each $x \in A$, we can encode x by its index in an enumeration of A. We can find A from n and $\delta(n)$. We can find n and $\delta(n)$ from $n - \delta(n)$ by a self-delimiting program of size at most c_1. Altogether, this description takes $\log d(A) + c_1 = n - \delta(n) - c + c_1 + \log c_2$ bits. Let this process of reconstructing x be executed by Turing machine T. Choosing $c = c_1 + \log c_2 + c_T$ we obtain by Theorem 2.1.1,

$$C(x) \leq C_T(x) + c_T \leq n - \delta(n).$$

□

Example 2.6.1 We consider some particular values of $\delta(n)$. Set $\delta_1(n) = \frac{1}{2}\log n - \log\log n$. If $|\#\text{ones}(x) - \frac{1}{2}n| = O(\log n)$, then $C(x) \leq n - \delta_1(n) + O(1)$. Set $\delta_2(n) = \frac{1}{2}\log n$. If

$$\left|\#\text{ones}(x) - \frac{n}{2}\right| = O(1),$$

then $C(x) \leq n - \delta_2(n) + O(1)$. That is, if the number of 1's is too close to the number of 0's, then the complexity of the string drops significantly below its maximum. ◇

An incompressible string of length n cannot have precisely or almost $\frac{1}{2}n$ ones by Lemma 2.6.2. Then how many ones should an incompressible string contain? The next lemma shows that for an incompressible x having $j + \frac{1}{2}n$ ones, $K(j|n)$ must be at least about order $\log n$.

Lemma 2.6.3 *There is a constant c such that for all n and all $x \in \{0,1\}^n$, if*

$$\left|\#\text{ones}(x) - \frac{n}{2}\right| = j,$$

then $C(x|n) \leq n - \frac{1}{2}\log n + K(j|n) + c$.

Proof. Let $A = \{x \in \{0,1\}^n : |\#\text{ones}(x) - \frac{1}{2}n| = j\}$. There is a constant c_3 such that there are

$$d(A) \leq \binom{n}{n/2} \leq c_3 \frac{2^n}{\sqrt{n}} \tag{2.6}$$

elements in A (use Stirling's approximation, Exercise 1.5.4 on page 17). In order to enumerate elements in A, we need only to describe j and n. Thus, for any $x \in A$, we can encode x by its index i (in $\log d(A)$ bits) in an enumeration of A. With n given, we can recover x from an encoding of j in $K(j|n)$ bits, followed by i. This description of x, given n, takes $\log d(A) + K(j|n) \leq n - \frac{1}{2}\log n + \log c_3 + K(j|n)$ bits. Let T be the Turing machine embodying this procedure to recover x given n. Choosing $c = \log c_3 + c_T$, we have

$$C(x|n) \leq C_T(x|n) + c_T \leq n - \frac{1}{2}\log n + K(j|n) + c.$$

□

Example 2.6.2 For $j = O(1)$ we have $C(x|n) \leq n - \frac{1}{2}\log n + O(1)$, which is slightly stronger than the statement about unconditional $C(x)$ in Example 2.6.1. For $j = O(\sqrt{n})$ and j incompressible ($K(j|n) \geq \frac{1}{2}\log n$), we have $C(x|n) \leq n - O(1)$. Only for such j's is it possible that a number x is incompressible. ◇

2.6.2 Statistics of Blocks

The analysis up till now has been about the statistics of 0's and 1's. But in a normal infinite binary sequence, according to Definition 2.6.2 on page 166, each block of length k occurs with limiting frequency 2^{-k}. That is, blocks 00, 01, 10, and 11 should occur about equally often, and so on. Finite sequences will generally not be exactly normal, but normality will be a matter of degree. We investigate the block statistics for finite binary sequences.

Definition 2.6.3 Let $x = x_1 \ldots x_n$ be a binary string of length n, and y a much smaller string of length l. Let $p = 2^{-l}$ and $\#y(x)$ be the number of (possibly overlapping) distinct occurrences of y in x. For convenience, we assume that x wraps around, so that an occurrence of y starting at the end of x and continuing at the start also counts.

Theorem 2.6.1 *Assume the notation of Definition 2.6.3 with $l \leq \log n$. There is a constant c such that for all n and $x \in \{0,1\}^n$, if $C(x) \geq n - \delta(n)$, then*

$$|\#y(x) - pn| \leq \sqrt{\alpha pn},$$

with $\alpha = [K(y|n) + \log l + \delta(n) + c]3l/\log e$.

Proof. We prove by contradiction. Assume that n is divisible by l. (If it is not, then we can put x on a Procrustean bed to make its length divisible by l at the cost of having the above frequency estimate $\#y(x)$ plus or minus an error term of at most $l/2$.) There are l ways of dividing (the ring) x into $N = n/l$ contiguous equal-sized blocks, each of length l. For each such division $i \in \{0, 1, \ldots, l-1\}$, let $\#y(x,i)$ be the number of (now nonoverlapping) occurrences of block y. We apply the Chernoff bound, Equation 2.4, again. With $A = \{x \in \{0,1\}^n : |\#y(x,i) - pN| > m\}$ this gives $d(A) \leq 2^{n+1}e^{-m^2/3pN}$. We choose m such that for some constant c to be determined later,

$$\frac{m^2 \log e}{3pN} = K(\langle y, i\rangle|n) + \delta(n) + c.$$

To describe an element x in A, we now need only to enumerate A and indicate the index of x in such an enumeration. The description contains the following items:

1. *A description used to enumerate A.* Given $n - \delta(n)$, we can retrieve n and $\delta(n)$ using a self-delimiting description of at most c_1 bits. To enumerate A, we also need to know i and y. Therefore, given $n - \delta(n)$, the required number of bits to enumerate A is at most

$$K(\langle y, i, \delta(n), n\rangle|n - \delta(n)) \leq K(\langle y, i\rangle|n) + c_1.$$

2. *A description of the index of x.* The number of bits to code the index j of x in A is

$$\begin{aligned}\log d(A) &\leq \log\left(2^{n+1}e^{-m^2/3pN}\right)\\ &= n + 1 - \frac{m^2 \log e}{3pN}\\ &= n + 1 - K(\langle y, i\rangle|n) - \delta(n) - c.\end{aligned}$$

This total description takes at most $n + 1 - \delta(n) - c + c_1$ bits. Let T be a Turing machine reconstructing x from these items. According to Theorem 2.1.1, therefore

$$C(x) \leq C_T(x) + c_T \leq n + 1 - \delta(n) - c + c_1 + c_T.$$

With $c = c_1 + c_T + 2$ we have $C(x) < n - \delta(n)$, which contradicts the assumption of the theorem. Therefore, $|\#y(x,i) - pN| \leq m$, which in turn implies

$$|\#y(x,i) - pN| \leq \sqrt{\frac{K(\langle y,i\rangle|n) + \delta(n) + c}{\log e} 3pN}.$$

For each division i $(0 \leq i \leq l-1)$ this inequality follows from the $\delta(n)$ incompressibility of x. Notwithstanding the fact that occurrences of substrings in different divisions are dependent, the inequality holds for each division separately and independently. The theorem now follows by noting that $|\#y(x) - pn| = \sum_{i=0}^{l-1} |\#y(x,i) - pN|$, $K(\langle y,i\rangle|n) \leq K(y|n) + K(i|n) + O(1)$ and $K(i|n) \leq \log l + O(1)$. □

Similar to the analysis of blocks of length 1, the complexity of a string drops below its maximum in case some block y of length l occurs in one of the l block divisions, say i, with frequency exactly pN $(p = 1/2^l)$. Then we can point out x by giving n, y, i, and its index in a set of cardinality

$$\binom{N}{pN}(2^l - 1)^{N-pN} = O\left(\frac{2^{Nl}}{\sqrt{p(1-p)N}}\right).$$

Therefore,

$$C(x|\langle n,y\rangle) \leq n - \frac{1}{2}\log n + \frac{1}{2}(l + 3\log l) + O(1).$$

2.6.3 Length of Runs

It is known from probability theory that in a randomly generated finite sequence the *expectation* of the length of the longest run of zeros or ones is pretty high. For each individual finite sequence with high Kolmogorov complexity we are *certain* that it contains each block (say, a run of zeros) up to a certain length.

Theorem 2.6.2 *Let x of length n satisfy $C(x) \geq n - \delta(n)$. Then for sufficiently large n,* each *block y of length*

$$l = \log n - \log\log n - \log(\delta(n) + \log n) - O(1)$$

occurs at least once in x.

Proof. We are sure that y occurs at least once in x if $\sqrt{\alpha pn}$ in Theorem 2.6.1 is less than pn. This is the case if $\alpha < pn$, that is,

$$\frac{K(y|n) + \log l + \delta(n) + O(1)}{\log e} 3l < pn.$$

$K(y|n)$ is majorized by $l + 2\log l + O(1)$ (since $K(y|n) \leq K(y) + O(1)$) and $p = 2^{-l}$ with l set at

$$l = \log n - \log(3\delta(n)\log n + 3\log^2 n)$$

(which equals l in the statement of the theorem up to an additive constant). Substitution yields

$$\frac{l + 3\log l + \delta(n) + O(1)}{\log e} 3l < 3(\delta(n)\log n + \log^2 n),$$

and it is easy to see that this holds for sufficiently large n. □

Corollary 2.6.1 If $\delta(n) = O(\log n)$, then *each* block of length $\log n - 2\log\log n - O(1)$ occurs at least once in x.

In Lemma 6.9.1 we show that if $C(x|n,p) \geq n$ then no substring of length greater than $2\log n$ occurs (possibly overlapping) twice in x. Here, $n = l(x)$, and p is some fixed program used to reconstruct x from a description of length $C(x|n,p)$ and n.

Analyzing the proof of Theorem 2.6.2, we can improve the corollary for low values of $K(y|n)$.

Corollary 2.6.2 If $\delta(n) = O(\log\log n)$, then for each $\epsilon > 0$ and every large enough n, every string x of length n contains an all-zero run y (for which $K(y|n) = O(\log l)$) of length $l = \log n - (1+\epsilon)\log\log n + O(1)$.

Since there are $2^n(1 - O(1/\log n))$ strings x of length n with $C(x) \geq n - \log\log n + O(1)$, the *expected length of the longest run of consecutive zeros* if we flip a fair coin n times is at least l as in Corollary 2.6.2.

We show in what sense Theorem 2.6.2 is sharp. Let $x = uvw$, $l(x) = n$, and $C(x) \geq n - \delta(n)$. We can describe x by giving

1. A description of v in $K(v)$ bits;
2. The literal representation of uw;
3. A description of $l(u)$ in $\log n + \log\log n + 2\log\log\log n + O(1)$ bits.

Then, since we can find n by $n = l(v) + l(uw)$,

$$C(x) \leq n - l(v) + K(v) + \log n + (1 + o(1))\log\log n + O(1). \tag{2.7}$$

Substitute $C(x) = n - \delta(n)$ and $K(v) = o(\log\log n)$ (choose v to be very regular) in Equation 2.7 to obtain

$$l(v) \leq \delta(n) + \log n + (1 + o(1)) \log\log n.$$

This means that for instance, for each $\epsilon > 0$, no maximally complex string x with $C(x) = n + O(1)$ contains a run of zeros (or the initial binary digits of π) of length $\log n + (1 + \epsilon) \log\log n$ for n large enough and regular enough. By Corollary 2.6.2, on the other hand, such a string x *must* contain a run of zeros of length $\log n - (1 + \epsilon) \log\log n + O(1)$.

Exercises

2.6.1. [24] The great majority of binary strings of length n have a number of 0's in between $\frac{1}{2}n - \sqrt{n}$ and $\frac{1}{2}n + \sqrt{n}$. Show that there are x's of length n with $\frac{1}{2}n + \Omega(\sqrt{n})$ 0's, with $C(x) = n + O(1)$.

2.6.2. [29] Let $\lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} \omega_i = p$ for an infinite binary sequence $\omega = \omega_1\omega_2\ldots$, for some p between 0 and 1 (compare Section 1.9).

(a) Show that if $C(\omega_{1:n}) \sim n$, then $p = \frac{1}{2}$.

(b) Show that if $p = \frac{1}{4}$, then $C(\omega_{1:n}) \leq 0.82n$ asymptotically.

(c) Show that in general, if $c = p \log 1/p + (1 - p) \log 1/(1 - p)$, then $C(\omega_{1:n}) \leq cn + o(n)$. If p is about $\frac{1}{4}$, then $C(\omega_{1:n}) \leq 0.80n + o(n)$.

Comments. Source: P. Gács, *Lecture Notes on Descriptional Complexity and Randomness*, Manuscript, Boston University, 1987; attributed to A.N. Kolmogorov. Hint: Item (a), use Lemma 2.6.2.

2.6.3. [M35] A finite binary string x of length n is called δ-*random* if $C(x|n) \geq n - \delta$. A Turing machine place-selection rule R is a Turing machine that selects and outputs a (not necessarily consecutive) substring $R(x)$ from its input x. If R is the kth Turing machine in the standard enumeration, then $C(R) = C(k)$.

Show that for any $\epsilon > 0$, there exist numbers n_0 and $\mu > 0$ such that if $l(x) = n$, $l(R(x)) = r \geq n_0$, $\sum_{1\leq i\leq r} R(x)_i = m$, and $(\delta + C(R|n))/r < \mu$, then

$$\left|\frac{m}{r} - \frac{1}{2}\right| < \left(\frac{\delta + C(R|n) + 2.5 \log r}{(2 \log e - \epsilon)r}\right)^{1/2}.$$

Comments. δ-random sequences were introduced by A.N. Kolmogorov, *Lect. Notes Math.*, Vol. 1021, Springer-Verlag, 1983, 1–5. He noted that "sequences satisfying this condition have for sufficiently small δ the particular property of frequency stability in passing to subsequences." In this exercise we supply some quantitative estimates of frequency stability. Source: E.A. Asarin, *SIAM Theory Probab. Appl.*, 32(1987), 507–508. This exercise is used by Asarin to show that δ-random elements of

certain finite sets obey familiar probability-theoretic distribution laws. In particular, for some particular finite sets each δ-random element is ϵ-normal; see also E.A. Asarin, *Soviet Math. Dokl.*, 36(1988), 109–112.

2.7 Algorithmic Properties of C

By algorithmic properties of C we mean properties with a recursion-theoretic flavor as in Section 1.7. We have already met a few of these. By Theorem 2.3.1, the greatest monotonic lower bound on $C(n)$ is unbounded, but goes to infinity more slowly than any monotonic unbounded partial recursive function. By Theorem 2.3.2 the integer function $C(n)$ is not recursive. Nonetheless, by Theorem 2.3.3 the integer function $C(n)$ can be approximated arbitrarily closely from above; it is upper semicomputable. Unfortunately, at each stage of such an approximation process, for each size of error, there are infinitely many x such that the approximation of $C(x)$ and its real value are at least this error apart.

In fact, a much stronger statement holds: For each total recursive function f with $\lim_{x\to\infty} f(x) = \infty$ the set of x for which we can prove $C(x) > f(x)$ is finite (Theorem 2.7.1 Item (iii) below). Thus, if we choose $f(x) \ll l(x)$, then we know that $C(x) \gg f(x)$ for almost all x of each length, Theorem 2.2.1, yet we can prove this only for finitely many x.

Theorem 2.7.1 (i) *The set $A = \{(x,a) : C(x) \leq a\}$ is recursively enumerable, but not recursive.*

(ii) *Every partial recursive function $\phi(x)$ that is a lower bound on $C(x)$ is bounded.*

(iii) *Let $f(x)$ be a total recursive function with $g(x) \leq f(x) \leq l(x)$ for all x and some unbounded monotonic function g. Then the set $B = \{x : C(x) \leq f(x)\}$ is simple. That is, B is recursively enumerable and the complement of B is infinite but does not contain an infinite recursively enumerable subset.*

Proof. (i) That A is recursively enumerable follows immediately from Theorem 2.3.3. However, A is not recursive. Namely, if A is recursive, then we can compute $C(x)$ by asking the consecutive questions "is $C(x) \leq a$?" for $a := 0, 1, \ldots$, contradicting Theorem 2.3.2.

(ii) Let ϕ be a partial recursive function and define $D = \{x : \phi(x) \leq C(x)\}$. If D is finite, there is nothing to prove. Assume that D is infinite and ϕ is unbounded, by way of contradiction. Recursively enumerate the domain of definition of ϕ without repetition, and define a total recursive function g by $g(n)$ that equals the least x in this enumeration such that $\phi(x) \geq n$. For each n there is such an x, by the contradictory assumption.

If $\phi = \phi_k$ in the standard effective enumeration $\phi_1, \phi_2, \ldots$ of the partial recursive functions, as in Section 1.7, then $n \leq C(x) \leq l(n) + l(\bar{k})$, up to a constant. For n large enough we have a contradiction.

(iii) That B is recursively enumerable follows from Item (i). The complement of B is infinite by Theorem 2.2.1. We prove that B is simple. Let D be an infinite recursively enumerable set contained in the complement of B. The restriction $f_D(x)$ of $f(x)$ to D is a partial recursive lower bound for $C(x)$. By Item (ii), therefore $f_D(x)$ is bounded. Since $f(x)$ rises unboundedly with x this is possible only if D is finite. □

Corollary 2.7.1 The set RAND defined by $\{x : C(x) \geq l(x)\}$ is immune—it is infinite and has no infinite recursively enumerable subset. In fact, the proofs support stronger results in that the set B above is *effectively* simple and RAND is *effectively* immune (Exercise 2.7.6).

2.7.1 Undecidability by Incompressibility

This approach allows us to give a result similar to Lemma 1.7.6 on page 35, but with examples of undecidable statements that differ from the ones given by Gödel. Namely, for each formal system T, there is a constant c_T such that no formula of form "$C(x) \geq c_T$" is provable in T.

Example 2.7.1 If T is an axiomatizable sound theory whose axioms and rules of inference require about k bits to describe, then T cannot be used to prove the randomness of any number much longer than k bits. If the system could prove randomness for a number much longer than k bits, then the *first* such proof (first in an unending enumeration of all proofs obtainable by repeated application of axioms and rules of inference) could be used to derive a contradiction: an approximately k-bit program to find and print out the specific random number mentioned in this proof, a number whose smallest program is by assumption considerably larger than k bits. Therefore, even though most strings are random, we will never be able to explicitly exhibit a string of reasonable size that demonstrably possesses this property. Formally,

- Let T be an axiomatizable theory (T is a recursively enumerable set consisting of axioms and provable formulas). Hence, there is a k such that T is describable in k bits: $C(T) \leq k$.
- Let T be sound: all formulas in T are true (in the standard model of the natural numbers).
- Let $S_c(x)$ be a formula in T with the meaning "x is the lexicographically least binary string of length c with $C(x) \geq c$." Here x is a formal parameter and c an explicit constant, so $C(S_c) \leq \log c$ up to a fixed constant independent of T and c.

For each c, there exists an x such that $S_c(x) = \mathbf{true}$ is a true statement by a simple counting argument (Theorem 2.3.1). Moreover, S_c expresses that this x is unique. It is easy to see that combining the descriptions of T, S_c, we obtain a description of this x. Namely, for each candidate string y of length c, we can decide $S_c(y) = \mathbf{true}$ (which is the case for $y = x$) or $\neg S_c(y) = \mathbf{true}$ (which is the case for $y \neq x$) by simple enumeration of all proofs in T. (Here we use the soundness of T.) We need to distinguish the descriptions of T and S_c. We can do this by coding T's description in self-delimiting format; see Equation 1.4. This takes not more than $2k$ bits. Hence, for some fixed constant c' independent of T and c, we obtain $C(x) \leq 2k + \log c + c'$, which contradicts $C(x) > c$ for all $c > c_T$, where $c_T = 3k + c'$ for another constant c'. (A minor improvement of the argument shows that $c_T = k + 2\log k + c'$ suffices.) ◇

Corollary 2.7.2 There is a recursively enumerable set B with an infinite complement such that for every axiomatizable sound theory T there are only finitely many n for which the formula "$n \notin B$" is both true and provable in T. (But with finitely many exceptions, all infinitely many such formulas are true.)

Proof. Let B be the simple set in Theorem 2.7.1, Item (iii), and let $\bar{B}$ be its complement. Clearly, the set $D \subseteq \bar{B}$ of elements n that can be proved in T to belong to $\bar{B}$ is recursively enumerable. Since B is simple, its complement $\bar{B}$ does not contain an infinite recursively enumerable subset. Therefore, D is finite, which proves the theorem. □

We have formulated Corollary 2.7.2 so as to bring out some similarities and differences with Lemma 1.7.6 as clearly as possible. As pointed out in Section 1.7, the set K_0 used in Lemma 1.7.6 is complete, whereas the set B used in Corollary 2.7.2 is simple. According to generally accepted viewpoints in recursion theory, the set K_0 is different from the set B in an essential way. Therefore, we can regard the proofs of the existence of undecidable statements in sufficiently rich axiomatizable theories by Lemma 1.7.6 and Corollary 2.7.2 as essentially different.

The set K_0 is not only complete in the sense of Turing reducibility, it is also complete in the sense of many-to-one reducibility. A set A is *many-to-one* reducible to a set B if there exists a recursive function f such that for all x, $x \in A$ iff $f(x) \in B$. A set A is complete in the sense of many-to-one reducibility, *m-complete* for short, if A is recursively enumerable and all recursively enumerable sets B are many-to-one reducible to A. As it turns out, m-complete sets are nonrecursive. This raises the question, are all nonrecursive recursively enumerable sets m-complete? The answer was given by E. Post in 1944 by introducing simple sets as the first examples of nonrecursive recursively enumerable sets that are not m-complete. (For the notions of reducibility, completeness, and simple sets, see the exercises in Section 1.7, in particular Exercises 1.7.16 and 1.7.15.)

The set B used in Corollary 2.7.2 is a simple set and hence not m-complete. Therefore, although B is many-to-one reducible to K_0, the set K_0 is not many-to-one reducible to B. This shows that K_0 is of a so-called higher degree of unsolvability with respect to many-to-one reducibility than B, which is the substance of the viewpoint that they differ in an essential way.

In less formal terms the approaches are different because the first one can be viewed as a form of *Russell's paradox* and the other one as a form of the *Richard–Berry paradox*. Both paradoxes are described in [B. Russell and A.N. Whitehead, *Principia Mathematica*, Oxford, 1917]. While the first paradox formed the original incentive for the authors to supply the sophisticated logical foundation for set theory in the *Principia*, in a footnote they state that the second paradox "was suggested to us by Mr. G.G. Berry of the Bodleian Library."

The paradox due to Bertrand Russell (1872–1970) arises when the collection of all sets that are not members of themselves is considered as a set. If this collection is a member of itself, then it contradicts the set definition, but if it is not a member of itself, then by the set definition it is a member of itself (which is a contradiction as well). There is a close connection between Russell's paradox and the result of Gödel cited as Lemma 1.7.6 on page 35. We have seen that this result was proved by reducing the halting problem in the form of K_0 to the decision problem in a sufficiently strong, sound, axiomatized theory. Since K_0 is m-complete, this shows that any problem shown to be unsolvable *in this way* must have a degree of unsolvability at least as high as the maximal degree of unsolvability with respect to many-to-one reducibility as any recursively enumerable set.

The Richard–Berry paradox is the definition of a number as "the least number that cannot be defined in fewer than twenty words." Formalizing the notion of 'definition' as the shortest program from which a number can be computed by the reference machine U, it turns out that the quoted statement (reformulated appropriately) is not an *effective description*. This was essentially what we did in the proof of Corollary 2.7.2, by reducing the set B to the decision problem in a sufficiently strong, sound, axiomatizable theory. But B is of a lesser degree of unsolvability with respect to many-to-one reducibility than is K_0. Therefore, showing undecidability of sufficiently rich axiomatizable theories using Kolmogorov complexity *in this way* is essentially different from Gödel's original approach.

Gödel's first incompleteness theorem entails an *explicit* construction of a statement s, associated with each sufficiently strong, sound, axiomatized theory T, that is undecidable in T. Formula s simply says of itself "I am unprovable in T." In contrast, the construction in Corollary 2.7.2 says that for any sound axiomatized system T there is a constant $c_T < \infty$ such that *all* true statements with the meaning "$C(x) \geq c_T$" are unprovable in T. By Theorem 2.3.1 there are infinitely many such statements. Now suppose we have an effective procedure to find such constants for given theories, that is, a total recursive function ϕ such that $\phi(T) \geq c_T$ for all T. Then, unfortunately, Theorem 2.7.1, Item (ii), tells us that *no* effective procedure can determine for more than finitely many pairs

(x, T) whether $C(x) \geq \phi(T)$. This shows that in general, although the undecidable statements based on Kolmogorov complexity are plentiful for each theory, we cannot explicitly construct them. Thus, the new approach entails loss of constructivity.

2.7.2 Barzdins's Lemma

Using Kolmogorov complexity one can quantify the distinction between recursively enumerable sets and recursive sets. Let A be a set of natural numbers.

Definition 2.7.1 The *characteristic sequence* of $A \subseteq \mathcal{N}$ is an infinite binary sequence $\chi = \chi_1\chi_2\ldots$ defined by

$$\chi_i = \begin{cases} 1 & \text{if } i \in A, \\ 0 & \text{otherwise.} \end{cases}$$

If A is recursively enumerable, and also its complement consisting of the i's such that $\chi_i = 0$ is recursively enumerable, then $f(i) = \chi_i$ is recursive, and the conditional complexity $C(\chi_{1:n}|n)$ is bounded by a fixed constant for all n. (The converse also holds, Exercise 2.3.4 on page 131.) But in the general case of recursively enumerable sets A, the complexity $C(\chi_{1:n}|n)$ can grow unboundedly with n. However, this growth is at best logarithmically slow, which shows that such characteristic sequences are very nonrandom. For instance, they are not random in the sense of Martin-Löf according to Theorem 2.5.4. The result below is known as *Barzdins's lemma.* (Actually, J.M. Barzdins proved the sharper version of Exercise 2.7.2 on page 180.)

Theorem 2.7.2 (i) *Any characteristic sequence χ of a recursively enumerable set A satisfies $C(\chi_{1:n}|n) \leq \log n + c$ for all n, where c is a constant dependent on A (but not on n).*

(ii) *Moreover, there is a recursively enumerable set such that its characteristic sequence χ satisfies $C(\chi_{1:n}) \geq \log n$ for all n.*

Proof. (i) Since A is recursively enumerable, there is a partial recursive function ϕ such that $A = \{x : \phi(x) < \infty\}$. Dovetail the computations of $\phi(1), \phi(2), \ldots$. In this way, we enumerate A without repetitions in the order in which the computations of the $\phi(i)$'s terminate. The prefix $\chi_{1:n}$ can be reconstructed from the number m of 1's it contains. For if we know m, then it suffices to use ϕ to enumerate the elements of A until we have found m distinct such elements less than or equal to n. If the set of these elements is $B = \{a_1, a_2, \ldots, a_m\}$, then by assumption these are *all* elements in A that do not exceed n. Hence, from B we can reconstruct all 1's in $\chi_{1:n}$, and the remaining positions must be the 0's. In this way, we can reconstruct $\chi_{1:n}$, given n, from a description of ϕ and m. Since $m \leq n$ and $C(\phi) < \infty$, we have proved (i).

(ii) Let ϕ_0 be the additively optimal function ϕ_0 of Theorem 2.1.1. Define $\chi = \chi_1\chi_2\ldots$ by

$$\chi_i = \begin{cases} 1 & \text{if } \phi_0(i,i) = 0, \\ 0 & \text{if } \phi_0(i,i) \neq 0 \text{ or } \phi_0(i,i) = \infty. \end{cases}$$

Obviously, χ is the characteristic sequence of a recursively enumerable subset of the natural numbers. We prove that χ satisfies the property stated in the theorem. For suppose to the contrary that $C(\chi_{1:n}) < \log n$ for some n. This means that there is a short program p, of length less than $\log n$, that computes $\chi_{1:n}$. Since $p < n$ this implies that $\phi_0(p,p) = \chi_p$, which contradicts the definition of χ_p. □

The converse of Theorem 2.7.2, Item (i), does not hold in general. This follows by the construction of a meager set that is not recursively enumerable. For instance, let χ be the characteristic sequence of a set A and $C(\chi_{1:n}|n) \geq n - c$ for infinitely many n and a fixed constant c. By Theorem 2.5.5 such strings are abundant, and by Theorem 2.7.2, Item (i), we find that A is not recursively enumerable. Construct a sequence ζ by $\zeta = \chi_1\alpha_1\chi_2\alpha_2\ldots$ with $\alpha_i = 0^{f(i)}$, where $f(i)$ is some fast-growing total recursive function with an inverse. Obviously, if ζ is the characteristic sequence of set B, then B is not recursively enumerable. But also there is now another constant c such that $C(\zeta_{1:n}|n) \leq f^{-1}(n) + c$ for all n. Choosing f such that $\log\log f(n) = n$ gives

$$C(\zeta_{1:n}|n) \leq \log\log n + O(1).$$

Theorem 2.7.2, Item (i), cannot be improved to the unconditional "$C(\chi_{1:n}) \leq \log n + c$ for all n and some c," since all χ's satisfying this are recursive (and hence the corresponding sets A are recursive) by Exercise 2.3.4 on page 131. Theorem 2.7.2, Item (ii), cannot be improved to the conditional "$C(\chi_{1:n}|n) \geq \log n$ for all n" by Exercise 2.7.3 on page 181.

Example 2.7.2 Diophantine equations are algebraic equations of the form $X = 0$, where X is built up from nonnegative integer variables and nonnegative integer constants by a finite number of additions $(A + B)$ and multiplications $(A \times B)$. The best-known examples are $x^n + y^n = z^n$, where $n = 1, 2, \ldots$.

Pierre de Fermat (1601–1665) stated that this equation has no solution in positive integers x, y, and z for n an integer greater than 2. (For $n = 2$ there exist solutions, for instance $3^2 + 4^2 = 5^2$.) However, he did not supply a proof of this assertion, often called *Fermat's last theorem*. After 350 years of withstanding concerted attempts to come up with a proof or disproof, the problem had become a celebrity among unsolved mathematical problems. However, A. Wiles [*Ann. of Math.*, 141:3(1995), 443–551] has finaly settled the problem by proving Fermat's last theorem. Let us for the moment disregard Wiles's proof and reason naively. Suppose we substitute all possible values for x, y, z with $x + y + z \leq n$, for $n = 3, 4, \ldots$. In this way, we recursively enumerate all solutions of Fermat's equation. Hence, such a process will eventually give a counterexample to Fermat's conjecture *if one exists*, but the process will never yield conclusive evidence if the conjecture happens to be true.

In his famous address to the International Congress of Mathematicians in 1900, D. Hilbert proposed twenty-three mathematical problems as a program to direct the mathematical efforts in the twentieth century. The tenth problem asks for an algorithm that given an arbitrary Diophantine equation, produces either an integer solution for this equation or indicates that no such solution exists. After a great deal of preliminary work by other mathematicians, the Russian mathematician Yu.V. Matijasevich finally showed that no such algorithm exists. Suppose we weaken the problem as follows. First, effectively enumerate all Diophantine equations, and consider the characteristic sequence $\Delta = \Delta_1\Delta_2\ldots$, defined by $\Delta_i = 1$ if the ith Diophantine equation is solvable, and 0 otherwise. Then $C(\Delta_{1:n}) \leq n + O(1)$. But the theorem above shows that $C(\Delta_{1:n}|n) \leq \log n + c$, for some fixed constant c. The nonrandomness of the characteristic sequence means that the solvability of Diophantine equations is highly interdependent—it is impossible for a random sequence of them to be solvable and the remainder unsolvable. ◇

Example 2.7.3 In the proof of Theorem 2.7.2, Item (ii), we used a set recursively isomorphic to the halting set $K_0 = \{\langle x, y\rangle : \phi_x(y) < \infty\}$. In fact, almost every recursively enumerable set that is complete under the usual reducibilities would have done. We can use this to obtain natural examples for incompressible finite strings. Let d be the number of elements in K_0 that are less than 2^n. Then by running all the computations of all $\phi_x(y)$ with $z < 2^n, z = \langle x, y\rangle$, in parallel until d of them have halted, we effectively find all computations among them that halt. That is, if χ is the characteristic sequence of K_0, then we can effectively compute $\chi_{1:m}$ (where $m = 2^n$) from d. The program p for this computation has length $l(p) \leq l(d) + c \leq n + c$, for some fixed constant c independent of n. By Theorem 2.7.2, Item (ii), the shortest program from which we can compute $\chi_{1:m}$ has length at least n. Hence, there is another constant c such that p is c-incompressible. ◇

Exercises

2.7.1. [10] Show that there exists a constant c such that $C(0^n|n) \leq c$ for all n, and $C(0^n) \geq \log n - c$ for infinitely many n.

2.7.2. • [26] Let $A \subseteq \mathcal{N}$ be a recursively enumerable set, and let $\chi = \chi_1\chi_2\ldots$ be its characteristic sequence. We use the uniform complexity $C(\chi_{1:n}; n)$ of Exercise 2.3.2 on page 130.

(a) Show that $C(\chi_{1:n}; n) \leq \log n + O(1)$ for all A and n.

(b) Show that there exists an A such that $C(\chi_{1:n}; n) \geq \log n$ for all n.

Comments. This implies Theorem 2.7.2. It is the original Barzdins's lemma. Source: J.M. Barzdins, *Soviet Math. Dokl.*, 9(1968), 1251–1254.

2.7.3. [27] Is there a symmetric form of Theorem 2.7.2 (Barzdins's lemma) using only conditional complexities? The answer is negative. Show that there is no recursively enumerable set such that its characteristic sequence χ satisfies $C(\chi_{1:n}|n) \geq \log n + O(1)$ for all n.

Comments. Hint: let χ be the characteristic sequence of a recursively enumerable set A. Consider $C(\chi_{1:f(n)}|f(n))$, with $\chi_{1:f(n)}$ containing exactly 2^{2^n} ones. Then, $\log f(n) \geq 2^n$. But using the partial recursive function enumerating A, we can compute $\chi_{1:f(n)}$, given $f(n)$, from just the value of n. Hence, we have a program of $\log n + O(1)$ bits for $\chi_{1:f(n)}$. Compare this to Barzdins's lemma (Theorem 2.7.2) and Exercise 2.7.2. Source: R.M. Solovay, sci.logic electronic newsgroup, 24 November 1989.

2.7.4. [34] Is there a symmetric form of Theorem 2.7.2 (Barzdins's lemma) using only unconditional complexities? The answer is negative. Show that there is a recursively enumerable set $A \subseteq \mathcal{N}$ and a constant c such that its characteristic sequence χ satisfies $C(\chi_{1:n}) \geq 2\log n - c$ for infinitely many n.

Comments. First note the easy fact that the Kolmogorov complexity of $\chi_{1:n}$ is at most $2\log n + O(1)$ for all n ($\leq \log n$ bits to specify n and $\leq \log n$ bits to specify $k = \sum_{i=1}^{n} \chi_i$). Hint: partition $\mathcal{N}$ into exponentially increasing half-open intervals $I_k = (t_k, 2^{t_k}]$ with $t_0 = 0$. Note that $\log(2^{t_k} - t_k) = 2\log t_{k+1} - 2 - o(1)$ for $k \to \infty$. Use increasingly precise approximations of $C(\chi_{1:n})$ for $n \in I_k$ for increasing k to enumerate A. Source: R.M. Solovay, sci.logic electronic newsgroup, 24 November 1989; posed as open problem [O39] in the first printing of this book; solved by M. Kummer [*SIAM J. Comput.*, 25:6(1996), 1123–1143].

2.7.5. [25] Prove the following strange fact (Kamae's theorem). For every natural number m there is a string x such that for all but finitely many strings y, $C(x) - C(x|y) > m$.

Comments. There exist strings x such that almost all strings y contain a large amount of algorithmic information about x. Hint: x must be such that almost all large numbers contain much information about x. Let c be a large enough fixed constant. Let A be a recursively enumerable set of integers, and let $\alpha_1\alpha_2\ldots$ be the characteristic sequence of A. Set $x = x(k) = \alpha_1\alpha_2\ldots\alpha_h$, where $h = 2^k$. By Barzdins's lemma, Theorem 2.7.2, we can assume $C(x(k)) \geq k$. Enumerate A without repetition as $b_1, b_2, \ldots$. Let $m(k) = \max\{i : b_i \leq 2^k\}$. Then for any integer $y \geq m(k)$ we have $C(x(k)|y) \leq \log k + c$. Namely, using y we can enumerate $b_1, b_2, \ldots, b_t$, and with $\log k$ extra information describing k we can find $x(k)$. Therefore, $C(x) - C(x|y) \geq k - \log k - c$. Source: T. Kamae, *Osaka J. Math.*, 10(1973), 305–307. See also Exercise 2.2.13.

2.7.6. [25] Consider an enumeration $W_1, W_2, \ldots$ of all recursively enumerable sets. A simple set A is *effectively simple* if there is a recursive

function f such that $W_i \subseteq \bar{A}$ implies that $d(W_i) \leq f(i)$ (where $\bar{A}$ is the complement of A). The set $\bar{A}$ is called *effectively immune*.

(a) Show that the set B defined in Theorem 2.7.1 is effectively simple.

(b) Show that the set RAND defined by $\{x : C(x) \geq l(x)\}$ is effectively immune.

(c) Show that Item (a) implies that B is Turing complete for the recursively enumerable sets.

Comments. Hint: the proof of Theorem 2.7.1, Item (iii), showing that B is simple actually shows that B is also effectively simple, which demonstrates Item (a). For Item (c), see for example P. Odifreddi, *Classical Recursion Theory*, North-Holland, 1989.

2.7.7. [32] Let $\phi_1, \phi_2, \ldots$ be the standard enumeration of partial recursive functions. The *diagonal halting set* is $\{x : \phi_x(x) < \infty\}$ (also denoted by K). The *Kolmogorov set* is $\{(x, y) : C(x) \leq y\}$. We assume familiarity with notions in Exercise 1.7.16. To say that a set A is *recursive in* a set B is the same as saying that A is Turing reducible to B.

(a) Show that the diagonal halting set is recursive in the Kolmogorov set.

(b) Show that the Kolmogorov set is recursive in the diagonal halting set.

(c) Show that the Kolmogorov set is Turing-complete for the recursively enumerable sets.

Comments. This means that if we can solve the halting problem, then we can compute C, and conversely. Hint for Item (a): given x we want to know whether $x \in K$, that is, whether $T_x(x)$ halts. Let $l(\langle x, T_x \rangle) = n$. Now use the Kolmogorov set to recursively find the least number t such that for all y with $l(y) = 2n$ and $C(y) < 2n$ the reference universal machine U computes y from some program of length less than $2n$ in at most t steps. Note that t is found with some organized dovetailing. Claim: $T_x(x)$ halts iff $T_x(x)$ halts within t steps (hence we can see whether $T_x(x)$ halts). If not, then we can use $T_x(x)$ as a clock and run the same dovetailing process as above, but now we produce a string of complexity $2n$ via a description of length n. Source: Attributed to P. Gács by W. Gasarch, personal communication February 13, 1992.

2.7.8. [42] We can express the nonrecursivity of $C(x)$ in terms of $C(C(x)|x)$, which measures what we may call the *complexity of the complexity function*. Denote $l(x)$ by n.

(a) Prove the *upper bound* $C(C(x)|x)) \leq \log n + O(1)$.

(b) Prove the following *lower bound*: For each length n there are strings x such that

$$C(C(x)|x) \geq \log n - \log\log n - O(1).$$

Comments. This means that x only marginally helps to compute $C(x)$; most information in $C(x)$ is extra information related to the halting problem. Hint for Item (b): same proof as in Section 3.8. Source: P. Gács, *Soviet Math. Dokl.*, 15(1974), 1477–1480.

2.7.9. [44] Show that every infinite sequence is Turing-reducible (Exercise 1.7.16, page 43, with sets replaced by characteristic sequences of sets) to an infinite sequence that is random with respect to the uniform measure.

Comments. C.H. Bennett raised the question whether every infinite binary sequence can be obtained from an incompressible one by a Turing machine. He proved this for a special case. Philosophically, the result implied in the exercise allows us to view even very pathological sequences as the result of two relatively well understood notions, to wit, the completely chaotic outcome of coin-tossing and a Turing machine transducer algorithm. Source: P. Gács, *Inform. Contr.*, 70(1986), 186–192. See also [W. Merkle, N. Mihailovic, *J. Symb. Logic*, 69(2004), 862–878].

2.7.10. [30] This exercise assumes knowledge of the notion of *Turing degree*, Exercise 1.7.16. Every Turing degree contains a set A such that if χ is the characteristic sequence of A, then $C(\chi_{1:n}|n) \leq \log n$ for all n.

Comments. Hence, a high degree of unsolvability of a set does not imply a high Kolmogorov complexity of the associated characteristic sequence. Hint: call a set B *semirecursive* if there exists a recursive linear ordering $<_B$ of $\mathcal{N}$ such that there exists a lower cut element y such that $B = \{x : x \leq_B y\}$. For any set A there is a semirecursive set B such that $B \equiv_T A$ [C.G. Jockusch, *Trans. AMS* 131(1968), 420–436]. Every semirecursive set B has a characteristic sequence χ of $(\mathcal{N}, <_B)$ such that $C(\chi_{1:n}|n) \leq \log n + c$, by the same proof as Theorem 2.7.2, Item (i). Since $<_B$ is recursive, the same property holds for the usual characteristic sequence of B. Source: W. Gasarch, Letter, August 1988. See also R.P. Daley, *J. Comput. System Sci.*, 9(1974), 151–163; *Math. Systems Theory*, 9(1975), 83–94; *Inform. Contr.*, 44(1980), 236–244.

2.7.11. [42] Use Kolmogorov complexity to prove the existence of Turing degrees of unsolvability (Exercise 1.7.16) between the recursive sets and Turing-complete sets (such as K_0).

Comments. Source: R.P. Daley, *J. Symb. Logic*, 46(1981), 460–474; *Inform. Contr.*, 52(1982), 52–67.

2.7.12. [39] We assume familiarity with the notion of truth-table reducibility. Let χ be the characteristic sequence of a recursively enumerable set A. Here $C(\chi_{1:n}; n)$ is the uniform complexity of Exercise 2.3.2.

(a) Show that A is complete under weak truth-table reducibility iff for some unbounded total recursive function $f(n)$, we have $C(\chi_{1:n}; n) \geq f(n)$.

(b) Show that A is complete under Turing reducibility iff $C(\chi_{1:n}; n) \geq f(n)$ for some unbounded total function f recursive in A.

Comments. For resource-bounded versions of Kolmogorov complexity the situation is quite different. Source: M.I. Kanovich, *Soviet Math. Dokl.*, 10(1969), 700–701; 11(1970), 1224–1228.

2.7.13. [20] Define the *state complexity* $S(x)$ of a finite binary string x as the least n such that there is a Turing machine with n states that started in the standard initial conditions of empty tape and distinguished start state will eventually halt with x on its output tape. All machines considered are of the original model as in Section 1.7. Define $B = \{\langle x, y\rangle : S(x) \leq y\}$.

(a) Prove that B is recursively enumerable but not recursive.

(b) Prove that B is Turing complete (in the sense of Exercise 1.7.16).

Comments. Suppose our Turing machines use an m-letter alphabet. Let $T_m(x)$ denote the complexity of x in terms of the minimal number of internal states of a Turing machine. Then

$$T_m(x) \sim C(x)/\,(m-1)\log C(x).$$

Source: problem by J. Andrews, electronic news, June 24, 1988; solutions by V.R. Pratt, R.M. Solovay, electronic news, June 1988.

2.7.14. [22] Show that the set K_0 used in Lemma 1.7.6 on page 35 is not many-to-one reducible to the set B featured in Corollary 2.7.2 on page 176, while B is many-to-one reducible to K_0.

Comments. Hint: Use Exercise 1.7.16. K_0 is m-complete, while B is simple and hence not m-complete. The set K_0 is of a *higher degree of unsolvability with respect to many-to-one reducibility* than B.

2.7.15. [32] Show that there exists an immune set I (a set without an infinite recursively enumerable subset, for instance the complement of a set B as in Theorem 2.7.1, Item (iii)), such that there is a probabilistic machine that computes the characteristic function of some infinite subset of I.

Comments. Hint: Use Theorems 2.5.4, 2.7.1, and the following framework. A *probabilistic* machine is just like a deterministic machine except

that at some steps there are several actions (instead of a single action) that the machine can perform with given probabilities. For simplicity assume that there are exactly two possible actions, each with probability $\frac{1}{2}$. (That is, at each such choice the machine flips a coin.) That a probabilistic machine computes a function ϕ with probability p means that the machine with input x halts with output $\phi(x)$ with probability p. We usually assume $p > \frac{1}{2}$. It can be shown that a machine with any value p between zero and one can be simulated by a machine with value p close to one. It turns out that ϕ is computable by a probabilistic machine iff ϕ is partial recursive [K. de Leeuw, et al., pp. 183–212 in: *Automata Studies*, C.E. Shannon and J. McCarthy, eds., Princeton Univ. Press, 1956]. This result is often interpreted as showing that probabilistic machines cannot perform tasks that are impossible for deterministic machines. But a task may not consist only in finding an unambiguous value, but may consist in finding some value out of a set of possible values. In this form there are obviously tasks that deterministic machines cannot do that probabilistic machines can do, such as the construction of a nonrecursive sequence or to output the characteristic function of some infinite subset of a fixed immune set. The probabilistic machine computes such a characteristic sequence or set if it outputs the sequence or set with positive probability. Source: A.K. Zvonkin and L.A. Levin, *Russ. Math. Surv.*, 25:6(1970), 83–124, attributed to J.M. Barzdins.

2.7.16. [37] A set H of natural numbers is called *hyperimmune* if there is no total recursive function f such that $f(i) > h_i$ for all i, where h_i is the ith element of H in increasing order. That is, H is immune (Exercise 2.7.15, page 184) but the variety of immunity of H is due to the fact that the function that enumerates H's elements in increasing order of size grows faster than any recursive function. Prove the following:

(a) Every hyperimmune set H contains an infinite subset whose characteristic sequence is not computable by a probabilistic machine.

(b) However, there is a probabilistic machine that computes the characteristic sequence of some hyperimmune set.

Comments. Source: A.K. Zvonkin and L.A. Levin, *Russ. Math. Surv.*, 25:6(1970), 83–124, attribute Item (a) to V.N. Agafonov and L.A. Levin, and Item (b) to N.V. Petri. Hint: Item (a) follows from the fact that if a fixed set is computable by a probabilistic machine then it is recursively enumerable. Theorems 2.2 and 2.3 in P. Gács, [*Theoret. Comput. Sci.*, 22(1983), 71–93], cover related issues.

2.7.17. [19] We define a variant of the busy beaver function $BB(n)$ in Exercise 1.7.19 on page 45. Let $BC(n)$ be the largest natural number m such that $C(m) \leq n$. Let $\phi_1, \phi_2, \ldots$ be the standard effective enumeration of partial recursive functions.

(a) Show that $BC(n) > \phi(n)$, where $\phi = \phi_k$, for all $n \geq C(k) - 4 \log n$.

(b) Show that $BC(n)$ is not a recursive function.

(c) Show that the nonrecursiveness of $BC(n)$ can be used to prove the unsolvability of the halting problem (Lemma 1.7.5 on page 34) and vice versa.

(d) Let F be an axiomatizable sound formal theory that can be described completely (axioms, inference rules, ...) in m bits. Show that no provable true statement in F asserts "$BC(n) = x$" for $BC(n) = x$ with any $n > m + O(1)$.

Comments. Hint for Item (a): $C(\phi(n)) \leq C(k, n)+O(1)$. Then, $C(\phi(n)) \leq n - \log n$. Hence, $BC(n) > \phi(n)$. Hint for Item (b): It grows faster than any recursive function. Hint for Item (c): If the halting problem were solvable, we could compute $BC(n)$ from the outputs of all halting programs of length at most n. Conversely, every halting program p halts within $BC(n)$ steps, for $n \geq l(p) + O(1)$. So recursiveness of BC implies the solvability of the halting problem. This exercise is an application of Theorem 2.3.1, Item (iii). In fact, $BC(n)$ is some sort of inverse function of $m(n)$, the greatest monotonic increasing function bounding $C(n)$ from below. Source: G.J. Chaitin, pp. 108–111 in: *Open Problems in Communication and Computation*, T.M. Cover, B. Gopinath, eds., Springer-Verlag, 1988.

2.8 Algorithmic Information Theory

One interpretation of the complexity $C(x)$ is as the quantity of information needed for the recovery of an object x from scratch. Similarly, the conditional complexity $C(x|y)$ quantifies the information needed to recover x given only y. Hence the complexity is 'absolute information' in an object. Can we obtain similar laws for complexity-based 'absolute information theory' as we did for the probability-based information theory of Section 1.11?

If $C(x|y)$ is much less than $C(x)$, then we may interpret this as an indication that y contains a lot of information about x.

Definition 2.8.1 The *algorithmic information* about y contained in x is defined as

$$I_C(x : y) = C(y) - C(y|x).$$

Choosing reference function ϕ_0 in Theorem 2.1.1 with $\phi_0(x, \epsilon) = x$ yields

$$C(x|x) = 0 \text{ and } I_C(x : x) = C(x).$$

By the additive optimality of ϕ_0, these equations hold up to an additive constant independent of x, for any reference function ϕ_0. In this way we

can view the complexity $C(x)$ as the algorithmic information about itself contained in an object. For applications, this definition of the quantity of information has the advantage that it refers to individual objects, and not to objects treated as elements of a set of objects with a probability distribution given on it, as in Section 1.11.

Does the new definition have the desirable properties that hold for the analogous quantities in classic information theory? We know that equality and inequality can hold only up to additive constants, according to the indeterminacy in the invariance theorem, Theorem 2.1.1. Intuitively, it is reasonable to require that

$$I_C(x : y) \geq 0,$$

up to an additive fixed constant independent of x and y. Formally, this follows easily from the definition of $I_C(x : y)$, by noting that $C(y) \geq C(y|x)$ up to an independent additive constant.

2.8.1 Entropy, Information, and Complexity

The major point we have to address is the relation between the Kolmogorov complexity and Shannon's entropy as defined in Section 1.11. Briefly, classic information theory says that a random variable X distributed according to $P(X = x)$ has entropy (complexity) $H(X) = \sum P(X = x) \log 1/P(X = x)$, where the interpretation is that $H(X)$ bits are on average sufficient to describe an outcome x. Algorithmic complexity says that an object x has complexity, or algorithmic information, $C(x)$ equal to the minimum length of a binary program for x. It is a beautiful fact that these two notions turn out to be much the same. The statement below may be called the theorem of equality between stochastic entropy and expected algorithmic complexity. (The theorem actually gives an inequality, but together with the simple argument in Example 2.8.1 on page 188 this turns into an asymptotic equality.)

Theorem 2.8.1 *Let $x = y_1 y_2 \dots y_m$ be a finite binary string with $l(y_1) = \dots = l(y_m) = n$. Let the frequency of occurrence of the binary representation of $k = 1, 2, 3, \dots, 2^n$ as a y-block be denoted by $p_k = d(\{i : y_i = k\})/m$. Then up to an independent additive constant,*

$$C(x) \leq m(H + \epsilon(m)),$$

with $H = \sum p_k \log 1/p_k$, the sum taken for k from 1 to 2^n, and $\epsilon(m) = 2^{n+1} l(m)/m$. Note that $\epsilon(m) \to 0$ as $m \to \infty$ with n fixed.

Proof. Denote 2^n by N. To reconstruct x it suffices to know the number $s_k = p_k m$ of occurrences of k as a y_i in x, $k = 1, 2, \dots, N$, together with x's serial number j in the ordered set of all strings satisfying these constraints. That is, we can recover x from $s_1, \dots, s_N, j$. Therefore, up

to an independent fixed constant, $C(x) \leq 2l(s_1) + \cdots + 2l(s_N) + l(j)$. By construction,

$$j \leq \binom{m}{s_1, \ldots, s_N},$$

a multinomial coefficient (Exercise 1.3.4 on page 10). Since also each $s_k \leq m$, we find that

$$C(x) \leq 2^{n+1} l(m) + l\binom{m}{s_1, \ldots, s_N}.$$

Writing the multinomial coefficient in factorials, and using Stirling's approximation, Exercise 1.5.4 on page 17, to approximate j, the theorem is proved. □

In Theorem 2.8.1 we have separated the frequency regularities from the remaining regularities. The entropy component mH measures the frequency regularities only, while the remaining component $m\epsilon(m)$ accounts for all remaining factors.

Example 2.8.1 For x representing the sequence of outcomes of independent trials, the inequality in Theorem 2.8.1 can be replaced by asymptotic equality with high probability.

We give a simple example to show the relation between the entropy H of a stochastic source X, emitting n-length binary words with probability $P(X = x)$ of outcome x, and the complexity C. Let $P(X = x) = 2^{-n}$ be the uniform probability distribution on the outcomes of length n. The entropy H in Theorem 2.8.1 is, according to Section 1.11, especially designed to measure frequency regularities. We show that it is asymptotically equal to the expected complexity of a string. By Theorem 2.2.1 almost all x are c-incompressible, that is, there are $2^n(1 - 2^{-c+1})$ many x's that have $C(x) \geq n - c$. A simple computation shows that the entropy $H(X) = \sum_{l(x)=n} P(X = x) \log 1/P(X = x)$ is asymptotically equal to the *expected complexity* $\mathbf{E} = \sum_{l(x)=n} P(X = x)C(x)$ of an n-length word. Namely, we obtain

$$\frac{n}{n + O(1)} \leq \frac{H(X)}{\mathbf{E}} < \frac{n}{(1 - 2^{-c+1})(n - c)}.$$

Substitute $c = \log n$ to obtain

$$\lim_{n \to \infty} \frac{H(X)}{\mathbf{E}} = 1.$$

◇

In Section 4.3.5 we prove the following generalization: Let p be a *recursive* probability distribution on $\mathcal{N}$, that is, there is a Turing machine computing the function p. Let, moreover, $K(x)$ be the prefix complexity as defined in Chapter 3. Then $\log 1/P(x)$ and $K(x)$ are close to each other with high probability. Since $|K(x) - C(x)| \leq 2 \log C(x)$ by Example 3.1.4 on page 203, also $\log 1/P(x)$ and $C(x)$ are close to each other with high probability.

In particular, the entropy $H = \sum_{l(x)=n} P(x) \log 1/P(x)$ of the distribution P is asymptotically equal to the expected complexity $\sum_{l(x)=n} P(x)K(x)$ of words of length n; see Section 8.1.1.

Because we saw a few lines above that $K(x)$ and $C(x)$ are equal up to a logarithmic additive term, the expected plain complexity $C(\cdot)$ is also asymptotically equal to the entropy,

$$\sum_x P(x)C(x) \sim \sum_x P(x) \log \frac{1}{P(x)}.$$

Thus, the intended interpretation of complexity $C(x)$ as a measure of the information content of an individual object x is supported by a tight quantitative relationship to Shannon's probabilistic notion.

2.8.2 Symmetry of Information

Is algorithmic information symmetric? In Section 1.11 we noted that in Shannon's information theory, the mutual information in one random variable about another one is symmetric. While the equation $I_C(x : y) = I_C(y : x)$ cannot be expected to hold exactly, a priori it can be expected to hold up to a constant related to the choice of reference function ϕ_0 in Theorem 2.1.1. However, with the current definitions, information turns out to be symmetric only up to a logarithmic additive term.

Example 2.8.2 By Theorem 2.2.1, there is a binary string x of each length n such that $C(x|n) \geq n$. Similarly, there are infinitely many n such that $C(n) \geq l(n)$. Choosing x such that its length n is random in this sense yields, up to independent constants,

$$I_C(x : n) = C(n) - C(n|x) \geq l(n),$$
$$I_C(n : x) = C(x) - C(x|n) \leq n - n = 0.$$

◇

This example shows that the difference (the asymmetry of algorithmic information) $|I_C(x : y) - I_C(y : x)|$ can be of order the logarithm of the complexities of x and y. However, it cannot be greater, as we proceed to show now. This may be called the theorem of symmetry of algorithmic information for C-complexity. As usual, $C(x, y) = C(\langle x, y \rangle)$ is the length of the least program of U that prints out x and y and a way to tell them apart.

Theorem 2.8.2 *For all $x, y \in \mathcal{N}$, $C(x,y) = C(x) + C(y|x) + O(\log C(x,y))$.*

Since $C(x,y) = C(y,x)$ up to an additive constant term, the following symmetry of information property follows immediately.

Corollary 2.8.1 Up to an additive term $O(\log C(x,y))$,

$$C(x) - C(x|y) = C(y) - C(y|x).$$

Therefore,

$$|I_C(x:y) - I_C(y:x)| = O(\log C(x,y)).$$

Theorem 2.8.2 cannot be improved in general, since in Example 2.8.2 we have seen that the difference $|I_C(x:y) - I_C(y:x)|$ is at least $\log C(x)$ for some nontrivial x and y. The proof of Theorem 2.8.2 follows.

Proof. ($\leq$) We can describe $\langle x,y\rangle$ by giving a description of x, a description of y given x, and an indication of where to separate the two descriptions. If p is a shortest program for x and q is a shortest program for y, with $l(p) \leq l(q)$, then there is a Turing machine for which $\overline{l(p)}pq$ is a program to compute $\langle x,y\rangle$. Invoking the invariance theorem, Theorem 2.1.1, we obtain $C(x,y) \leq C(x) + C(y|x) + 2l(C(x)) + O(1)$.

($\geq$) Recall that the implied constant in the $O(\log C(x,y))$-notation can be both positive and negative. Thus, we need to prove that there is a constant $c \geq 0$ such that $C(x,y) \geq C(x) + C(y|x) - c\log C(x,y)$. Assume to the contrary that for every constant $c \geq 0$, there are x and y such that

$$C(y|x) > C(x,y) - C(x) + cl(C(x,y)). \tag{2.8}$$

Let $A = \{\langle u,z\rangle : C(u,z) \leq C(x,y)\}$. Given $C(x,y)$, the set A can be recursively enumerated. Let $A_x = \{z : C(x,z) \leq C(x,y)\}$. Given $C(x,y)$ and x, we have a simple algorithm to recursively enumerate the set A_x. One can describe y, given x, using its serial number in enumeration order of A_x and $C(x,y)$. Therefore,

$$C(y|x) \leq l(d(A_x)) + 2l(C(x,y)) + O(1). \tag{2.9}$$

By Equations 2.8, 2.9,

$$d(A_x) > 2^e, \quad e = C(x,y) - C(x) + (c-2)l(C(x,y)) - O(1). \tag{2.10}$$

But now we can obtain a too short description for x as follows. Given $C(x,y)$ and e, we can recursively enumerate the strings u that are candidates for x by satisfying

$$A_u = \{z : C(u,z) \leq C(x,y)\}, \tag{2.11}$$
$$2^e < d(A_u).$$

Denote the set of such u by U. Clearly, $x \in U$. Also,

$$\{\langle u, z\rangle : u \in U, z \in A_u\} \subseteq A. \tag{2.12}$$

The number of elements in A cannot exceed the available number of programs that are short enough to satisfy its definition:

$$d(A) \leq 2^{C(x,y)+O(1)}. \tag{2.13}$$

Combining Equations 2.11, 2.12, and 2.13, we obtain

$$d(U) \leq \frac{d(A)}{\min\{d(A_u) : u \in U\}} < \frac{d(A)}{2^e} \leq \frac{2^{C(x,y)+O(1)}}{2^e}.$$

Hence, we can reconstruct x from $C(x, y)$, e, and the serial number of x in enumeration order of U. Therefore,

$$C(x) < 2l(C(x, y)) + 2l(e) + C(x, y) - e + O(1).$$

Substituting e as given in Equation 2.10, this yields a contradiction, $C(x) < C(x)$, for large enough c. □

Exercises

2.8.1. [17] The following equality and inequality seem to suggest that the shortest descriptions of x contain some extra information besides the description of x.

(a) Show that $C(x, C(x)) = C(x) + O(1)$.

(b) Show that $C(x|y, i - C(x|y, i)) \leq C(x|y, i) + O(1)$.

Comments. These (in)equalities are in some sense pathological and may not hold for all reasonable descriptional complexities. However, these phenomena also hold for the prefix complexity K introduced in Chapter 3. Source: P. Gács, *Lecture Notes on Descriptional Complexity and Randomness*, Manuscript, Boston University, 1987.

2.8.2. [27] Let x be a string of length n.

(a) Show that the equality $C(x, C(x)) = C(C(x)|x) + C(x) + O(1)$ can be satisfied only to within an additive term of about $\log n$.

(b) Prove that $C(x, y) = C(x|y) + C(y)$ can hold only to within an additive logarithmic term without using Exercise 2.8.1, Item (a), and Exercise 2.7.8.

Comments. Hint for Item (a): use Exercise 2.8.1, Item (a), and Exercise 2.7.8. Hint for Item (b): additivity is already violated on random strings of random length. Source: P. Gács, *Ibid.*; A.K. Zvonkin and L.A. Levin, *Russ. Math. Surv.*, 25:6(1970), 83–124.

2.8.3. [12] Show that given x, y, and $C(x, y)$, one can compute $C(x)$ and $C(y)$ up to an additive logarithmic term $O(\log C(x, y))$.

Comments. Hint: use symmetry of information and upper semicomputability. Suggested by L. Fortnow.

2.8.4. [28] Let $\omega = \omega_1\omega_2\ldots$ be an infinite binary sequence. The entropy function $H(p)$ is defined by $H(p) = p\log 1/p + (1-p)\log 1/(1-p)$. Let $\lim_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n} \frac{1}{n}\omega_i = p$.

(a) Show that

$$C(\omega_{1:n}|n) \leq nH\left(\frac{1}{n}\sum_{i=1}^{n}\omega_i\right) + \log n + c.$$

(b) Prove the following: If the ω_i's are generated by coin flips with probability p for outcome 1 (a Bernoulli process with probability p), then for all $\epsilon > 0$,

$$\Pr\left\{\omega : \left|\frac{C(\omega_{1:n}|n)}{n} - H(p)\right| > \epsilon\right\} \to 0,$$

as n goes to infinity.

2.8.5. [26] Show that $2C(a, b, c) \leq C(a, b)+C(b, c)+C(c, a)+O(\log n)$.

Comments. For an application relating the 3-dimensional volume of a geometric object in Euclidean space to the 2-dimensional volumes of its projections, see the discussion in Section 6.13 on page 530. Hint: use the symmetry of information, Theorem 2.8.2. Source: D. Hammer and A.K. Shen, *Theor. Comput. Syst.*, 31:1(1998), 1–4.

2.9 History and References

The confluence of ideas leading to Kolmogorov complexity is analyzed in Section 1.8 through Section 1.12. Who did what, where, and when, is exhaustively discussed in Section 1.13. The relevant documents are dated R.J. Solomonoff, 1960/1964, A.N. Kolmogorov, 1965, and G.J. Chaitin, 1969. According to L.A. Levin, Kolmogorov in his talks used to give credit also to A.M. Turing (for the universal Turing machine). The notion of nonoptimal complexity (as a complexity based on shortest descriptions but lacking the invariance theorem) can be attributed, in part, also to A.A. Markov [*Soviet Math. Dokl.*, 5(1964), 922–924] and G.J. Chaitin [*J. ACM*, 13(1966), 547–569], but that is not a very crucial step from Shannon's coding concepts.

The connection between incompressibility and randomness was made explicit by Kolmogorov and later by Chaitin. Theorem 2.2.1 is due to Kolmogorov. The idea to develop an algorithmic theory of information

is due to Kolmogorov, as is the notion of deficiency of randomness. Universal a priori probability (also based on the invariance theorem) is due to Solomonoff. This is treated in more detail in Chapter 4. (Solomonoff did not consider descriptional complexity itself in detail.)

In his 1965 paper, Kolmogorov mentioned the incomputability of $C(x)$ in a somewhat vague form: "[...] the function $C_\phi(x|y)$ cannot be effectively calculated (generally recursive) even if it is known to be finite for all x and y." Also Solomonoff suggests this in his 1964 paper: "it is clear that many of the individual terms of Eq. (1) are not 'effectively computable' in the sense of Turing [... but can be used] as the heuristic basis of various approximations." Related questions were considered by L. Löfgren [*Automata Theory*, E. Caianiello, ed., Academic Press, 1966, 251–268; *Computer and Information Sciences II*, J. Tou, ed., Academic Press, 1967, 165–175]. Theorem 1 in the latter reference demonstrates in general that for *every* universal function ϕ_0, $C_{\phi_0}(x)$ is not recursive in x. (In the invariance theorem we considered only universal functions using a special type of coding.)

Despite the depth of the main idea of Kolmogorov complexity, the technical expression of the basic quantities turned out to be inaccurate in the sense that many important relationships hold only to within an error term such as the logarithm of complexity. For instance, D.W. Loveland introduced n-strings in [*Inform. Contr.*, 15(1969), 510–526; *Proc. ACM 1st Symp. Theory Comput.*, 1969, 61–65] and proved that the length-conditional $C(x_{1:n}|n)$ measure is not monotonic in n, Example 2.2.5, page 119. He proposed a uniform complexity to solve this problem, and relationships between these complexities are the subject of Exercises 2.3.2, 2.3.4, 2.5.10, 2.5.11, 2.5.12, and 2.5.13.

In the subsequent development of this chapter we have used time and again the excellent 1970 survey by L.A. Levin and A.K. Zvonkin [*Russ. Math. Surv.*, 25:6(1970), 83–124], which describes mainly the research performed in the former USSR. We have drawn considerably on and profited from the point of view expressed in P. Gács's [*Komplexität und Zufälligkeit*, Ph.D. thesis, J.W. Goethe Univ., Frankfurt am Main, 1978, unpublished; *Lecture Notes on Descriptional Complexity and Randomness*, Manuscript, Boston University, 1987]. Another source for the Russian school is the survey by V.V. Vyugin, *Selecta Mathematica,* formerly *Sovietica*, 13:4(1994), 357–389 (translated from the Russian *Semiotika and Informatika*, 16(1981), 14–43).

In [A.K. Zvonkin and L.A. Levin, *Russ. Math. Surv.*, 25:6(1970), 83–124], Theorems 2.2.1 through 2.3.2 are attributed to Kolmogorov. The result on meager sets in Section 2.2 is from [M. Sipser, *Lecture Notes on Complexity Theory*, MIT Lab Computer Science, 1985, unpublished]. We avoided calling such sets 'sparse' sets because we need to reserve

the term for sets that contain a polynomial number of elements for each length. The approximation theorem, Theorem 2.3.3, is stated in some form in [R.J. Solomonoff, *Inform. Contr.*, 7(1964), 1–22, 224–254], and is attributed also to Kolmogorov by Levin and Zvonkin. Some other properties of the integer function C we mentioned were observed by H.P. Katseff and M. Sipser [*Theoret. Comput. Sci.*, 15(1981), 291–309].

The material on random strings (sequences) in Sections 2.4 and 2.5 is primarily due to P. Martin-Löf [*Inform. Contr.*, 9(1966), 602–619; *Z. Wahrsch. Verw. Geb.*, 19(1971), 225–230]. The notions of random finite strings and random infinite sequences, complexity oscillations, lower semicomputable (sequential) Martin-Löf tests and the existence of universal (sequential) tests, the use of constructive measure theory, Theorems 2.4.2, and 2.5.1 through 2.5.5, are taken from P. Martin-Löf's papers. Weaker oscillations are mentioned by G.J. Chaitin [*J. ACM*, 16(1969), 145–159]. We also used [A.K. Zvonkin and L.A. Levin, *Russ. Math. Surv.*, 25:6(1970), 83–124; P. Gács, *Lecture Notes on Descriptional Complexity and Randomness*, Manuscript, Boston University, 1987].

As noted in the main text, the complexity oscillations of infinite sequences prevent a clear expression of randomness in terms of complexity. This problem was investigated by L.A. Levin in [A.K. Zvonkin and L.A. Levin, *Russ. Math. Surv.*, 25:6(1970), 83–124] and independently by C.P. Schnorr [*Lect. Notes Math.*, Vol. 218, Springer-Verlag, 1971]. As a part of the wider issue of (pseudo) random number generators and (pseudo) randomness tests, the entire issue of randomness of individual finite and infinite sequences is thoroughly reviewed by D.E. Knuth, *Seminumerical Algorithms*, Addison-Wesley, 1981, pp. 142–169; summary, history, and references: pp. 164–166. The whole matter of randomness of individual finite and infinite sequences of zeros and ones is placed in a wider context of probability theory and stochastics, and is analyzed in [A.N. Kolmogorov and V.A. Uspensky, *Theory Probab. Appl.*, 32(1987), 389–412; V.A. Uspensky, A.L. Semenov and A.K. Shen, *Russ. Math. Surv.*, 45:1(1990), 121–189; V.A. Uspensky, A.L. Semenov, An.A. Muchnik, A.L. Semenov, V.A. Uspensky, *Theoret. Comput. Sci.*, 2:207(1998), 1362–1376]. Developments in the theory, at the crossroads of notions of individual randomness, Kolmogorov complexity, and recursion theory have blossomed in the last decades. Such work has been partially incorporated in the main text, and in the exercises, of Chapters 2 through 4. Detailed treatment is beyond the scope and physical size of this book, and is the subject of more specialized treatment, as in R.G. Downey, D.R. Hirschfeldt, *Algorithmic Randomness and Complexity*, Springer-Verlag, New York, to appear; A.K. Shen, V.A. Uspensky, N.K. Vereshchagin, *Kolmogorov Complexity and Randomness*, Elsevier, Amsterdam, to appear; A. Nies, *Computability and Randomness*, Oxford Univ. Press, to appear.

Section 2.6, which analyzes precisely the relative frequencies of 0's and 1's and k-length blocks in individual infinite and finite sequences in terms of their Kolmogorov complexity, is from [M. Li and P.M.B. Vitányi, *Math. Systems Theory*, 27(1994), 365–376].

The recursion-theoretic properties we treat in Section 2.7 are related to Gödel's famous incompleteness theorem. Theorem 2.7.1 is attributed to J.M. Barzdins in [A.K. Zvonkin and L.A. Levin, *Russ. Math. Surv.*, 25:6(1970), 83–124]. The proof of Corollary 2.7.2 was given by G.J. Chaitin [*J. ACM*, 21(1974), 403–423; *Scientific American*, 232:5(1975), 47–52]. This application and some philosophical consequences have been advocated with considerable eloquence by G.J. Chaitin and C.H. Bennett [C.H. Bennett and M. Gardner, *Scientific American*, 241:5(1979), 20–34].

We also used the insightful discussion in [P. Gács, *Lecture Notes on Descriptional Complexity and Randomness*, Manuscript, Boston University, 1987]. These results are analyzed and critically discussed from a mathematical logic point of view by M. van Lambalgen [*J. Symb. Logic*, 54(1989), 1389–1400]. Theorem 2.7.2, Barzdins's lemma, occurs both in [J.M. Barzdins, *Soviet Math. Dokl.*, 9(1968), 1251–1254] and [D.W. Loveland, *Proc. ACM 1st Symp. Theory Comput.*, 1969, 61–65]. Examples in Section 2.7 are due to Kolmogorov, 1970, published much later as [*Russ. Math. Surv.*, 38:4(1983), 27–36] and a footnote in [L.A. Levin, *Problems Inform. Transmission*, 10:3(1974), 206–210].

The treatment of the relation between plain Kolmogorov complexity and Shannon's entropy in Section 2.8 is based on the work of A.N. Kolmogorov [*Problems Inform. Transmission*, 1:1(1965), 1–7; *IEEE Trans. Inform. Theory*, IT-14(1968), 662–665; *Russ. Math. Surv.*, 38:4(1983), 27–36; *Lect. Notes Math.*, Vol. 1021, Springer-Verlag, 1983, 1–5] and on [A.K. Zvonkin and L.A. Levin, *Russ. Math. Surv.*, 25:6(1970), 83–124]. The latter reference attributes Theorem 2.8.1 to Kolmogorov. Theorem 2.8.2 and its Corollary 2.8.1, establishing the precise error term in the additivity of complexity and symmetry of information as logarithmic in the complexity, are due to Levin and Kolmogorov [A.K. Zvonkin and L.A. Levin, *Russ. Math. Surv.*, 25:6(1970), 83–124; A.N. Kolmogorov, *Russ. Math. Surv.*, 38:4(1983), 27–36].

3

Algorithmic Prefix Complexity

While the development of an algorithmic theory of complexity according to the original definitions (plain Kolmogorov complexity) in Chapter 2 was very fruitful, for certain goals the mathematical framework is not yet satisfactory. This has resulted in a plethora of proposals of modified measures to get rid of one or the other problem. Let us list a few conspicuous inconveniences.

- The plain complexity is *not subadditive*: the inequality $C(x,y) \leq C(x) + C(y)$ holds only to within a term logarithmic in $C(x)$ or $C(y)$—Example 2.1.5 on page 109 and Example 2.2.3 on page 118. An attempt to solve this problem is to use conditional complexity, in which case we indeed find that $C(x, y|C(x)) \leq C(x) + C(y)$ up to an additive constant.
- Another problem is *nonmonotonicity over prefixes*: it would be pleasing if the complexity of xy were never less than the complexity of x. The complexity measure $C(x)$ does not have this property. In contrast to the subadditivity question, here use of conditional complexity $C(x|l(x))$ instead of $C(x)$ does not help. Not only is this measure nonmonotonic, but it has also another counterintuitive property: it drops infinitely often below a fixed constant as x runs through the natural numbers. A first proposal to remedy this defect was the uniform complexity variant $C(x; l(x))$; see Exercise 2.3.2 on page 130. Informally, this measure gives the length of the shortest program p that computes x_i for all inputs i, $i = 1, 2, \ldots, l(x)$.
- In the development of the theory of *random infinite sequences* it would be natural to identify infinite random sequences with infinite sequences of which all finite initial segments are random. As it

M. Li and P.M.B. Vitányi, *An Introduction to Kolmogorov Complexity and Its Applications*,
DOI: 10.1007/978-0-387-49820-1_3, © Springer Science + Business Media, LLC 2008

turned out, no infinite sequence satisfies this criterion, due to complexity oscillations. This holds for any of the $C(x)$, $C(x|l(x))$, and $C(x;l(x))$ measures. It is quite complicated to express Martin-Löf's notion of randomness in terms of the C-complexity of prefixes.

- The original motivation of Solomonoff to introduce algorithmic complexity was as a device through which to assign a universal *prior probability* to each finite binary string (Section 1.6 and Section 1.10). Choosing the reference Turing machine U as in the invariance theorem, Theorem 2.1.1, this induces a function P over $\mathcal{N}$ (equivalently $\{0,1\}^*$) defined by $P(x) = \sum 2^{-l(p)}$, the sum taken over all inputs p for which U computes output x and halts.

  This approach is different from the one in Example 1.1.3 on page 6, where we used $P'(x) = 2^{-l(x^*)}$, where x^* is a shortest program for x. Anticipating Chapter 4, if we allow only self-delimiting programs as developed in the current chapter, the two approaches turn out to be the same according to the later Theorem 4.3.3. Namely, $P(x) = \Theta(P'(x))$ if we allow only Turing machines with self-delimiting programs (no program for which the machine halts is the prefix of another program for which the machine halts).

  Unfortunately, P is not a proper probability mass function, since the series $\sum_x P(x)$ diverges. Worse, for each individual x we have $P(x) = \infty$. Namely, for each $x \in \mathcal{N}$ there is a Turing machine T that computes the constant x for all inputs. If $l(T)$ denotes the length of the description of T, then

  $$P(x) \geq 2^{-l(T)} \sum_{p \in \{0,1\}^*} 2^{-l(p)} = \infty.$$

  Our next try is to redefine $P(x)$ by not considering all programs that output x, but only the shortest program that outputs x. This yields $P(x) = 2^{-C(x)}$. However, we still have $\sum_x P(x)$ diverging, so again P is not proper. This holds also with respect to $C(x|l(x))$ and $C(x;l(x))$, the simple reason being that since all of these measures are close to $\log x$ for almost all x, the corresponding $P(x)$ will be close to $1/x$ for almost all x. Divergence of $\sum_x P(x)$ then follows from divergence of the harmonic series $\sum_x 1/x$.

- There is a close relation between the complexity of a finite string and its Shannon entropy, Section 2.8 and the later Section 8.1.1. Indeed, it would be troublesome if it were not so, since both notions have been introduced as a measure of information content: in the one case of individual objects, and in the other case of random variables. Therefore, we could hope that classical information-theoretic identities as derived in Section 1.11 would have complexity analogues that are satisfied up to an additive constant (reflecting the

choice of reference machine). However, in Section 2.8 we have established that the complexity analogues of some information-theoretic identities are satisfied only to within an additive logarithmic term and that this cannot be improved in general.

Looking at this list of deficiencies with the wisdom of hindsight, it is not too difficult to transcend a decade of strenuous investigation and alternative proposals by proceeding immediately to what now seems a convenient version of algorithmic complexity. This is the complexity induced by Turing machines with a set of programs in which no program is a proper prefix of another program. This proposal gets rid of all problems above (except for nonmonotonicity). Let us consider this matter in some detail.

The study of the algorithmic complexity of descriptions asks in effect for a code in the sense of Section 1.11.1. In his original paper, Shannon restricted attention to those codes for which no value is the prefix of another value, the so-called prefix-codes. This restriction is highly motivated by the implicit assumption that descriptions will be concatenated and must be uniquely decodable.

Recall from Section 1.11.1 that uniquely decodable codes and prefix-codes share the same sets of code-word lengths. Moreover, the minimal average code-word length L of a prefix-code encoding source-word sequences emitted by a random variable with probability distribution P satisfies

$$H(P) \leq L \leq H(P) + 1,$$

where $H(P)$ is the entropy of the random variable (Theorem 1.11.2 on page 77). This is obtained (up to an additive constant term) by assigning code-word lengths $l_i = \lceil \log 1/p_i \rceil$ to the ith outcome of the random variable, where p_i is the probability of that outcome. In this way, $H(P)$ may be interpreted as the minimal expected length of a description of an outcome, using a prefix-free code. Then, the expected additive fudge term in complexity equations based on prefix-codes is $O(1)$ rather than logarithmic. However, for complexity equations about individual objects in general the situation is more complicated.

The divergence of the series $\sum 2^{-C(x)}$ was a major blow to Solomonoff's program. But if the set of programs of the reference machine is prefix-free, then convergence of this series as required by the probability interpretation is a property ensured by Kraft's inequality, Theorem 1.11.1.

At present, prefix-code-based complexity is often considered as some sort of a standard algorithmic complexity. Lest the reader be deluded into the fallacy that this is the most perfect of all possible worlds, we state that different applications turn out to require different versions of complexity, and all of these are natural for their own purposes.

3.1 The Invariance Theorem

In the theory of algorithmic complexity we have a more refined goal than in classic information theory, but not essentially different objectives. Here, too, it is natural to restrict the effective descriptions to uniquely decodable codes, and since our interest is only in the length of code words, to prefix-codes.

Definition 3.1.1 A *partial recursive prefix function* $\phi : \{0,1\}^* \to \mathcal{N}$ is a partial recursive function such that if $\phi(p) < \infty$ and $\phi(q) < \infty$, then p is not a proper prefix of q.

Let $T_1, T_2, \ldots$ be the standard enumeration of Turing machines, and let $\phi_1, \phi_2, \ldots$ be the corresponding enumeration of partial recursive functions (Section 1.7). Obviously, all partial recursive prefix functions occur in this list. We change every Turing machine T computing ϕ to a machine T' computing a partial recursive prefix function ψ, where $\psi = \phi$ if ϕ was already a partial recursive prefix function. The machine T' executes the algorithm below using machine T. In contrast to T, which is presented with an input from $\{0,1\}^*$ delimited by end markers, machine T' is presented with a potentially infinite binary input sequence $b_1 b_2 \ldots$, which it reads from left to right without backing up.

A *halting input*, or *program*, of T' is an initial segment $b_1 b_2 \ldots b_m$ such that T' halts after reading b_m and before reading b_{m+1}. There are no other programs of T'. In this way, no program is a proper prefix of any other program, that is, the set of programs is prefix-free. Moreover, T' determines the end of each program without reading the next symbol. Such programs are called *self-delimiting*; see Definition 1.11.4 on page 79. Each such program p is the code word for the source word $T'(p)$ consisting of the word written on the output tape by the time T' halts its computation with program p.

Definition 3.1.2 The computation of T' on input $b_1 b_2 b_3 \ldots$ is given by the following algorithm that modifies the operation of the original T:

Step 1. Set $p := \epsilon$.

Step 2. Dovetail all computations of T computing $\phi(pq)$, for all $q \in \{0,1\}^*$. **{**Let q_j be the jth string of $\{0,1\}^*$; dovetailing here means executing consecutive stages $i := 1, 2, \ldots$, such that in the ith stage we do one next step of the computation of $\phi(pq_j)$ for all j with $j \leq i$**}**
If $\phi(pq) < \infty$ is the first halting computation **then go to** Step 3.

Step 3. If $q = \epsilon$ **then** output $\phi(p)$ and halt **{**These p are already self-delimiting**} else** read $b :=$ next input bit; set $p := pb$ **{**If this case happens for every initial segment of the input,then the machine never halts**}**; **go to** Step 2.

This construction produces an effective enumeration

$$T'_1, T'_2, \ldots$$

of Turing machines, called *prefix machines*, computing all, and only, partial recursive prefix functions $\psi_1, \psi_2, \ldots$. That is, the Turing machines that did already compute a partial recursive prefix function have an unchanged input–output behavior, although they compute much more slowly than before. Each of the remaining Turing machines is changed into a machine that computes some partial recursive prefix function, and it is irrelevant which one. This brings us to the *invariance theorem for prefix complexity*.

Example 3.1.1 The previous construction results in a version of prefix machines we may call *partial recursive prefix function machines.* An alternative way of defining a prefix machine is as follows. It is a Turing machine with a separate one-way input tape, a separate one-way output tape, and a two-way work tape, all of them one-way infinite. In the start state the input is written on the input tape, and the read-only head on that tape scans the first symbol. Initially, both the work tape and the output tape are empty and the read/write heads are at their leftmost squares. At the start of each step, the state of the finite control and the symbols under scan on the input tape and the work tape determine what the Turing machine does in this step. It does the following: It either moves the reading head on the input tape to the next input symbol or does not move the reading head at all, followed by a computation step consisting in either changing the scanned symbol on the work tape or moving the work tape head one square left, right, or not at all, followed by a change of state of the finite control and either writing a symbol to the next empty square on the output tape or not writing on the output tape.

Let the infinite input tape contain only 0's and 1's (no blanks). Clearly, with this definition every machine T scans one maximal prefix, say p, of any infinite binary input. The output is the contents of the output tape when the machine halts. This type of prefix machine is called a *self-delimiting machine*. We leave it to the reader to show that the self-delimiting machines can be effectively enumerated. Moreover, every partial recursive prefix function is computed by a self-delimiting machine, and every self-delimiting machine computes a partial recursive prefix function. Without restrictions on the computation time, the difference between the two definitions does not matter for us. In particular, the invariance theorem, Theorem 3.1.1 below, is invariant under the two definitions. With time bounds it is not known whether the properties are invariant under the two definitions; see the discussion in Example 7.1.1 on page 535. ◇

Recall Definition 2.0.1, page 103, of a function that is additively optimal (a special type of universality) for a class of functions.

Theorem 3.1.1 *There exists an additively optimal universal partial recursive prefix function ψ_0 such that for every partial recursive prefix function ψ there is a constant c_ψ such that $C_{\psi_0}(x|y) \leq C_\psi(x|y) + c_\psi$, for all $x, y \in \mathcal{N}$.*

Proof. Using standard techniques, like those in Section 1.7, we can show that there exists a universal prefix machine U such that $U(\langle y, \langle n, p\rangle\rangle) = T'_n(y, p)$ for all $y, p \in \mathcal{N}$. The remainder of the proof is analogous to the proof of Theorem 2.1.1, page 105. □

For each pair of additively optimal partial recursive prefixes ψ and ψ',

$$|C_\psi(x|y) - C_{\psi'}(x|y)| \leq c_{\psi,\psi'},$$

for all x and some constant $c_{\psi,\psi'}$. We fix one additively optimal partial recursive prefix function ψ_0 as the standard reference, and U in the proof of Theorem 3.1.1 as the *reference prefix machine*, and define the *prefix complexity* of x, conditional to y, by $K(x|y) = C_{\psi_0}(x|y)$ for all x. The (unconditional) prefix complexity of x is defined as $K(x) = K(x|\epsilon)$.

Example 3.1.2 Define $K(x, y) = K(\langle x, y\rangle)$. Requiring the decoding algorithm to be prefix-free has advantages, since now we can concatenate descriptions without marking where one description ends and the other one begins. Previously, end markers disappeared in concatenation of descriptions, which was responsible for logarithmic fudge terms in our formulas. More formally, in contrast to C (Example 2.2.3 on page 118), the measure K is *subadditive*, that is,

$$K(x, y) \leq K(x) + K(y) + O(1).$$

Namely, let U be the reference machine of Theorem 3.1.1. Let $U(x^*) = x$ with $l(x^*) = K(x)$, and $U(y^*) = y$ with $l(y^*) = K(y)$. Since the set of programs of U is a prefix-code, we can modify U to a machine V that first reads x^* and computes x, then reads y^* and computes y, and finally computes and outputs $\langle x, y\rangle$. If $V = T_n$ in the enumeration of prefix Turing machines, then $U(\langle n, \langle x^*, y^*\rangle\rangle) = \langle x, y\rangle$. A similar argument shows that $K(xy) \leq K(x) + K(y) + O(1)$. ◇

Example 3.1.3 For all x, we have $K(x) \leq C(x) + K(C(x)) + O(1)$, since if p is a shortest program for x (on an ordinary machine) with $l(p) = C(x)$, and q is a shortest program for $l(p)$ (on a prefix machine) with $l(q) = K(l(p))$, then qp is a program for x on some prefix machine. Similarly, one can show that $C(xy) \leq K(x) + C(y) + O(1)$. ◇

Example 3.1.4 The functions C and K are asymptotically equal. For all x and y, we have, up to additive constant terms,

$$C(x|y) \leq K(x|y) \leq C(x|y) + 2\log C(x|y).$$

Since the prefix partial recursive functions are a restriction on the notion of partial recursive functions, it is obvious that $C(x|y) \leq K(x|y)$. To prove the second inequality, we recall that for each x and y the reference machine of the C-complexity of Definition 2.1.2 on page 106 computes output x from some input $\langle y, p\rangle$ with $l(p) = C(x|y)$. We know that $\overline{l(p)}p$ is a self-delimiting encoding for p. Therefore, $K(x|y) \leq C(x|y) + 2l(C(x|y)) + O(1)$. ◇

It is straightforward to extend this idea by having a prefix machine V compute x from input $p_r 0 p_{r-1} 0 \ldots 0 p_0 1$. Here p_{i+1} is the shortest program for the length of p_i, and p_0 is the shortest program to compute x given y. If $V = T_n$, then $U(n, y, p_r 0 p_{r-1} 0 \ldots 0 p_0 1) = x$, and

$$\begin{aligned} K(x|y) &\leq C(x|y) + C(C(x|y)) + \cdots + O(1) + r \\ &\leq C(x|y) + C(C(x|y)) + O(\log C(C(x|y))) + O(1), \end{aligned}$$

where the $\cdots$ indicates all r positive terms, and the $O(1)$ term comprises the cost of encoding n. Using more efficient prefix-codes, we can improve the above estimate. Let l^* be as defined by Equation 1.23; we have

$$C(x|y) \leq K(x|y) \leq C(x|y) + l^*(C(x|y)) + O(1). \tag{3.1}$$

A more precise relation between $C(x)$ and $K(x)$ was shown by R.M. Solovay:

$$\begin{aligned} K(x) &= C(x) + C(C(x)) + O(C(C(C(x)))), \\ C(x) &= K(x) - K(K(x)) - O(K(K(K(x)))). \end{aligned}$$

The content of this observation is that in order to make the minimal C-style program self-delimiting, we must prefix it with a self-delimiting encoding of its own length. To within the cited error term, this simple procedure is always optimal.

We have seen that $C(x)$ is less than $K(x)$, since in the definition of $C(x)$ we did not take the information into account that is needed to make the program prefix-free. We can express C precisely in terms of K.

Lemma 3.1.1 *$C(x)$ is the unique (to within an additive constant) function satisfying*

$$C(x) = \min\{i : K(x|i) \leq i\} + O(1) = K(x|C(x)) + O(1).$$

This equation can be generalized in the natural way to $C(x|y)$.

Proof. ($\geq$) We prove $C(x) \geq K(x|C(x)) + O(1)$. The shortest Turing machine program x^* for x has length $C(x)$. There is a prefix machine that given $C(x)$, computes x from x^*. Clearly, this also implies $C(x) \geq \min\{i : K(x|i) \leq i\} + O(1)$.

($\leq$) We prove $C(x) \leq \min\{i : K(x|i) \leq i\} + O(1)$. This is the same as if $K(x|i) \leq i$, then $C(x) \leq i + O(1)$. Assume that x is computed by reference prefix machine U, given i, from input p, $l(p) \leq i$. A Turing machine (not necessarily prefix machine) T, when presented input $0^{i-l(p)-1}1p$, in case $i - l(p) - 1 > 0$, or input p otherwise, can extract i from the input length and simulate U on p, using the extracted i as the conditional value. By Theorem 2.1.1, page 105, $C(x) \leq C_T(x) \leq i$, up to additive constants. This proves the required inequality. A similar proof, with $C(x)$ substituted for i, proves $C(x) \leq K(x|C(x)) + O(1)$. □

In Theorem 2.1.2 on page 108, we proved the upper bound $C(x) \leq n + O(1)$ for all x of length n. We have $K(x) \leq n + 2\log n + O(1)$ by encoding x as $\overline{l(x)}x$. We give a concrete upper bound on the implied constant of the $O(1)$ term in Section 3.2. We can improve the estimate by more efficient prefix-codes (Section 1.11.1) to

$$K(x) \leq \log^* n + n + l(n) + l(l(n)) + \cdots + O(1), \tag{3.2}$$

where the sum is taken over all positive terms.

Example 3.1.5 The length-conditional plain complexity figured prominently in Chapter 2. Let us look at the prefix complexity version. With $l(x) = n$,

$$\begin{aligned} K(x) &\leq K(x|n) + K(n) + O(1) \\ &\leq K(x|n) + \log^* n + l(n) + l(l(n)) + \cdots + O(1), \end{aligned}$$

and, straightforwardly, $K(x|n) \leq C(x) + O(1) \leq C(x|n) + K(n) + O(1)$. The first inequality holds since the concatenation of a self-delimiting program for $l(x)$ and a self-delimiting program to compute x given $l(x)$ constitutes a self-delimiting program for x. The second inequality follows from Equation 3.2, since $K(n) \leq \log^* n + l(n) + l(l(n)) + \cdots + O(1)$. ◇

Exercises

3.1.1. [12] Use the Kraft inequality, Theorem 1.11.1, to show that $K(x) \geq \log x + \log\log x$ for infinitely many x.

3.1.2. [14] We investigate transformations of complexity under the arithmetic operation $*$. Show that

(a) $K(x * y) \leq K(x) + K(y) + O(1)$;

(b) If x and y are primes, then $K(x * y) = K(x, y) + O(1)$;

(c) $K(x*y)+\log(x*y) \geq K(x,y)+O(1)$;

Comments. Item (c): consider the prime factorization of $z := x*y$. Enumerate all decompositions of z into two factors and let x, y be the jth decomposition. Use j and $x*y$.

3.1.3. [16] Let us backtrack to the original definition of a Turing machine in Section 1.7. Instead of a tape alphabet consisting of 0, 1, and the blank symbol B, we want to consider Turing machines with a tape alphabet consisting only of 0 and 1. We can modify the original effective enumeration $T_1, T_2, \ldots$ in Section 1.7 to an effective enumeration ${T^b}_1, {T^b}_2, \ldots$ of Turing machines using a purely binary tape alphabet, without blanks B. Furthermore, assume that the set of inputs for halting computations of each Turing machine form a uniquely decodable code.

(a) Define $K^b(x) = \min\{l(p) + i + 1 : {T'}_i(p) = x, i \geq 1\}$. Show that $K^b(x) = K(x) + O(1)$.

(b) Show that restricting the Turing machines to a one-letter tape alphabet (with or without delimiting blanks) does not yield a proper complexity measure.

Comments. By reformulating the notion of Turing machine with a purely binary tape alphabet, without distinguished blank symbol B, we naturally end up with the prefix complexity $K(x)$. Hint for Item (a): use the McMillan–Kraft theorem, Exercise 1.11.9 on page 88. Further restrictions of the tape alphabet do not yield proper complexity measures. Hint for Item (b): show that there is no additively optimal partial recursive function in this enumeration of partial recursive functions.

3.1.4. • [19] Do Exercise 2.1.9 on page 114 for K-complexity.

3.1.5. • [31] We can derive $K(x)$ in another way. Define a *complexity* function $F(x)$ to be a function on the natural numbers with the property that the series $\sum 2^{-F(x)} \leq 1$ and such that $F(x)$ is upper semicomputable. That is, the set $\{(m,x) : F(x) \leq m\}$ is recursively enumerable. (If $F(x) = \infty$, then $2^{-F(x)} = 0$.)

(a) Show that the number of x's for which $F(x) \leq m$ is at most 2^m.

(b) Show that there is an additively optimal (minimal) complexity F such that for each complexity F' there is a constant c depending only on F and F' but not on x such that $F(x) \leq F'(x) + c$. This is the invariance theorem corresponding to Theorem 3.1.1.

(c) Show that $F(x) = K(x)+O(1)$, where $F(x)$ is the optimal complexity in Item (b).

Comments. Hint for Item (b): define $F(x) = \min_k\{1, F_k(x) + k\}$, where F_k is the complexity resulting from the kth partial recursive function

after modifying it to satisfy the requirements above. This approach has been proposed by L.A. Levin in a sequence of papers: [*Russ. Math. Surv.*, 25:6(1970), 83–124, *Sov. Math. Dokl.*, 14(1973), 1413–1416, and *Problems Inform. Transmission*, 10(1974), 206–210 (where the equivalence with the prefix machine approach is established)]. See also [P. Gács, *Sov. Math. Dokl.*, 15(1974), 1477–1480]. G.J. Chaitin [*J. Assoc. Comp. Mach.*, 22(1975), 329–340] used the prefix machine approach to define $K(x)$ but gave a different interpretation to the conditional complexity, resulting in the *Kc* version in Example 3.9.2 and Exercise 3.9.3.

3.1.6. [39] Giving a definition of complexity of description, we use recursive decoding functions from the set of finite binary strings to itself. However, we can consider this set with different topologies. If we consider distinct binary strings as incomparable, then this set corresponds to the natural numbers $\mathcal{N}$. If we consider the distinct binary words as compared by the prefix relation, then we view them as nodes in a binary tree $\mathcal{B}^*$, $\mathcal{B} = \{0, 1\}$, in the obvious way. Then there are four possible definitions of recursive decoding mappings: from $\mathcal{N}$ to $\mathcal{N}$, from $\mathcal{N}$ to $\mathcal{B}^*$, from $\mathcal{B}^*$ to $\mathcal{N}$, and from $\mathcal{B}^*$ to $\mathcal{B}^*$. Exploit this idea to formalize the following:

(a) Using recursive decoding from $\mathcal{N}$ to $\mathcal{N}$ we obtain the plain complexity $C(x)$.

(b) Using recursive decoding from $\mathcal{N}$ to $\mathcal{B}^*$ we obtain the uniform complexity $C(x; n)$ as defined in Exercise 2.3.2, page 130.

(c) Using recursive decoding from $\mathcal{B}^*$ to $\mathcal{N}$ we obtain the prefix complexity K.

(d) Using recursive decoding from $\mathcal{B}^*$ to $\mathcal{B}^*$ we obtain the monotone complexity as defined by Levin and Schnorr (Section 4.5.4).

Comments. Source: A.K. Shen, *Sov. Math. Dokl.*, 29:3(1984), 569–574; V.A. Uspensky, pp. 85–101 in: *Kolmogorov Complexity and Computational Complexity*, O. Watanabe, Springer-Verlag, 1992.

3.2 *Sizes of the Constants

The additive constants in the Kolmogorov complexity and prefix complexity of an object depend on the particular choice of universal machine fixed as the reference machine. A minor modification of the universal Turing machine U of Example 1.7.4 on page 30 yields small implied constants—1 instead of $O(1)$. If we modify U to the reference universal machine U' such that $U'(0p) = p$ and $U'(1p) = U(p)$, then

$$C(x) \leq l(x) + 1, \; K(x) \leq l(x) + K(l(x)) + 1.$$

But it is quite tedious to exhibit the explicit encoding of such a machine. On the other hand, if we aim for a 'small' universal Turing machine,

then it becomes difficult to determine the concrete implied constants—which also may become quite large. With this in mind, we want to find a machine model in which a concrete universal machine is easily, and shortly, implemented from weak elementary operations and the implied constants above are small and can be easily determined.

R. Penrose in [*The Emperor's New Mind*, Oxford University press, 1989] encoded a universal Turing machine in 5495 bits. G.J. Chaitin [*Complexity*, 1:4(1995/1996), 55–59] used LISP and powerful elementary operations to define a universal machine to concretely determine prefix complexity and some associated constants. Lambda calculus is simpler and more elegant than LISP. Pure lambda calculus with combinators S and K is elegant, but slow to evaluate. Since we are concerned with simplicity of definition rather than with efficiency of execution, we base a concrete definition of Kolmogorov complexity on combinatory logic. What follows is a very brief exposition on combinatory logic (see [H.P. Barendregt, *The Lambda Calculus, its Syntax and Semantics*, North-Holland, Amsterdam, 1984]), which develops into a discussion of how to represent binary strings in it.

The *primitive combinators* S and K are defined by the rewrite rules

$$Sxyz = xz(yz),$$
$$Kxy = x.$$

Other *combinators* are constructed from these two by application. We assume that application of x on y, written (xy), associates to the left, and omit parentheses accordingly. Thus, $Sxyz$ should be interpreted as $(((Sx)y)z)$. Perhaps surprisingly, S and K suffice to define any function definable in standard lambda calculus. As an example, we can check that $I = SKz$, for every z, is the identity function:

$$SKzx = Kx(zx) = x.$$

The combinator $\infty \equiv SSK(S(SSK))$ is another interesting example—when applied to any combinator, it leads to an infinite rewriting chain (abbreviating $S(SSK)$ to L):

$$\begin{aligned}\infty x &\equiv SSKLx = SL(KL)x = Lx(KLx) = S(SSK)xL \\ &= SSKL(xL) \equiv \infty(xL) = \infty(xLL) = \infty(xLLL)\dots\ .\end{aligned}$$

In our machine model we want to consider each combinator to be a machine, capable of processing input bits and producing output bits. This entails representing an input binary string by a combinator, to which the machine can be applied, and a way of interpreting the result, if possible, as an output binary string.

A rewrite of $Sxyz$ or Kxy will form the basic machine cycle. Our combinator machines employ a *lazy* reduction strategy, in which only the

frontal primitive combinator can be reduced. This is to avoid endless rewrites that might be inconsequential for interpreting the result as a string. Lazy reduction ends when a combinator of the form K, Kx, S, Sx, or Sxy is reached, where we lack the required number of arguments for reducing the front S/K (as in $S(KKK)K$).

3.2.1 Encoding Combinators as Binary Strings

The nice thing about combinators is that they have a wonderfully simple encoding as binary strings that is also self-delimiting: encode S as 00, K as 01, and application as 1. Formally, we define the encoding $\langle C\rangle$ of a combinator C as

$$\begin{aligned}&\langle S\rangle \equiv 00,\\ &\langle K\rangle \equiv 01,\\ &\langle C_0C_1\rangle \equiv 1\langle C_0\rangle\langle C_1\rangle.\end{aligned}$$

For instance, the combinator $S(KSS)$, $(S((KS)S))$ in full, is encoded as 10011010000. The length of this encoding is $3n-1$, where n is the number of primitive combinators (S and K) it contains.

3.2.2 Encoding Booleans, Pairs, and Binary Strings as Combinators

Define

$$\begin{aligned}&0 \equiv K \quad (\texttt{false}),\\ &1 \equiv SK \quad (\texttt{true}).\end{aligned}$$

Then, '**if** B **then** P **else** Q' can be represented by BQP. The simplest way to pair two combinators x and y is to have something that when applied to 0 gives x, and when applied to 1 gives y. What we want then is a combinator, call it P, that satisfies $Pxyz = zxy$, for example,

$$P \equiv S(S(KS)(S(KK)(S(KS)(S(S(KS)(SK))K))))(KK).$$

Lists are then constructed by repeated pairing as usual: $Pl_0(Pl_1(Pl_2\ldots))$ represents the list $[l_0, l_1, l_2, \ldots]$. A special element to signal the end of a list, like the null character signaling the end of a string in the C programming language, will be helpful. Since we consider only lists of bits, we require an element that is easily distinguished from 0 and 1. The simple KS is such an element, since $KS00 = 1$, while $000 = 100 = 0$. An empty list is then a list whose first element is KS, and we define this to be $\$ \equiv K(KS)$. In this way, we can make arbitrary lists of 0's and 1's. For instance, the string $s = 0111001$ maps to the list $P0(P1(P1(P1(P0(P0(P1\$))))))$, conveniently—and a little confusingly, since s itself doesn't represent a combinator—denoted by $(s\$)$. One can check that $(s\$)0$ gives the first bit 0 of s, while $(s\$)1$ gives the tail $P1(P1(P1(P0(P0(P1\$))))) \equiv (111001\$)$.

3.2.3 Output Conventions

If the machine halts, then how do we interpret the result as a binary string? It is unreasonable to expect the result to be exactly identical to $(s\$)$ for some string s, due to the lazy reduction. Instead, we should focus on the individual bits $s_0 s_1 \dots s_{n-1}$ making up a string s. For a combinator C, let $C11\dots10$ with i 1's be denoted by C^i, that is, the ith element of the list C. We say that combinator C produces output $s = s_0 \dots s_{n-1}$ iff $C^i x_0 x_1$ reduces to x_{s_i} $(= s_i x_0 x_1)$ for variables x_0 and x_1 and $i < n$, while $C^n x_0 x_1$ reduces to Sx_1 $(= \$0x_0x_1)$. In this way a single reduction distinguishes among the three possibilities.

Of course, not all combinators produce proper output in this sense. For instance, $S^0x_0x_1 \equiv S0x_0x_1 = 0x_1(x_0x_1) = x_1$ and $S^1x_0x_1 \equiv S10x_0x_1 = 1x_0(0x_0)x_1 = 0x_0x_1 = x_0$, but $S^2x_0x_1 \equiv S110x_0x_1 = 10(10)x_0x_1 = 10x_0x_1 = x_0x_1$. So the primitive combinator S can be interpreted as a list starting with 2 bits, but the remainder cannot be interpreted as either starting with a bit or behaving like the empty list. Such computations must then be considered invalid. Having defined now how a combinator can be seen as a machine taking binary strings as input and producing binary strings as output, we return to our main objective.

3.2.4 The Universal Combinator

A machine in our model is a combinator that is applied to the input combinator—the list that the input string maps to—in order to produce a proper output. For machines that are supposed to be able to detect the end of their input, an input string s is mapped to the list $(s\$)$.

For *prefix machines*, which need to decide when to stop reading more input based on what they have read so far, we can present them lists of the form $(s\infty)$ for increasingly longer s until they halt. Recall that ∞ leads to an infinite computation. Therefore, halting of the machine on $(s\infty)$ implies it has not read past string s.

3.2.5 Decoding a Combinator Encoding

We require combinators that given a list (i) produce the combinator encoded on the initial part of that list and (ii) return the remaining part of the list.

The following combinators do just that:

$$Qx = x0(x10SK)(Q(x1)(Q(R(x1)))) \text{ (initial combinator)},$$
$$Rx = x0(x11)(R(R(x1))) \qquad \text{(skip combinator)}.$$

It may look like cheating to define these combinators recursively, but the fixpoint operator $Y \equiv SSK(S(K(SS(S(SSK))))K)$ helps out. Y has the nice property that $Yx = x(Yx)$, allowing us to take $Q \equiv YQ'$ and $R \equiv YR'$, where $Q'xy = y0(y11)(x(x(y1)))$ and $R'xy = y0(y10SK)(x(y1)(x(R(y1))))$.

Putting the pieces together, we form the universal combinator

$$U \equiv SQR,$$



FIGURE 3.1. The 425-bit universal combinator U' in pixels

which has the property that

$$U(\langle C\rangle x) \equiv SQR(\langle C\rangle x) = Q(\langle C\rangle x)(R(\langle C\rangle x)) = Cx.$$

3.2.6 Concrete Complexities

At long last, we can define Kolmogorov complexity and prefix complexity concretely.

The Kolmogorov complexity $C(x)$ of a string x is the length of a minimal string p such that $U(p\$)$ outputs x. The prefix complexity $K(x)$ of a string x is the length of a minimal string p such that $U(p\infty)$ outputs x.

To define Kolmogorov complexity conditional to a string y, we simply, after applying U to the input, apply the result to $(y\$)$ in turn. (It is clear how this approach can be extended to an arbitrary number of conditional arguments.)

The Kolmogorov complexity $C(x|y)$ of a string x conditional to a string y is the length of a minimal string p such that $U(p\$)(y\$)$ outputs x. The prefix complexity $K(x|y)$ of a string x conditional to a string y is the length of a minimal string p such that $U(p\infty)(y\$)$ outputs x. In this way, combinatory logic makes an excellent vehicle for defining Kolmogorov complexity concretely, and we obtain the following:

Theorem 3.2.1

$$C(x) \leq l(x) + 8,\ C(\epsilon) = 2,$$

$$C(x|y) \leq l(x) + 2,$$
$$K(x) \leq 2l(x) + 231,\ K(\epsilon) = 11,$$
$$K(x|l(x)) \leq l(x) + 689,$$
$$K(x) \leq l(x) + 2l(l(x)) + 1066.$$

For instance, to prove the result about $K(x)$, note that a string s can be encoded in a self-delimiting way as $1s_0 1s_1 \ldots 1s_{n-1}0$ of length $2n+1$. The combinator D defined by $Dx = x0\$(P(x10)(D(x11)))$, expressible as 77 primitive combinators, decodes this. The universal combinator U takes 518 bits. A mere 425 bits suffices to encode the currently most parsimonious universal combinator U', nearly 13 times smaller than the—admittedly less optimized—universal Turing machine of Penrose referred to above. Figure 3.1 represents U' graphically by a 17×25 matrix of pixels, where a white pixel denotes 1 and a black pixel denotes 0. The matrix should be read in row-major order.

3.3 Incompressibility

The quantitative relations between K and C show that there must be many incompressible strings with respect to K. For the C-complexity, for each n, the maximal complexity of strings of length n equals n, up to a fixed constant. With K-complexity life is not so simple.

Example 3.3.1 We first observe that since the range of $K(\cdot)$ is the code-word-length set of a prefix-code, and it follows from the Kraft inequality, Theorem 1.11.1, that $\sum 2^{-K(x)} \leq 1$. This implies the following:

If $f(x)$ is a function such that the series $\sum_x 2^{-f(x)}$ diverges, then $K(x) > f(x)$ infinitely often. Otherwise $K(x)$ could not satisfy the Kraft inequality. For instance, choose $f = f_k$ for each fixed k, with $f_k(x) = \log x + \log\log x + \cdots + \log\log\cdots\log x$, the last term consisting of the k-fold iterated logarithm.

Let f be a function such that the sum $\sum_x 2^{-f(x)}$ is at most 1. By the Kraft inequality, the set $\{f(x) : x \in \mathcal{N}\}$ is the length set of the code-word alphabet of a prefix-code. If, moreover, this prefix-code is decoded by a prefix partial recursive function, then by Theorem 3.1.1, we have $K(x) \leq f(x) + O(1)$. An easy example is $f(x) = \log(x) + 2\log\log(x)$. $\diamond$

We cannot even give a recursive upper bound $f(n)$ on the maximum of $K(x)$ for strings of length n that is always sharp to within a fixed constant. Nonetheless, we can give a precise expression for the maximal complexity of a string of length n. As we can expect, almost all x have nearly maximal complexity. This is expressed in Theorem 3.3.1, which we may regard as a prefix complexity analogue of Theorem 2.2.1 on page 117. Item (ii) gives the distribution of description lengths.

Theorem 3.3.1 (i) *For each n,* $\max\{K(x) : l(x) = n\} = n + K(n) + O(1)$.

(ii) *For each fixed constant r, the number of x of length n with $K(x) \leq n + K(n) - r$ does not exceed $2^{n-r+O(1)}$.*

Proof. (i) ($\leq$) Let U be the reference prefix machine of Theorem 3.1.1. Consider a prefix machine T that on input qx, where $U(q) = l(x)$, computes $T(qx) = x$. Let $T_1, T_2, \ldots$ be the standard enumeration of prefix machines. Since T is a prefix machine we have $T = T_m$ for some m. Then, $U(\bar{m}qx) = x$. Hence, $K(x) \leq l(x) + K(l(x)) + 2l(m)$ with m independent of x.

($\geq$) Since there are 2^n strings x of length n, this follows from Item (ii).

(ii) Assume that a string x of length n satisfies

$$K(x) \leq n + K(n) - r.$$

We now use a remarkable equality, whose proof is omitted at this point:

$$K(x) + K(n|x, K(x)) = K(n) + K(x|n, K(n)) + O(1).$$

(The proof uses Theorem 3.9.1—rather its corollary, Theorem 3.9.2 on page 249—which in turn depends on Theorem 4.3.4 on page 275. We did not find a satisfactory way to avoid dependence on later material.) Substitute $K(n|x, K(x)) = O(1)$, since $n = l(x)$, to obtain

$$K(x|n, K(n)) \leq n - r + O(1). \tag{3.3}$$

By simple counting, there are fewer than $2^{n-r+O(1)}$ strings x of length n satisfying Equation 3.3, which is the required result. □

By almost the same proof, the analogue of Theorem 3.3.1 holds for the conditional complexity $K(x|y)$. The notion of c-incompressible strings can be formulated for K-complexity. But as the upper bound in Theorem 3.3.1 suggests, this is more subtle than for C-complexity.

Example 3.3.2 Define $K^+(x) = \max\{K(y) : l(y) = l(x)\}$. By Theorem 3.3.1, the great majority of all x's of length n cannot be compressed. In fact, for the majority of x's the complexity $K(x)$ significantly exceeds $l(x)$. The maximal randomness achievable by a string x of length n is $K(x) = n + K(n) + O(1)$. It is already natural to call x incompressible if the length of a shortest program for x exceeds x's length. ◇

Definition 3.3.1 Let x be a string of length n. If $K(x) \geq n$, then x is *incompressible*.

Example 3.3.3 Consider the strings that are shortest programs themselves. Are they compressible with respect to K-complexity? Denote a *shortest program* for x by x^*. Clearly, $K(x^*) \leq l(x^*) + O(1)$, since a modification of the reference machine U changes it into an identity machine V using the same prefix-free set of programs as does U. The inequality follows if we simulate V by U. Moreover, $K(x^*) \geq l(x^*) + O(1)$, since otherwise x^* is not a shortest program for x, by the usual argument. Consequently, the prefix complexity of a shortest program does not rise above its length:

$$K(x^*) = l(x^*) + O(1).$$

By Theorem 3.3.1, therefore, the number of shortest programs occurring in the set of all strings of length n does not exceed $2^{n-K(n)+O(1)}$. That is, the fraction of shortest programs among the strings of length n is at most $2^{-K(n)+O(1)}$, which goes to zero as n rises unboundedly. In fact, it can be shown that the number of short programs is small, and that the number of shortest programs for every string x is $O(1)$ (Exercise 4.3.6, page 287). ◇

Exercises

3.3.1. [27] Let $K^+(x)$ be defined as in Example 3.3.2 on page 212. We know that $K^+(x) = n + K(n) + O(1)$.

(a) Show that there are infinitely many n, m, x of length n and y of length m with $n < m$ such that $K^+(x) > n + \log n + \log\log n$ and $K^+(y) < m + \log\log m$.

(b) Show that $K^+(x) = \max\{K(y) : y \leq x\} + O(1)$, where $\leq$ is with respect to the integer interpretation of x, y. This implies that $K^+(x)$ is monotonic nondecreasing with increasing x, up to an $O(1)$ additive term.

(c) Show that there is a constant $d > 0$ such that for all n, there are at least $2^n/d$ strings x of length n such that $K(x) = K^+(x)$.

Comment. Item (b) gives an alternative definition of K^+. Note that the monotonicity of K^+ in Item (b) only appears to be at odds with the seemingly fluctuating behavior in item (a). Compare Item (c) with a similar result for plain Kolmogorov complexity, Exercise 2.2.6 on page 122. Source: G.J. Chaitin, *Appl. Math. Comput.*, 59(1993), 97–100.

3.3.2. [22] Show that neither $K(x)$ nor $K(x|l(x))$ is invariant with respect to cyclic shifts. For example, $K(x_{1:n}) = K(x_{m+1:n}x_{1:m}) + O(1)$ is not satisfied for all m, $1 \leq m \leq n$.

Comments. Hint: choose $x = 10\ldots0$, $l(x) = 2^k$.

3.3.3. [32] Show that Kamae's result, Exercise 2.7.5 on page 181, does not hold for $K(x|y)$.

Comments. Source: P. Gács, *Lecture Notes on Descriptional Complexity and Randomness*, Manuscript, Boston University, 1987.

3.3.4. [17] Show that $K(x) \leq K(x|n, K(n)) + K(n) + O(1)$, for $n = l(x)$.

Comments. Hint: $K(x) \leq K(x, n)+O(1)$. It is easy to see that $K(x, n) \leq K(x|n, K(n)) + K(n) + O(1)$. Source: G.J. Chaitin, *J. Assoc. Comp. Mach.*, 22(1975), 329–340.

3.3.5. [15] Show that with $n = l(x)$, we have $C(x|n) \leq K(x) \leq C(x|n) + l^*(C(x|n)) + l^*(n) + O(1)$.

Comments. Hint: this is an easy consequence of Equation 3.1. Source: S.K. Leung-Yan-Cheong and T.M. Cover, *IEEE Trans. Inform. Theory*, IT-24(1978), 331–339.

3.3.6. [15] Let $\phi(x, y)$ be a recursive function.

(a) Show that $K(\phi(x, y)) \leq K(x) + K(y) + c_\phi$, where c_ϕ is a constant depending only on ϕ.

(b) Show that (a) does not hold for C-complexity.

Comments. Hint: in Item (b) use the fact that the logarithmic error term in Theorem 2.8.2, page 190, cannot be improved.

3.3.7. [11] Show that $K(x) \leq C(x) + C(C(x)) + O(\log C(C(x)))$.

3.3.8. [17] The following equality and inequality seem to suggest that the shortest descriptions of x contain some extra information besides the description of x.

(a) Show that $K(x, K(x)) = K(x) + O(1)$.

(b) Show that $K(x|y, i - K(x|y, i)) \leq K(x|y, i)$.

Comments. These (in)equalities are in some sense pathological. But they hold also for $C(\cdot)$. Source: P. Gács, *Ibid.*

3.3.9. [29] Let $f(i, x)$ be the number of strings y such that $K(y|x) \leq i$. Show that $\log f(i, x) = i - K(i|x) \pm O(1)$.

3.3.10. • [41] (a) Show that $d(\{x : l(x) = n, K(x) < n - K(n) - r\}) \leq 2^{n-r-K(r|n^*)+O(1)}$.

(b) Show that there is a constant c such that if string x of length n ends in at least $r + K(r|n^*) + c$ zeros then $K(x) < n + K(n) - r$.

(c) Show that $d(\{x : l(x) = n, K(x) < n-K(n)-r\}) \geq \lfloor 2^{n-r-K(r|n^*)-O(1)} \rfloor$.

Comments. As usual, n^* denotes the shortest program for n, and if there is more than one then the first one in standard enumeration. This

improves the counting of the distribution of description lengths in Theorem 3.3.1, Item (ii), to a tight bound, up to a multiplicative constant. Hint: for Item (c) use Item (b). The right-hand side of Item (c) equals 0 for a negative exponent. Source: J.S. Miller, L. Yu [*Oscillations in the initial segment complexity of random reals*, Manuscript, 2007 (with as prequel *Trans. Amer. Math. Soc.*, 360:6(2008), 3193–3210)].

3.3.11. • [46] How are K and C precisely related? Is K just a pumped-up C-version? The following formulas relate C and K:

(a) Show that $K(x) = C(x) + C(C(x)) + O(C(C(C(x))))$.

(b) Show that $C(x) = K(x) - K(K(x)) - O(K(K(K(x))))$.

(c) Show that $C(C(x)) - K(K(x)) = O(K(K(K(x))))$.

(d) Show that $K(K(K(x))) \sim C(C(C(x)))$.

Comments. Clearly, we can replace $C(C(x))$ by $K(C(x))$ in the equality of Item (a). Granted Items (c) and (d), it follows that Items (a) and (b) are equivalent. These formulas express the number of extra bits needed to convert a minimal-length program for the reference machine for C-complexity into a minimal-length program for the reference machine of K-complexity. Source is R.M. Solovay, *Lecture Notes*, 1975, unpublished, and quoted in R.M. Solovay, *Non-Classical Logics, Model Theory and Computability*, A.I. Aruda, N.C.A. da Costa and R. Chaqui, eds., North-Holland, 1977, 283–307.

3.3.12. [32] Show that there is a constant c such that for every d we have that if $K(x) > l(x) + K(l(x)) - (d - c)$ then $C(x) > l(x) - 2d$.

Comments. This shows that if we can compress the K-complexity of some x less than $d-c$ below $l(x)+K(l(x))$, then we can compress the C-complexity less than $2d$ below $l(x)$. Here $l(x)+K(l(x)) = K^+(x)-O(1)$ in Example 3.3.2 on page 212, and $l(x) = C^+(x)-O(1)$ of Exercise 3.4.1 on page 217. Source: A. Nies, *Computability and Randomness*, Oxford Univ. Press, to appear.

3.3.13. • [34] Show that C and K do not agree, to within any given additive constant, on which strings are more complex. Formally, show that for every positive integer c, there are strings x, y such that both $C(x) - C(y) \geq c$ and $K(y) - K(x) \geq c$.

Comments. Source: attributed to An.A. Muchnik in [An.A. Muchnik, S.Y. Positselsky, *Theor. Comput. Sci.*, 271:1-2(2002), 15–35]. It follows without too much difficulty from a theorem of R.M. Solovay, *Ibid.*, that maximal plain complexity does not imply maximal prefix complexity in Exercise 3.5.1 on page 220.

3.3.14. [24] Let $\phi : \{0,1\}^* \to \mathcal{N}$ be a prefix algorithm, that is, a partial recursive function with a prefix-free domain. Then the *extension*

complexity of x with respect to ϕ is defined by $E_\phi(x) = \min\{l(p) : x \text{ is a prefix of } \phi(p)\}$, or $E_\phi(x) = \infty$ if there is no such p.

(a) Show that there is an additively optimal prefix algorithm ϕ_0 such that for any other prefix algorithm ϕ there is a constant c such that $E_{\phi_0}(x) \leq E_\phi(x) + c$ for all x. Select one such ϕ_0 as reference and set the *extension complexity* $E(x) = E_{\phi_0}(x)$. Similarly, we can define the conditional extension complexity $E(x|y)$.

(b) Show that for the relation between the extension complexity E and the prefix complexity $K(x)$, with $n = l(x)$,

$$E(x) \leq K(x) \leq E(x) + K(n) + O(1) \leq E(x) + l^*(n) + O(1).$$

Comments. Source: S.K. Leung-Yan-Cheong, T.M. Cover, *IEEE Trans. Inform. Theory*, IT-24(1978), 331–339.

3.4 K as an Integer Function

Consider K as an integer function $K : \mathcal{N} \to \mathcal{N}$ and determine its behavior. Most of the properties stated for C and $C(\cdot|l(\cdot))$ hold for K and $K(\cdot|l(\cdot))$. We look at the items in the same order, and indicate the similarities and differences.

All of Theorem 2.3.1, page 126, holds with K substituted for C. In particular, therefore, $K(x)$ is unbounded, and it (its limit inferior) goes to infinity more slowly than any unbounded partial recursive function. Although the conditional complexity $K(\cdot|l(\cdot))$ is unbounded, there is no unbounded monotonic lower bound on it. That is, $K(\cdot|l(\cdot))$ drops below a fixed constant infinitely often, Figure 3.2. This is just like the case of $C(\cdot|l(\cdot))$ and has a similar proof.

In fact, even the least monotonic upper bounds on the length-conditional complexities are similar. $C(x|l(x))$ reaches upper bound $\log(x) + O(1)$ for infinitely many x just as $C(x)$ does. But while $K(x)$ exceeds $\log x + \log\log x$ infinitely often by Theorem 3.3.1, clearly $K(x|l(x)) \leq l(x) + O(1)$ for all x by Exercise 3.4.2 on page 217.

Both Theorem 2.3.2, page 127, and Theorem 2.3.3, page 127, hold with C replaced by K. That is, K is not partial recursive and can be computed only for finitely many x. However, K is an upper semicomputable function, that is, the set $\{(m, x) : K(x) \leq m\}$ is recursively enumerable. The same properties hold for the conditional complexity $K(x|y)$.

There are small differences with respect to the four properties discussed in Section 2.3. The function K is *continuous* in the sense that $|K(x) - K(x \pm h)| \leq K(h) + O(1)$. The function $K(x)$ mostly hugs $\log^* x + l(x) + l(l(x)) + \cdots + O(1)$ in the sense that this is a good upper bound. The function $K(x)$ has many *fluctuations* for the same reasons as $C(x)$.

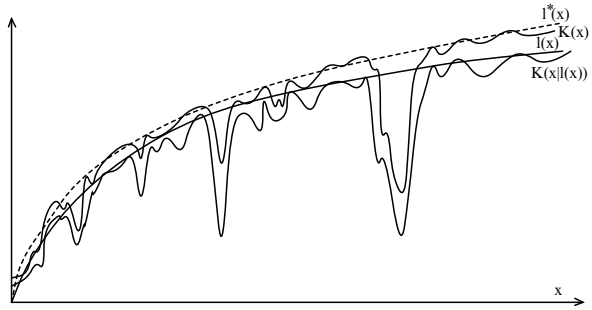


FIGURE 3.2. The graphs of $K(x)$ and $K(x|l(x))$

For each c there is a bound on the length of a run of consecutive c-incompressible numbers. However, the length of runs of c-incompressible numbers rises unboundedly with c.

Exercises

3.4.1. [10] Let $C^+(x) := \max\{C(y) : l(y) = l(x)\}$, and $K^+(x) := \max\{K(y) : l(y) = l(x)\}$ as in Example 3.3.2 on page 212.

(a) Show that $C^+(x) = \log x + O(1)$.

(b) Show that $K^+(x) = \log x + K(\lceil \log x \rceil) + O(1)$.

3.4.2. [20] Analyze the integer function $K(x|n)$ with $n = l(x)$.

(a) Show that there is a constant c such that there are infinitely many x such that $K(x|n) \leq c$.

(b) Let $h = n - C(x|n)$. Show that $K(x|n) \leq C(x|n) + K(h|n) + O(1)$.

(c) Use Item (b) to show that $K(x|n) \leq n + O(1)$ for all x.

(d) Show that $K(x|n) \leq C(x|n) + \log n + O(1)$.

3.4.3. [12] Show that $\sum_x 2^{-K(x|l(x))}$ does not converge.

3.4.4. [36] Use the notation of Exercise 3.4.1. Show that if $K(x) = K^+(x)$, then $C(x) = C^+(x)$.

Comments. Source: R.M. Solovay, *Lecture Notes*, UCLA, 1975, unpublished.

3.4.5. [39] (a) Show that $l^*(x) = \log x + \log\log x + \cdots$ (all positive terms) satisfies $\sum_x 2^{-l^*(x)} < \infty$.

(b) Show that for all x we have $K^+(x) \leq l^*(x) + O(1)$ (K^+ as in Exercise 3.4.1).

(c) Show that for most x we have $K^+(x) = l^*(x) + O(1)$.

Comments. Hint for Item (b): use Item (a). Source: attributed to T.M. Cover [P. Gács, *Lecture Notes on Descriptional Complexity and Randomness*, Manuscript, Boston University, 1987].

3.5 Random Finite Sequences

Recall that c-incompressible strings with respect to C coincide with c'-random strings under the uniform distribution. Since a C-incompressible string can be equated with its shortest program, Example 3.3.3 shows that for appropriate constants c and c', the c'-incompressible strings with respect to K-complexity are a superset of the c-incompressible strings with respect to C-complexity. But the following lemma shows that the converse does not hold: there are incompressible strings with respect to K-complexity that are not $O(1)$-random with respect to the uniform distribution.

Lemma 3.5.1 *For infinitely many n there are strings x of length n that have $K(x) \geq n$ and $C(x) \leq n - \log n$.*

Proof. Consider infinite sequences ω. By Corollary 2.5.1, for each ω there are infinitely many n such that $C(\omega_{1:n}) < n - \log n$. On the other hand, in Theorem 3.6.1 we will show that ω is (Martin-Löf) random with respect to the uniform measure iff $K(\omega_{1:n}) \geq n + O(1)$ for all n. Hence, the n-length prefixes of any random ω, for the sequence of values n denoting points where complexity oscillations reach $\log n$ below n, is an example demonstrating the lemma. □

Corollary 3.5.1 For each c, there are finite strings x that are incompressible according to prefix complexity ($K(x) \geq l(x)$) and that are not c-random finite strings with respect to the uniform distribution.

Example 3.5.1 How much can $K(x)$ exceed $C(x)$ in effective terms? For most x we have

$$K(x) = C(x) + K(C(x)) + O(1).$$

Namely, on the one hand, for all x we have $K(x) \leq C(x) + K(C(x)) + O(1)$ by Example 3.1.3 on page 202. On the other hand, by Theorem 3.3.1, for most x we have $K(x) \geq C(x) + K(C(x)) + O(1)$. Hence, we can conclude that the difference between K and C satisfies

$$K(x) - C(x) \leq \log^*(n) + l(n) + l(l(n)) + \cdots + O(1),$$

for all x of length n, and that the inequality is nearly sharp for infinitely many x, namely, for those x with $C(x) = n + O(1)$ and $K(n) \geq l(n) + l(l(n))$. ◇

Denoting $l(x)$ by n, Theorem 2.4.2 on page 139 identifies

$$\delta_0(x|L) = n - C(x|n) - 1$$

as a universal Martin-Löf test for the uniform distribution L. Corollary 3.5.1 shows that we cannot simply express randomness of a finite string under the uniform distribution in terms of its incompressibility in the sense of prefix complexity.

Substituting the length-conditional version of Lemma 3.1.1,

$$C(x|n) = K(x|C(x), n) + O(1),$$

we obtain an expression that involves both K and C. This can be avoided, in a somewhat contrived manner, by introducing an auxiliary complexity based on K. Define

$$\overline{K}(x;k) = \min\{i : K(x|k-i) \leq i\}.$$

Similar to Lemma 3.1.1 we obtain $\overline{K}(x;k) = K(x|k - \overline{K}(x;k))$. Define the conditional auxiliary complexity in a similar way and denote it by $\overline{K}(x;k|y)$. Using the same arguments as above, we obtain

$$\overline{K}(x;k|k) = C(x|k) + O(1).$$

Then we can express the randomness deficiency found by a universal P-test in terms of the auxiliary complexity. For P a recursive function, it can be proven that

$$\delta_0(x|P) = \log \frac{1}{P(x)} - \overline{K}\left(x; \log \frac{1}{P(x)} \middle| n\right)$$

is a universal P-test. If P is the uniform distribution on strings of length n, then $\log 1/P(x) = n$ for all x of length n, and

$$\delta_0(x|L) = n - \overline{K}(x;n|n) = n - C(x|n) + O(1)$$

is the familiar universal test for the uniform distribution L. Thus, while Martin-Löf's universal test for randomness of *finite* strings under the uniform distribution is easily expressed in terms of C-complexity, Section 2.4, we have to go through quite some contortions to express it in terms of (a variant of) K-complexity. These contortions are necessary, since it is only the form using K-complexity that is generalizable to universal P-tests for arbitrary computable distributions P.

The extended theory of P-tests, and exact expressions for universal P-tests, are treated in Section 4.3.5. These constructions show that the presence of C in Martin-Löf's expression $n - C(x|n)$ is a lucky freak of the uniform distribution. See P. Gács, *Komplexität and Zufälligkeit*, Ph.D. thesis, Mathematics Department, J.W. Goethe Universität, Frankfurt am Main, 1978. In contrast, while in Section 2.5 we did not succeed at all in expressing Martin-Löf's universal sequential test for randomness of *infinite* sequences under the uniform distribution in terms of $C(x)$, in Section 3.6 this will turn out to be straightforward in terms of $K(x)$.

Exercises

3.5.1. [43] Recall the universal Martin-Löf test for randomness of finite strings x of length n under the uniform distribution: $\delta_0(x|L) = n - C(x) + O(1)$. This was based on Theorems 2.1.2, 2.2.1 on pages 108, 117. The analogous facts of Theorem 3.3.1 for $K(x)$ suggest a test $\theta_0(x|L) = n + K(n) - K(x) + O(1)$, with the same intuitive interpretation.

(a) Show that $\theta_0(x|L) \geq \delta_0(x|L) + O(\log \delta_0(x|L) + 1)$. Hence, if θ_0 is small, then δ_0 is small (and this is the sense in which strings that are K-random are C-random).

(b) Construct an infinite sequence of finite strings x_m with the following properties: (i) $l(x_m) \to \infty$ for $m \to \infty$; (ii) $C(x_m) = l(x_m) + O(1)$; and $\lim_{m\to\infty} \theta_0(x_m|L)/\log^2 l(x_m) = 1$. In this sense, C-randomness does not entail K-randomness.

(c) Show that $\limsup_{n\to\infty}(K(n) - C(n) - K(K(n)))/K(K(K(n))) \leq 1$. Use Item (b) to improve this to an equality. The same method provides a counterexample to various improvements in the error terms of the formulas relating $C(x)$ and $K(x)$ in Exercise 3.3.11.

(d) Use Item (c) to show that the seductive equality $K(x) = C(x) + K(C(x)) + O(1)$ is *false* in general (although it obviously holds for all x that are random enough).

Comments. Hint for Item (c): use Use Exercise 3.3.11. Source is R.M. Solovay, *Lecture Notes*, UCLA, 1975, unpublished.

3.6 *Random Infinite Sequences

Let ω be an infinite binary sequence. We are interested in the behavior of $K(\omega_{1:n})$ as a function of n. The complexity $K(\omega_{1:n})$ will turn out to be nonmonotonic in n, just like C-complexity. We will find that an infinite sequence ω is random under the uniform measure iff a simple condition on $K(\omega_{1:n})$ is satisfied. For the C-complexity we were unable to find such a condition in Section 2.5.

Example 3.6.1 We show that $K(0^n)$ as a function of n is not monotonic. Choose n with $K(n) \geq l(n)$. Let $m = 2^k \geq n$ with k minimal. Then 0^n is a prefix of 0^m. Firstly, $K(0^n) \geq \log n + O(1)$. Secondly, $K(m) = K(k) + O(1) \leq 2\log\log n + O(1)$. Therefore, $K(0^n)$ is exponential in $K(0^m)$. $\diamond$

Theorems 2.5.1, 2.5.4 on pages 143, 152 show, for almost all infinite binary sequences, that the oscillations of the C-complexity of initial segments of almost all sequences, as a function of their length, are confined to a thin wedge below the identity function $n + O(1)$. Both random sequences and some nonrandom sequences oscillate in this wedge. For K-complexity the wedge

$$n + O(1) < K(\omega_{1:n}) \leq n + K(n) + O(1)$$

contains all and only infinite random sequences. (The upper bound holds for all infinite sequences by Theorem 3.3.1; the lower bound for random sequences is proved in Schnorr's Theorem 3.6.1 on page 222.) The complexity oscillations are in some form still there, but K exceeds C by so much that the complexity of the prefix does not drop below the length of the prefix itself (for random infinite ω).

3.6.1 Explicit Universal Randomness Tests

The idea that random infinite sequences are those sequences such that the complexity of the initial n-length segments is at most a fixed additive constant below n, for all n, is one of the first-rate ideas in the area of Kolmogorov complexity. In fact, this was one of the motivations for Kolmogorov to invent Kolmogorov complexity in the first place; see the discussion in Section 1.9 on page 56. We have seen in Section 2.5 that this does not work for the plain Kolmogorov complexity $C(\cdot)$, due to the complexity oscillations. The next result is important, and is a culmination of the theory. For prefix complexity $K(\cdot)$ it is indeed the case that random sequences are those sequences for which the complexity of each initial segment is at least its length.

A.N. Kolmogorov suggested the above relation between the complexity of initial segments and randomness of infinite sequences [A.N. Kolmogorov, *IEEE Trans. Inform. Theory*, IT-14:5(1968), 662–664]. This approach being incorrect using $C(\cdot)$ complexity, P. Martin-Löf [*Inform. Contr.*, 9(1966), 602–619] developed the theory as put forth in Section 2.5. Nevertheless, Kolmogorov did not abandon the general outlines of his original idea of connecting randomness of infinite sequences with complexity; see pp. 405–406 of [V.A. Uspensky, *J. Symb. Logic*, 57:2(1992), 385–412].

C.P. Schnorr [*J. Comput. System Sci.*, 7(1973), 376–388] for the uniform distribution, and L.A. Levin [*Sov. Math. Dokl.*, 14(1973), 1413–1416] for arbitrary computable distributions, introduced simultaneously and independently similar but unequal versions of complexity to characterize randomness. These versions are 'process complexity' and the 'monotone' variant of complexity $Km(x)$ (Definition 4.5.9 on page 305). They gave the first realizations of Kolmogorov's idea by showing Corollary 4.5.3 on page 318, that an infinite sequence ω is random iff $|Km(\omega_{1:n}) - n| = O(1)$ (Levin) and a similar statement for process complexity (Schnorr).

G.J. Chaitin [*J. Assoc. Comp. Mach.*, 22(1975), 329–340] proposed calling an infinite sequence ω random if $K(\omega_{1:n}) \geq n - O(1)$ for all n. Happily, this proposal characterizes once again precisely those infinite binary sequences that are random in Martin-Löf's sense (without proof attributed to C.P. Schnorr, 1974, in Chaitin's paper). This important result, now known as Schnorr's theorem, Theorem 3.6.1, was widely circulated, but the first *published proof* appears, perhaps, only as Corollary 3.2 in [V.V. Vyugin, *Semiotika i Informatika*, 16(1981), 14–43, in Russian]. See for historical notes [A.N. Kolmogorov and V.A. Uspensky, *SIAM J. Theory Probab. Appl.*, 32(1987), 387–412].

Another equivalent proposal in terms of constructive measure theory was given by R.M. Solovay, Exercise 2.5.9, page 159. The fact that such different effective formalizations of infinite random sequences turn out to define the same

mathematical object constitutes evidence that our intuitive notion of infinite sequences that are effectively random coincides with the precise notion of Martin-Löf random infinite sequences.

Theorem 3.6.1 *An infinite binary sequence ω is random with respect to the uniform measure iff there is a constant c such that for all n, $K(\omega_{1:n}) \geq n - c$.*

Proof. (Only if) Assume that ω is a random infinite sequence. Then, for each sequential test δ, we have $\delta(\omega) < \infty$, Section 2.5. We construct a particular sequential test, say δ, such that if $\delta(\omega) < \infty$, then there is a constant c such that $K(\omega_{1:n}) \geq n - c$, for all n.

Recall the usual preliminaries. If y is a finite string, then Γ_y denotes the set of infinite sequences ω that start with y. Let λ denote the uniform measure, so that with $l(y) = n$ we have $\lambda(\Gamma_y) = 2^{-n}$. Geometrically speaking, Γ_y corresponds to the half-open interval $[0.y, 0.y + 2^{-n})$.

Define δ as follows. Consider a sequence $A_0 \supset A_1 \supset \cdots$ of sets of finite strings such that

$$A_k = \bigcup_{n \geq 1} \{y : K(y) \leq n - k - c', n = l(y)\},$$

with c' a fixed constant that is large enough to make the remainder of the proof go through. Define a total function γ by $\gamma(y) = \sup_{k \in \mathcal{N}}\{k : y \in A_k\}$. Since K is an upper semicomputable function (it can be approximated from above), γ is a lower semicomputable function (it can be approximated from below). Finally, for each infinite sequence ω define $\delta(\omega) = \sup_{n \in \mathcal{N}}\{\gamma(\omega_{1:n})\}$.

To prove that δ is a sequential test, it suffices to show that $\lambda\{\omega : \delta(\omega) \geq k\} \leq 2^{-k}$, for each $k \geq 0$.

There are fewer than $a(k, n) = 2^{n - K(n) - k}$ strings y of length n of complexity $K(y) \leq n - k - c'$ (Theorem 3.3.1). This bounds the number of strings of length n in A_k. Overestimating $\lambda\{x : \delta(\omega) \geq k\}$, using $a(k, n)$, rearranging terms, and employing Kraft's inequality (Theorem 1.11.1) to argue that $\sum 2^{-K(n)} \leq 1$, we have

$$\begin{aligned}
\lambda\{\omega : \delta(\omega) \geq k\} &\leq \sum_{y \in A_k} \lambda\{\Gamma_y\} \\
&\leq \sum_{n \in \mathcal{N}} a(k, n) 2^{-n} \\
&= 2^{-k} \sum_{n \in \mathcal{N}} 2^{-K(n)} \leq 2^{-k}.
\end{aligned}$$

It is now proved that δ is a sequential test. If ω is random in the sense of Martin-Löf, then $\delta(\omega) < \infty$. That is, for each random ω there is a constant $c < \infty$ such that $K(\omega_{1:n}) \geq n - c$, for all n.

(If) Suppose that ω is not random, that is, there is a sequential test δ such that $\delta(\omega) = \infty$. We show that this implies that $n - K(\omega_{1:n})$ is positively unbounded. Let γ be the defining lower semicomputable function of δ as in Definition 2.5.1. For all $k \geq 1$, define A_k as the maximal prefix-free subset of $\{y : \gamma(y) \geq k\}$. Therefore, $\sum_{y \in A_k} 2^{-l(y)} \leq 1$ by Kraft's inequality. Then

$$L = \{l(y) - k : y \in A_{2k}, k > 1\}$$

satisfies Kraft's inequality

$$\sum_{k>1} \sum_{y \in A_{2k}} 2^{k-l(y)} \leq \sum_{k>1} 2^{-k} \leq 1.$$

This means that L is the length set of a prefix-code. Use γ to enumerate $\bigcup\{A_{2k} : k > 1\}$. Use this enumeration to construct a prefix machine T such that $K_T(y) = l(y) - k$ for all $y \in A_{2k}$, $k > 1$.
We have assumed $\delta(\omega) = \infty$. Hence, for each k, there is an n such that $\omega_{1:n} \in A_{2k}$, which means that $K_T(\omega_{1:n}) \leq n - k$. By Theorem 3.1.1, there is a constant c_T such that $K(\omega_{1:n}) \leq K_T(\omega_{1:n}) + c_T$. Therefore,

$$\limsup_{n \to \infty} (n - K(\omega_{1:n})) = \infty.$$

□

Corollary 3.6.1 The function $\rho_0(\omega|\lambda) = \sup_{n \in \mathcal{N}}\{n - K(\omega_{1:n})\}$ characterizes the random infinite sequences by $\rho_0(\omega|\lambda) < \infty$ iff ω is random with respect to the uniform measure λ. (In terms of our previous notation, $\rho_0(\omega|\lambda) < \infty$ iff $\delta_0(\omega|\lambda) < \infty$, where $\delta_0(\cdot|\lambda)$ is the universal sequential λ-test with λ the uniform measure.)

There are different types of tests that characterize precisely the same class of random infinite sequences. The sequential tests are but one type. The test ρ_0 is an example of an integral test (a universal one) with respect to the uniform measure as defined in Section 4.5.6. The introduction of different types of tests awaits the machinery developed in Chapter 4. The theory of integral tests and martingale tests of infinite sequences is developed in Sections 4.5.6 and 4.5.7, respectively. There we also give exact expressions for tests that separate the random infinite sequences from the nonrandom ones with respect to any recursive measure.

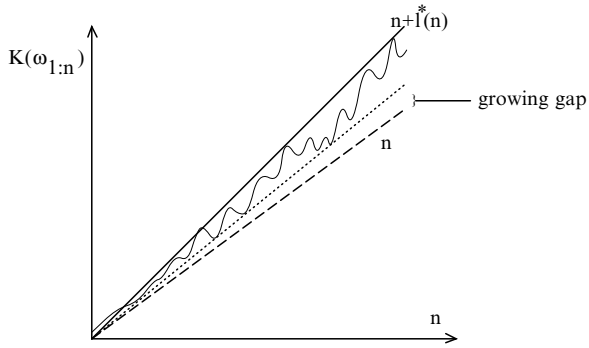


FIGURE 3.3. Complexity oscillations of a typical random sequence ω

Example 3.6.2 The proof of Theorem 3.6.1 shows that the separation of a random infinite sequence from the nonrandom ones involves a complexity gap. That is, for random ω the amount by which $K(\omega_{1:n})$ exceeds n goes to ∞ in the limit; for nonrandom ω the amount by which n exceeds $K(\omega_{1:n})$ may fluctuate but is also unbounded. So the value of $K(\omega_{1:n})$ as a function of n is bounded away from the diagonal in either direction for all ω: nonconstant far below for the nonrandom sequences, and nonconstant far above for the random sequences, Figure 3.3. See also [G.J. Chaitin, *Algorithmic Information Theory*, Cambridge Univ. Press, 1987].

In the C-complexity version, random infinite sequences have oscillations of the complexity of initial segments below the diagonal. With K complexity similar oscillations must take place above the diagonal. Some relevant properties are surveyed in the table of Figure 3.4.

For example, the maximal complexity of some $\omega_{1:n}$ is $K(\omega_{1:n}) = n + K(n) + O(1)$, Theorem 3.3.1. But similar to the case for $C(\omega)$ (Theorem 2.5.1 on page 143), no single ω satisfies $K(\omega_{1:n}) \geq n + K(n) + O(1)$ for all n. In fact, in analogy to the proof of Theorem 2.5.1, we find that for all ω, for infinitely many n,

$$\begin{aligned} K(\omega_{1:n}) &\leq K(C(\omega_{1:n})) + C(\omega_{1:n}) + O(1) \\ &\leq n - g(n) + K(n - g(n)) + O(1) \\ &\leq n - g(n) + K(n) + O(1), \end{aligned}$$

ω IS RANDOM iff	ω IS NOT RANDOM iff
$\rho_0(\omega\|\lambda) < \infty$ iff	$\rho_0(\omega\|\lambda) = \infty$ iff
$\exists c \forall n [K(\omega_{1:n}) \geq n - c]$ iff	$\forall c \exists n [K(\omega_{1:n}) < n - c]$ iff
$\lim_{n\to\infty} K(\omega_{1:n}) - n = \infty$	$\limsup_{n\to\infty} n - K(\omega_{1:n}) = \infty$

FIGURE 3.4. K-complexity criteria for randomness of infinite sequences

where $g(n)$ is as defined in the proof of Theorem 2.5.1. Actually,

$$\limsup_{n\to\infty} K(n) - g(n) = \infty,$$

since the series $\sum_n 2^{-g(n)}$ diverges and the series $\sum_n 2^{-K(n)}$ converges. The infinity on the right-hand side means something very slowly increasing, like $\log^* n$ and $l^*(n)$. Generally, the complexity oscillations of a random sequence ω will look like the graph in Figure 3.3.

In analogy to Theorem 2.5.4 on page 152, we observe the following: By Theorem 2.5.5 on page 153, the set of infinite binary sequences that are random in the sense of Martin-Löf have (uniform) measure one. By Schnorr's theorem, Theorem 3.6.1, this is precisely the set of ω's with $K(\omega_{1:n}) \geq n + O(1)$. To get some insight into the amplitude of the upward oscillations, we note that the set of ω's such that for infinitely many n,

$$K(\omega_{1:n}) \geq n + K(n) + O(1)$$

has uniform measure one, Exercise 3.6.3 on page 229. M. van Lambalgen [*J. Symb. Logic*, 52(1987), 725–755] suggests that far from being a nuisance, the complexity oscillations actually enable us to discern a fine structure in the theory of random sequences (see also Exercise 3.6.11, page 232). For all sequences relatively low in the recursive hierarchy, such as Δ_2^0 definable sequences (for a definition see Exercise 3.6.11 on page 232), the upward oscillations are not maximal, since

$$\lim_{n\to\infty} n + K(n) - K(\omega_{1:n}) = \infty.$$

There are Martin-Löf random sequences that are Δ_2^0 definable, such as Chaitin's halting probability number Ω of Section 3.6.2. The complexity oscillations of this restricted type of random sequences, of which Ω is a typical example, are confined to a narrower wedge, as in Figure 3.5, than the general random sequences of Figure 3.3. The monotone complexity, Chapter 4, was developed to smooth out the oscillations and to characterize randomness. But monotone complexity also obliterates all quantitative differences between Martin-Löf random sequences, and hence does not allow us to distinguish stronger randomness properties. ◇

3.6.2 Halting Probability

It is impossible to construct an infinite random sequence by algorithmic means. But using the reference prefix machine U of Theorem 3.1.1, we can define a particular random infinite binary sequence in a more or less natural way.

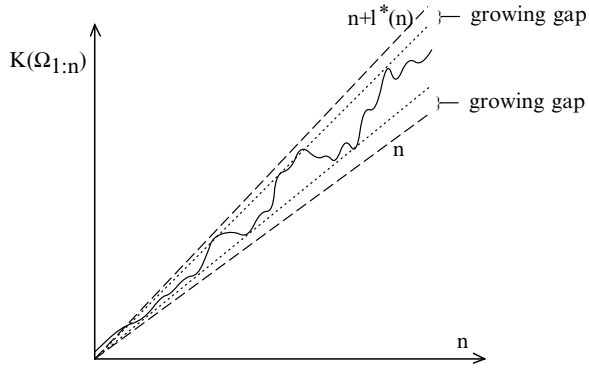


FIGURE 3.5. Complexity oscillations of Ω

Definition 3.6.1 The *halting probability* is the real number Ω defined by

$$\Omega = \sum_{U(p)<\infty} 2^{-l(p)},$$

the sum taken over all inputs p for which the reference machine U halts.

Since U halts for some p, we have $\Omega > 0$. Because U is a prefix machine, the set of its programs forms a prefix-code, and by Kraft's inequality we obtain $\Omega \leq 1$. Actually, $\Omega < 1$, since U does not always halt.

We call Ω the halting probability because it is the probability that U halts if its program is provided by a sequence of fair coin flips. The number Ω has interesting properties. In the first place, the binary representation of the real number Ω encodes the halting problem very compactly. Denote the initial n-length segment of Ω after the decimal point by $\Omega_{1:n}$. If Ω is a terminating binary rational number, then we use the representation with infinitely many zeros, so that $\Omega < \Omega_{1:n} + 2^{-n}$. We shall show that the binary expansion of Ω is an *incompressible* sequence.

Lemma 3.6.1 Let p be a binary string of length at most n. Given $\Omega_{1:n}$, it is decidable whether the reference prefix machine U halts on input p.

Proof. Clearly,

$$\Omega_{1:n} \leq \Omega < \Omega_{1:n} + 2^{-n}. \tag{3.4}$$

Dovetail the computations of U on all inputs as follows: The first phase consists in U executing one step of the computation on the first input. In the second phase, U executes the second step of the computation on the first input and the first step of the computation on the second input.

Phase i consists in U executing the jth step of the computation on the kth input, for all j and k such that $j + k = i$. We start with an approximation $\Omega' := 0$. Execute phases $1, 2, \ldots$. Whenever any computation of U on some input p terminates, we improve our approximation of Ω by executing

$$\Omega' := \Omega' + 2^{-l(p)}.$$

This process eventually yields an approximation Ω' of Ω, such that $\Omega' \geq \Omega_{1:n}$. If p is not among the halted programs that contributed to Ω', then p will never halt. With a new p halting we add a contribution of $2^{-l(p)} \geq 2^{-n}$ to the approximation of Ω, contradicting Equation 3.4 by

$$\Omega \geq \Omega' + 2^{-l(p)} \geq \Omega_{1:n} + 2^{-n}.$$

□

Lemma 3.6.2 There is a constant c such that $K(\Omega_{1:n}) \geq n - c$ for all n.

That is, Ω is a *particular random real*, and one that is naturally defined to boot. That Ω is random implies that it is not computable, and therefore *transcendental*. Namely, if it were computable, then $K(\Omega_{1:n}|n) = O(1)$, which contradicts Claim 3.6.2. By the way, irrationality of Ω implies that both inequalities in Equation 3.4 are strict.

Proof. From Lemma 3.6.1 it follows that given $\Omega_{1:n}$, one can calculate all programs p of length not greater than n for which the reference prefix machine U halts. For every x that is not computed by any of these halting programs, the shortest program x^* has size greater than n, that is, $K(x) > n$. Hence, we can construct a recursive function ϕ computing such high-complexity x's from initial segments of Ω such that for all n,

$$K(\phi(\Omega_{1:n})) > n.$$

Given a description of ϕ in c bits, for each n we can compute $\phi(\Omega_{1:n})$ from $\Omega_{1:n}$, which means that

$$K(\Omega_{1:n}) + c \geq n,$$

which was what we had to prove. □

Corollary 3.6.2 By Schnorr's theorem, Theorem 3.6.1, we find that Ω is random in Martin-Löf's sense with respect to the uniform measure.

It is possible to determine the first couple of bits of the halting probability. For example, with the concrete Kolmogorov complexity fixed in Section 3.2, J.T. Tromp has computed $0.00106502 < \Omega < 0.217643$. This is because many short

strings are either not syntactically correct programs for the concrete universal machine, or they halt quickly, or looping is easily detected. But knowing, say, the first 10,000 bits of Ω enables us to solve the halting of all programs of fewer than 10,000 bits. This includes programs looking for counterexamples to Fermat's last theorem, the Goldbach conjecture, the Riemann hypothesis, and most other conjectures in mathematics that can be refuted by single finite counterexamples. Moreover, for all axiomatic mathematical theories that can be expressed compactly enough to be conceivably interesting to human beings, say in fewer than 10,000 bits, $\Omega_{10,000}$ can be used to decide for every statement in the theory whether it is true, false, or independent. Finally, knowledge of $\Omega_{1:n}$ suffices to determine whether $K(x) \leq n$ for each finite binary string x. Thus, Ω is truly the number of Wisdom, and "can be known of, but not known, through human reason" [C.H. Bennett and M. Gardner, *Scientific American*, 241:11(1979), 20–34]. But even if you possess $\Omega_{1:10,000}$, you cannot use it except by spending time of a thoroughly unrealistic nature. (The time $t(n)$ it takes to find all halting programs of length less than n from $\Omega_{1:n}$ grows faster than any recursive function.)

Exercises

3.6.1. [21] Let A be an infinite recursively enumerable set of natural numbers. Show that if we define $\theta = \sum_{n \in A} 2^{-K(n)}$, then $K(\theta_{1:n}) \geq n - O(1)$ for all n. (Therefore, θ is a random infinite sequence in the sense of Martin-Löf by Schnorr's theorem, Theorem 3.6.1.)

Comments. It follows that θ is not a recursive number. Because θ is not a recursive real number it is irrational and even transcendental. Source: G.J. Chaitin, *Algorithmic Information Theory*, Cambridge University Press, 1987.

3.6.2. [23] Let $1 < r, s < \infty$ be integers. Show that a real number in the unit interval $[0, 1]$ expressed in r-ary expansion (equivalently, infinite sequence over r letters) is Martin-Löf random with respect to the uniform distribution in base r iff it is random expressed in s-ary expansion with respect to the uniform distribution in base s.

Comments. Hint: by Exercise 3.1.4 we have for each pair of integers $r, s \geq 2$ that $|K_r(x) - K_s(x)| \leq c_{r,s}$ for all $x \in \mathcal{N}$ and some constant $c_{r,s}$ independent of x. Use the fact that an infinite binary sequence ω is random with respect to the uniform distribution iff $K(\omega_{1:n}) \geq n - c$ for some constant c and all n, Schnorr's theorem, Theorem 3.6.1. (Note: $K \equiv K_2$.) The argument cannot be too trivial, since a total recursive 1–1 function f can map $\omega = \omega_1\omega_2\ldots$ to $\zeta = \omega_1 0 \omega_2 0 \ldots$. Then $f^{-1}(\zeta) = \omega$. But ζ is not random, even if ω is. Source: C. Calude and H. Jürgenson, pp. 44–66 in: H. Maurer, J. Karhumaki, and G. Rozenberg, eds., *Results and Trends in Theoretical Computer Science*, Springer-Verlag, Berlin, 1994. This result is a special case of the simple and much more general approach of Exercise 4.5.15 on page 329, and is folklore.

3.6.3. [29] We investigate the complexity oscillations of $K(\omega_{1:n})$ for infinite binary sequences ω. If ω is an infinite sequence that is random in the sense of Martin-Löf, then these oscillations take place above the identity line, $K(\omega_{1:n}) \geq n + O(1)$, for all but finitely many n. The maximal possible complexity of $\omega_{1:n}$ is $K(\omega_{1:n}) = n + K(n) + O(1)$, Theorem 3.3.1. But similar to the case for $C(\omega)$, Theorem 2.5.1 on page 143, no ω satisfies the following: there is a constant c such that for all n, we have $K(\omega_{1:n}) \geq n + K(n) - c$.

(a) Show that for every ω there are infinitely many n such that $K(\omega_{1:n}) \leq n + K(n) - g(n) + O(1)$, where $g(n)$ is as defined in the proof of Theorem 2.5.1. Can you generalize this to obtain the analogue of Theorem 2.5.1?

(b) Let the series $\sum_n 2^{-f(n)} < \infty$ converge recursively. Show that for almost all infinite binary sequences ω we have $K(\omega_{1:n}) \geq n + K(n) - f(n)$ for all but finitely many n. This gives a lower bound on the dips of the downward oscillations for almost all ω.

(c) In analogy with Theorem 2.5.4, page 152, we can note the following: By Theorem 2.5.5, page 153, the set of infinite binary sequences that are random in the sense of Martin-Löf have uniform measure one. By Schnorr's theorem, Theorem 3.6.1, these are the ω's, with $K(\omega_{1:n}) \geq n$ for all but finitely many n. Show that the set of ω's with $K(\omega_{1:n}) \geq n + K(n) + O(1)$, for infinitely many n, has uniform measure one. This gives some insight into the amplitude of the upward oscillations. Not all Martin-Löf random sequences achieve this.

(d) Let $f(n)$ be a recursive function such that the series $\sum_n 2^{-f(n)}$ diverges. Show that the set of ω's with $K(\omega_{1:n}) \geq n + f(n) + O(1)$, for infinitely many n, has uniform measure one and contains all infinite binary sequences that are Martin-Löf random.

Comments. Hint for Item (a): $K(\omega_{1:n}) \leq C(\omega_{1:n}) + K(C(\omega_{1:n})) + O(1)$, and use Theorem 2.5.1, page 143. Note that $\limsup_{n\to\infty} K(n) - g(n) = \infty$. This gives an upper bound on the dips of the downward oscillations: the least monotonic upper bound on $K(n) - g(n)$ rises very, very, slowly—more slowly than the k-fold iterated logarithm for any fixed k. Hint for Item (b): $\sum_n 2^{-f(n)-d} \leq 1$ for some nonnegative constant d. By the Kraft inequality, Theorem 1.11.1, there is an effective prefix-code E such that $l(E(n)) = f(n) + d$ for all n. By Theorem 3.1.1, $K(n) \leq l(E(n))$ for every effective prefix-code E, up to a constant depending only on E. Thus, $K(n) \leq f(n) + d$ for a different constant d depending only on f. The result now follows from Section 3.6. Compare Item (d) with the stronger Item (b) of Exercise 3.6.4. Source: van Lambalgen [*Random Sequences*, Ph.D. thesis, University of Amsterdam, 1987; *J. Symb. Logic*, 52(1987), 725–755]. Items (c) and (d) are attributed to R.M. Solovay.

3.6.4. [26/38] Let ω be an infinite binary sequence.

(a) Show that ω is Martin-Löf random iff $\sum_n 2^{n-K(\omega_{1:n})} < \infty$.

(b) Let f be a (possibly nonrecursive) function such that $\sum_n 2^{-f(n)} = \infty$. Assume that ω is random in Martin-Löf's sense. Then, $K(\omega_{1:n}) > n + f(n) - O(1)$ for infinitely many n.

Comments. Item (a) gives a characterization of Martin-Löf randomness. Item (b) is stronger than Item (d) of Exercise 3.6.3. Source: J.S. Miller, L. Yu, *Trans. Amer. Math. Soc.*, 360:6(2008), 3193:3210; The 'only if' side of Item (a) admits also of a simple proof by martingales due to A. Nies. The 'if' side is immediate from Schnorr's theorem, Theorem 3.6.1.

3.6.5. [36] Let f be a function. (a) Show that if the series $\sum_n 2^{-f(n)}$ converges, then there is a Martin-Löf random sequence ω such that $K(\omega_{1:n}) \leq n + f(n) + O(1)$, for all n.

(b) Show that $\sum_n 2^{-f(n)} = \infty$ iff for every Martin-Löf random sequence ω there are infinitely many n such that $K(\omega_{1:n}) > n + f(n)$.

Comments. This gives the extent of the upward oscillations of random sequences, in particular the functions f such that the initial n-segment complexity infinitely often exceeds $n+f(n)$. Source: J.S. Miller, L. Yu, *Oscillations in the initial segment complexity of random reals*, Manuscript, 2007.

3.6.6. [39] We improve on Exercise 3.6.3, Item (b). As usual, n^* denotes the shortest program for n, and if there is more than one, then the first one in standard enumeration. Let f be a function.

(a) Show that if $\sum_n 2^{-f(n)-K(f(n)|n^*)} < \infty$, then $K(\omega_{1:n}) \geq n+K(n)-f(n)$ for all but finitely many n, for almost every $\omega \in \{0,1\}^\infty$.

(b) Show that if $\sum_n 2^{-f(n)-K(f(n)|n^*)} = \infty$, then $K(\omega_{1:n}) < n+K(n)-f(n)$ for infinitely many n, for almost every $\omega \in \{0,1\}^\infty$.

(c) Show that if f is computable and $\sum_n 2^{-f(n)} = \infty$, then $K(\omega_{1:n}) < n + K(n) - f(n)$ for infinitely many n, for every $\omega \in \{0,1\}^\infty$.

(d) Show that there is a function f such that $\sum_n 2^{-f(n)-K(f(n)|n^*)} < \infty$ but $\sum_n 2^{-f(n)} = \infty$.

(e) Show that there is a function f with $\sum_n 2^{-f(n)} = \infty$ but $K(\omega_{1:n}) \geq n+K(n)-f(n)$ for all but finitely many n, for almost every $\omega \in \{0,1\}^\infty$.

(f) If $\sum_n 2^{-f(n)} < \infty$ then there exist infinitely many n such that $K(\omega_{1:n}) < n + f(n)$, for almost every $\omega \in \{0,1\}^\infty$.

Comments. This gives a necessary and sufficient condition on the downward oscillations of a function f to ensure that for almost all ω the complexity $K(\omega_{1:n})$ drops below $n + K(n) - f(n)$ infinitely often. Source: J.S. Miller, L. Yu, *Ibid.*

3.6.7. [43] Let ω be an infinite binary sequence. Show that the following are equivalent to the sequence ω being random in Martin-Löf's sense (with respect to the uniform distribution):

(a) $C(\omega_{1:n}) \geq n - K(n) \pm O(1)$ for every n.

(b) $\gamma_0(\omega_{1:n}|L) = n - C(\omega_{1:n}|n) - K(n) + O(1)$ is finite with L the uniform measure.

(c) For every n and every recursive function g such that $\sum_n 2^{-g(n)} < \infty$, we have $C(\omega_{1:n}) \geq n - g(n) \pm O(1)$, the constant depending on g.

(d) $C(\omega_{1:n}) \geq n - G(n) \pm O(1)$ for every n, and a single, appropriately defined, computable function G.

Comments. In Item (b), the formula for γ_0 expresses concisely and precisely how the complexity oscillations of $C(\omega_{1:n}|n)$ of random infinite sequences behave. This is *Levin's test*—the characterization for random sequences from which Theorem 2.5.4 on page 152 follows. Source for Items (a) and (b): P. Gács, [Ph.D. thesis, Frankfurt am Main, 1978, Theorem 5.4, Corollary 2; *Z. Math. Logik Grundl. Math.*, 26(1980), 385–394]. Source for Items (a), (c), and (d): J.S. Miller, L. Yu,, *Trans. Amer. Math. Soc.*, 360:6(2008), 3193–3210. Another proof of the difficult direction in the last reference is given in [L. Bienvenu, W. Merkle, A.K. Shen, *Fundamentae Informatica*, 83(2008), 1–4].

3.6.8. • [39] Let $\omega = \omega_1\omega_2\ldots$ and $\zeta = \zeta_1\zeta_2\ldots$ be two infinite binary sequences. Let $\omega \oplus \zeta = \eta$ mean that $\eta_{2i} = \omega_i$ and $\eta_{2i+1} = \zeta_i$, for all i.

(a) Show that $\omega \oplus \zeta$ is random in the sense of Martin-Löf iff ζ is random in Martin-Löf's sense and ω is Martin-Löf random in ζ (that is, given ζ as an oracle).

(b) Show that $\omega \oplus \zeta$ is random in Martin-Löf's sense iff $K(\omega_{1:n}) + C(\zeta_{1:n}) \geq 2n - O(1)$.

Comments. Item (a) is the important *van Lambalgen's theorem*. Source: M. van Lambalgen, *Ibid.* Source Item (b): J.S. Miller, L. Yu, *Ibid.*

3.6.9. • [35] Recall that an infinite sequence ω is recursive iff $C(\omega_{1:n}) \leq C(n) + O(1)$, Example 2.3.4. If ω is recursive, then for all n we have $K(\omega_{1:n}|n) = O(1)$ and $K(\omega_{1:n}) \leq K(n) + O(1)$.

(a) Show that if $K(\omega_{1:n}|n) \leq c$, for some c and all n, then ω is recursive.

(b) Show that if $K(\omega_{1:n}) \leq K(n) + c$, for some c and all n, then ω is recursive in $0'$, the Turing degree of the halting problem.

(c) Show that there are nonrecursive ω's satisfying $K(\omega_{1:n}) \leq K(n) + c$, for some c and all n.

(d) Show that for each constant c, there are at most $O(2^c)$ many ω such that for all n, $K(\omega_{1:n}) \leq K(n) + c$.

Comments. The constants c in Items (a) through (c) may depend on ω. Item (b) is attributed to G.J. Chaitin, Item (c) to R.M. Solovay. Source is R.M. Solovay, *Lecture Notes*, UCLA, 1975, unpublished, and personal communication. For Item (d) and other discussion on this topic, see [D. Zambella, *On sequences with simple initial segments*, ITLI Tech. Rept. ML-90-05, Fac. Wiskunde en Informatica, University of Amsterdam, 1990].

3.6.10. • [46] A set $A \subseteq \mathcal{N}$ is *K-trivial* if its characteristic sequence $\chi = \chi_1\chi_2\ldots$ satisfies $K(\chi_{1:n}) \leq K(n)+O(1)$. Certain properties of such sequences were already established in Exercise 3.6.9, Items (b) and (c). Below, we identify sets with their characteristic sequences.

(a) Show that all K-trivial sets are Δ^0_2-definable.

(b) Show that the class of K-trivial sets is closed under $\oplus$, where $\oplus$ is defined for the corresponding characteristic sequences as in Exercise 3.6.8.

(c) Show that a set A is K-trivial iff it is *low for the class of Martin-Löf random sets*, that is, every Martin-Löf random set is already Martin-Löf random relative to Turing machines with A as an oracle. In particular, K-triviality is closed downward under Turing reducibility; see Exercise 1.7.16 on page 43.

(d) Show that a set A is K-trivial iff A is low for K, that is, $K(x) \leq K^A(x) + O(1)$ for every x.

Comments. A great deal of recent recursion-theory research deals with K-triviality, in particular with the nonrecursive K-trivial sequences. They are nonrecursive, but only barely so, and in that sense not completely computably predictable. They are, so to speak, the sequences exhibiting the weakest form of randomness and form the other side of the random-sequence spectrum from the Martin-Löf random sequences. Source for Items (a) and (b): R.G. Downey, D.R. Hirschfeldt, A. Nies, F. Stephan, *Proc. 7th and 8th Asian Logic Confs*, Singapore Univ. Press, 2003, pp. 103–131. Source for Items (c) and (d): A. Nies, *Adv. Math.*, 197:1(2005), 274–305.

3.6.11. [27] Far from being a nuisance, the complexity oscillations actually enable us to discern a fine structure in the theory of random sequences. A sequence ω is Δ^0_2 definable if the set $\{n : \omega_n = 1\}$ is Δ^0_2 definable, Exercise 1.7.21 on page 46. We consider infinite binary sequences that are Δ^0_2 definable (such as the halting probability Ω, Section 3.6.2).

(a) Show that if ω is Δ^0_2 definable, then $\lim_{n\to\infty} n - K(\omega_{1:n}|n) = \infty$. (This is, of course, interesting only for random ω's.)

(b) Show that if ω is Δ_2^0 definable, then $\lim_{n\to\infty} n+K(n)-K(\omega_{1:n}) = \infty$.

(c) Show that if there is a constant c such that for infinitely many n we have $n + K(n) - K(\omega_{1:n}) \leq c$, then ω is not Δ_2^0 definable. That is, if such an ω is random, then it is not a simply definable random sequence.

Comments. Hint for Item (b): use Item (a). Items (a) and (b) delimit the upswing of the complexity oscillations for Δ_2^0 definable ω. Hint for Item (c): use Exercise 3.6.3. The Δ_2^0 definable random sequences are rather atypical random sequences. (An example is the halting probability Ω.) The K-complexity allows us to distinguish between easily definable random sequences and those that are not so easily definable. Source: M. van Lambalgen [*Random Sequences*, Ph.D. thesis, University of Amsterdam, 1987; *J. Symb. Logic*, 54(1989), 1389–1400].

3.6.12. [31] Let ω be an infinite binary sequence. Show that if there exists a constant c such that $K(\omega_{1:n}) \geq n - c$ for all n, then for all k we have $K(\omega_{1:n}) - n \geq k$ from some n onward.

Comments. Hint: This follows easily from Exercise 3.6.4 Item (a), and can be seen as a strengthening of that result. Not only is the limit of $K(\omega_{1:n})-n$ infinite when ω is Martin-Löf random, but it goes to infinity fast enough to make the series in Exercise 3.6.4, Item (a), diverge. In Schnorr's theorem, Theorem 3.6.1, we showed that an infinite binary sequence is random with respect to the uniform measure iff $K(\omega_{1:n}) \geq n-c$ for some c and all but finitely many n. There is not only a sharp dividing line but a wide complexity gap that separates the random infinite sequences from the nonrandom infinite sequences. With respect to the uniform measure, we have in Exercise 2.5.9 defined the notion of a *Solovay test* for randomness, and shown that a sequence ω is Martin-Löf random iff it is Solovay random. Call ω *weakly Chaitin* random if there is a constant c such that for all n, we have $K(\omega_{1:n}) \geq n - c$. The results referred to and this exercise show that the sets of infinite sequences defined by all of these definitions coincide precisely. That different points of departure attempting to formalize an intuitive notion (like randomness) turn out to define the same objects is commonly viewed as evidence that the formalization correctly represents our intuition. Source: G.J. Chaitin, *Algorithmic Information Theory*, Cambridge Univ. Press, 1987; attributed to C.P. Schnorr.

3.6.13. [38] Let ω be an infinite binary sequence that is random with respect to the uniform measure. Let g be a recursive function such that the series $\sum_n 2^{-g(n)}$ diverges, for example $g(n) = \log n$. Let h be a recursive function that is monotone and unbounded, such as $h(n) = \log\log n$. Show that for infinitely many n we have $K(n) \geq g(n)$ and $K(\omega_{1:n}) \leq n + h(n)$.

Comments. Note that this does not imply that the greatest monotonic nondecreasing lower bound on $K(\Omega_{1:n}) - n$ is unbounded. Nonetheless,

this is true. Source: R.M. Solovay, *Lecture Notes*, UCLA, 1975, unpublished.

3.6.14. [29] Let $\omega = \omega_1\omega_2\ldots$ be an infinite binary sequence and let $c(\omega)$ be the smallest c for which $K(\omega_{1:n}) \geq n - c$. Let ω be Martin-Löf random with respect to the uniform distribution. Let $S(n) = \sum_{i=1}^{n} \omega_i$.

(a) Show that given $\epsilon > 0$, we can compute an $n(c, \epsilon)$ such that

$$\left| \frac{S_n}{n} - \frac{1}{2} \right| < \epsilon,$$

for every $n > n(c, \epsilon)$.

(b) Show that for given $\lambda > 1$, we can compute an $n(c, \lambda)$ such that

$$S_n \leq \frac{n}{2} + \lambda\sqrt{\frac{n}{2}\ln\ln n},$$

for every $n > n(c, \lambda)$.

(c) Show that for given $\lambda < 1$, we can compute an $n(c, \lambda)$ such that

$$S_n > \frac{n}{2} + \lambda\sqrt{\frac{n}{2}\ln\ln n},$$

for some $n \leq n(c, \lambda)$.

Comments. Source: Suggested by J.T. Tromp. This is the law of the iterated logarithm (Exercise 1.10.5 on page 65).

3.6.15. [41] Consider an infinite binary sequence ω as a real number $r = 0.\omega$. Recall that ω is lower semicomputable if there is a total recursive function $f : \mathcal{N} \to \mathcal{Q}$ such that $f(i+1) \geq f(i)$ and $\lim_{i\to\infty} f(i) = r$. We call ω an Ω-*like real number* if (i) there is a lower semicomputable function g such that $g(n) = 0.\omega_{1:n}$ for every n; and (ii) $K(\omega_{1:n}) = K(\Omega_{1:n}) + O(1)$. A real number ω is *arithmetically random* (with the uniform measure understood) if every arithmetic property of ω (viewed as an infinite sequence of zeros and ones) holds for some set of reals of uniform measure 1. In other words, ω is arithmetically random iff ω does not belong to any arithmetic set of reals of uniform measure zero. Since the arithmetic sets are Borel sets, and the union of countably many Borel sets of uniform measure zero is again a Borel set of uniform measure zero, the set of arithmetically random reals has uniform measure one. Let $m(n)$ be the greatest monotonic lower bound on $K(n)$, that is, $m(n) = \min\{K(x) : x \geq n\}$. By the same argument as in the proof of Theorem 2.3.2, page 127, for any recursive function $\phi(n)$ that goes to infinity monotonically with n, we have $m(n) < \phi(n)$ from some n onward.

(a) Show that Ω is Ω-like.

(b) Show that Ω-like reals are Martin-Löf random.

(c) Show that arithmetically random reals are Martin-Löf random.

(d) Show that since Ω-like reals are Δ^0_2 definable, they are not even 2-random in the sense of Exercise 3.6.19, so the set of Ω-like reals has uniform measure zero.

(e) Show that if ω is arithmetically random, then there is a constant c (depending on ω) such that for infinitely many n we have $K(\omega_{1:n}) \geq n + K(n) - c$.

(f) Show that the 'then' property in Item (e) holds for ω's that are 2-random in the sense of Exercise 3.6.19.

(g) Show that if ω is Ω-like then $K(\omega_{1:n}) \leq n + K(n) - m(n) + O(\log m(n))$.

(h) Show that if ω is arithmetically random, then $K(\omega_{1:n}) \geq n + m(n) + O(\log m(n))$.

Comments. Hint for Item (g): compute $\Omega_{m(n)}$ effectively from n by doing n steps in the computation of Ω. R.M. Solovay credits this idea to C.P. Schnorr. Source is R.M. Solovay, *Lecture Notes*, UCLA, 1975, unpublished. Source for Item (f): J.S. Miller, *The K-degrees, low for K degrees, and weakly low for K sets*, Manuscript, 2007.

3.6.16. • [37] Consider the family $\mathcal{U}$ of universal prefix machines U satisfying the conditions of the proof of Theorem 3.1.1 on page 202. Every such U has an associated halting probability $\Omega_U = \sum_{U(p)<\infty} 2^{-l(p)}$. Everything in Exercise 3.6.15 is invariant under the choice of reference universal prefix machine among these U's. Hence, the class of Ω-like reals consists precisely of those reals that satisfy the definition in that exercise with $\Omega = \Omega_U$ for some $U \in \mathcal{U}$.

(a) Show that if ω is Ω-like, then $\omega = \Omega_U$ for some $U \in \mathcal{U}$.

(b) Show that the set of Martin-Löf random binary sequences (equivalently, Martin-Löf random reals) that are lower semicomputable equals the set $\{\Omega_U : U \in \mathcal{U}\}$.

Comments. This provides a first characterization of natural examples of Martin-Löf random infinite sequences (or reals) that are lower semicomputable. These are random, hence incomputable, but only barely so. Source Item (a): C. Calude, P. Hertling, B. Khoussainov, Y. Wang: *Theor. Comput. Sci.*, 255:1-2(2001), 125–149. Item (b) characterizes both randomness among the lower semicomputable reals and lower semicomputability among the random reals. Trivially, every Ω_U is lower semicomputable, and it is Martin-Löf random by Corollary 3.6.2 on page 227. This shows inclusion in one direction. The difficult part is the inclusion

in the other direction, shown by A. Kučera, T.A. Slaman, *SIAM J. Comput.*, 31:1(2001), 199–211. A survey is [C. Calude, *Theor. Comput. Sci.*, 271:1-2(2002), 3–14].

3.6.17. [41] Let $U_0, U_1, \ldots, U_k \subseteq \{0,1\}^\infty$ for all $k \geq 0$ denote a sequential universal Martin-Löf test as in Section 2.5.2. Let λ denote the uniform measure on $\{0,1\}^\infty$. Note that $U_0 = \{0,1\}^\infty$ by definition and $\lambda(U_0) = 1$. Lower semicomputable reals are defined in Exercise 3.6.15. A sequence $r_1, r_2, \ldots$ of reals is uniformly lower semicomputable if there is a total recursive function $f(k,i)$ such that for every $k \geq 1$, we have $f(k,i+1) \geq f(k,i)$ for all i and $\lim_{i\to\infty} f(k,i) = r_k$.

(a) Show that for every sequential universal Martin-Löf test $U_0, U_1, \ldots$, the uniform measure $\lambda(U_k)$ is a Martin-Löf random real.

(b) Show that for every lower semicomputable Martin-Löf random real r, there is a sequential universal Martin-Löf test $U_0, U_1, \ldots$ such that $\sum_{k=1}^\infty \lambda(U_k)$ equals r.

(c) Let $r_1, r_2, \ldots$ be a uniformly lower semicomputable sequence of reals such that $r_n \leq 1/2^n$ for every $n \geq 1$, and let λ be the uniform measure on $\{0,1\}^\infty$. Show that r_k is a Martin-Löf random real for every $k \geq 1$ iff there is a universal sequential Martin-Löf randomness test $U_0, U_1, \ldots$ with $\lambda(U_k) = r_k$ for every $k \geq 1$.

Comments. In this way, the measure-theoretic treatment of randomness in Martin-Löf's sense provides a second characterization of natural examples of Martin-Löf random sequences (or reals) that are lower semicomputable. In fact, it is a curious connection between infinite binary sequences that are members of the complement of the universal constructive null set, Section 2.5.2, and the binary expansions of the real numbers that are uniform measures of the constituent elements of a universal sequential Martin-Löf test, which elements themselves have infinite binary sequences as members. Source of Items (a) and (b): A. Kučera, T.A. Slaman, *Ibid.* (according to R.G. Downey, Item (a) is due to A. Kučera alone). Source for Item (c): (only if) A. Kučera, T.A. Slaman, *Ibid.*; (if) W. Merkle, N. Mihailovic, T.A. Slaman, *Theor. Comput. Syst.*, 39(2006), 707–721.

3.6.18. [37] In Exercises 3.6.16 and 3.6.17 we gave natural examples of lower semicomputable Martin-Löf random infinite binary sequences (or reals). They form as it were the fringe, the lowest, first, order of Martin-Löf randomness. These objects are random with respect to the primary notion of effective computability as represented by Turing machines. One can also consider more powerful notions of computability, called relativized computability, such as Turing machines equipped with oracles. Such an oracle is a subset A of the natural numbers, and a Turing machine T equipped with oracle A, denoted by T^A, can ask "is

$n \in A$?" Thus, if A is the set of programs (the binary code considered as a natural number) for which T (without oracle A) halts, then T^A can compute more than T. Let $T_1, T_2, \ldots$ be the standard enumeration of prefix machines, and let U be the reference prefix machine with $U(\langle i,p\rangle) = T_i(p)$ for all i,p. In recursion theory one defines the *jump* A' of A as $A' = \{x : U^A(x) < \infty\}$. Main jumps are those of the empty set: $\varnothing, \varnothing', \varnothing'', \ldots$. Clearly, $\varnothing'$ is recursively enumerable, by $U = U^{\varnothing}$, $\varnothing''$ is recursively enumerable by U' defined as $U' = U^{\varnothing'}$, $\varnothing'''$ is recursively enumerable by $U'' = U^{\varnothing''}$, and so on. Define the halting probability of U' by $\Omega' = \sum_{U'(p)<\infty} 2^{-l(p)}$, and similarly the halting probability of U'' by $\Omega'' = \sum_{U''(p)<\infty} 2^{-l(p)}$, and so on. Just as Ω is random with respect to the $\varnothing$ jump, every such halting probability is Martin-Löf random with respect to its respective jumps, that is, of the higher orders of randomness. We are interested in natural examples of infinite binary sequences (equivalently, real numbers) that are of these higher orders of randomness, but are defined without recourse to oracles.

(a) Show that the probability that a program for the reference universal prefix machine U both outputs finitely many symbols and does not halt (has an infinite computation) is Martin-Löf random in the first jump of the halting problem.

(b) Define the probability $\beta = \sum 2^{-l(p)}$, where the summation is taken over the shortest $p \in \{0,1\}^*$ such that the set $Q = \{q : U(pq) < \infty\}$ is cofinite ($\{0,1\}^* - Q < \infty$). Show that β is as random as Ω'': it is Martin-Löf random in the second jump of the halting problem.

Comments. Source Item (a): V. Becher, S. Diacz, G. Chaitin, *Proc. 3rd Discr. Math. Theor. Comput. Sci. Conf.*, Springer, London, 2001, pp. 55–68. Source for Item (b): V. Becher, G. Chaitin, *Fundamenta Informaticae*, 51(2002), 1–14.

3.6.19. [37] An infinite binary sequence ω is *n-random* if it is Martin-Löf random in $\varnothing^{n-1}$. That is, 1-randomness is Martin-Löf randomness. Show that $\omega = \omega_1\omega_2\ldots$ is 2-random iff $C(\omega_{1:n}) \geq n - O(1)$ for infinitely many n.

Comments. This interprets and explains Theorem 2.5.5 on page 153. Some authors call ω satisfying $C(\omega_{1:n}) \geq n - O(1)$ for infinitely many n *Kolmogorov random*. This definition was essentially proposed by D.W. Loveland, *Proc. ACM 1st Symp. Theory Comput.*, 1969, 61–65, using uniform Kolmogorov complexity in the definition. That definition is robust enough that uniform, length-conditional, and plain Kolmogorov complexity all give the same class. Source: For the 'if' side, [L. Yu, D. Ding, R.G. Downey, *Ann. Pure Appl. Logic*, 129:1-3(2004), 163–180] analyzed an argument of R.M. Solovay, *Ibid.* to show that all 3-random reals are Kolmogorov random; for the 'only if' side, [J.S. Miller, *J. Symbol. Logic*, 69(2004), 907–913]. Independently, the full 'iff' statement

was proved in [A. Nies, F. Stephan, S.A. Terwijn, *J. Symbol. Logic*, 70:2(2005), 515–535].

3.6.20. • [42] Does a similar result to that in Exercise 3.6.19 hold for prefix complexity? We know that the maximal complexity of $K(x)$ is $K^+(x) = l(x) + K(l(x)) + O(1)$, Example 3.3.2 on page 212. Moreover, if $K(x) = K^+(x)$ is maximal then $C(x) = C^+(x)$ by Exercise 3.4.4 on page 217. An infinite binary sequence ω is called *strongly Chaitin random* if $K(\omega_{1:n}) \geq n + K(n) - O(1)$ for infinitely many n.

(a) Show that every strongly Chaitin random infinite binary sequence is 2-random.

(b) Show that every 2-random infinite binary sequence is strongly Chaitin random.

Comments. Hint for Item (a): use Exercises 3.6.19 and 3.4.4. Source for item (a): R.M. Solovay, *Lecture Notes*, UCLA, 1975, unpublished. Source for Item (b): J.S. Miller, *The K-degrees, low for K degrees, and weakly low for K sets*, Manuscript, 2007.

3.6.21. [29] Let ω be an infinite binary sequence that is Martin-Löf random with respect to the uniform measure (for example, the halting probability Ω of Section 3.6.2).

(a) Show that ω is von Mises–Wald–Church random (Section 1.9).

(b) Show that ω is *effectively unpredictable*. That is, let f be a recursive function that given an arbitrary finite binary string predicts "the next bit is one," "the next bit is zero," or "no prediction." Then if f predicts infinitely many bits of ω, it does no better than chance because in the limit the relative frequency of both correct predictions and incorrect predictions is $\frac{1}{2}$.

(c) Use Item (b) to show that the zeros and ones have limiting relative frequency $\frac{1}{2}$.

(d) Show that ω is *normal* in base two in the sense of Borel: each of the 2^k possible blocks of k bits in ω has limiting relative frequency $1/2^k$.

(e) Show that the law of the iterated logarithm, Exercise 1.10.5 on page 65, applies to the relative frequency of correct and incorrect predictions of bits of ω.

Comments. Hint for Item (c): predict the next bit of ω by a total recursive function that always has value one. Source: G.J. Chaitin, *Algorithmic Information Theory*, Cambridge Univ. Press, 1987.

3.7 Algorithmic Properties of K

According to Section 3.4, the function K is not recursive but it is upper semicomputable (it can be approximated from above). This is also the case for the two-variable function $K(x|y)$. Also, Theorem 2.7.1 on page 174 and Corollary 2.7.2 on page 176 hold in precisely the same way by the same proofs with K replacing C. Barzdins's lemma, Theorem 2.7.2 on page 178, holds by the same proof with the following obvious modification: Every characteristic sequence χ of a recursively enumerable set A satisfies $K(\chi_{1:n}|n) \leq K^+(n) + c$ for all n, where c is a constant depending on A but not on n. There is a recursively enumerable set such that its characteristic sequence satisfies $K(\chi_{1:n}) \geq \log n - c$ for all n, where c is a constant that does not depend on n.

3.7.1 Randomness in Diophantine Equations

There is an algorithm to decide the solvability of the first n Diophantine equations, given about $\log n$ bits of extra information (Example 2.7.2 on page 179). Namely, given the number $m \leq n$ of soluble equations in the first n equations, we can recursively enumerate solutions to the first n equations in the obvious way until we have found m solvable equations. The remaining equations are unsolvable. This shows that the solubility of the enumerated Diophantine equations is interdependent in some way.

Suppose we replace the question of mere solubility by the question of whether there are finitely many or infinitely many different solutions. That is, no matter how many solutions we find for a given equation, by itself this can give no information on the question to be decided. It turns out that the set of indices of the Diophantine equations with infinitely many different solutions versus finitely many ones is not recursively enumerable.

Lemma 3.7.1 *There is an (exponential) Diophantine equation*

$$A(n, x_1, x_2, \ldots, x_m) = 0$$

that has only finitely many solutions $x_1, x_2, \ldots, x_m$ if the nth bit of Ω is zero and that has infinitely many solutions $x_1, x_2, \ldots, x_m$ if the nth bit of Ω is one.

The role of *exponential* Diophantine equations should be clarified. Yu.V. Matijasevich [*Soviet Math. Dokl.*, 11(1970), 354–357] proved that every recursively enumerable set has a polynomial Diophantine representation. Moreover, he proved in 1974 that every recursively enumerable set has a singlefold exponential Diophantine representation. This was published only later (with yet different proofs) in [J.P. Jones, Yu.V. Matijasevich, *J. Symbol. Logic*, 49(1984), 818–829; Yu.V. Matijasevich, *Hilbert's 10th Problem*, MIT Press, 1993]. It is not known whether singlefold representation (which is important in our application) is always possible without exponentiation. See also G.J. Chaitin, *Algorithmic Information Theory*, Cambridge University Press, 1987.

Proof. By dovetailing the running of all programs of the reference prefix machine U in the obvious way, we obtain a recursive sequence of rational numbers $\omega_1 \leq \omega_2 \leq \cdots$ such that $\Omega = \lim_{n\to\infty} \omega_n$. The set

$$R = \{(n,k) : \text{ the } n\text{th bit of } \omega_k \text{ is a one}\}$$

is a recursively enumerable (even recursive) set. The main step is to use a theorem due to Yu.V. Matijasevich, first published in [J.P. Jones, Yu.V. Matijasevich, *J. Symbol. Logic* 49(1984), 818–829], to the effect that "every recursively enumerable set R has a singlefold exponential Diophantine representation $A(\cdot,\cdot)$." That is, $A(p,y) = 0$ is an exponential Diophantine equation, and the singlefoldedness consists in the property that $p \in R$ iff there is a y such that $A(p,y) = 0$ is satisfied, and moreover, there is only a single such y. (Here both p and y can be multituples of integers; in our case, p represents $\langle n, x_1\rangle$, and y represents $\langle x_2, \ldots, x_m\rangle$. For technical reasons we consider as proper solutions only solutions x involving no negative integers.) It follows that there is an exponential Diophantine equation $A(n,k,x_2,\ldots,x_m) = 0$ that has exactly one solution $x_2,\ldots,x_m$ if the nth bit of the binary expansion of ω_k is a one, and it has no solution $x_2,\ldots,x_m$ otherwise. Consequently, the number of different m-tuples $x_1,x_2,\ldots,x_m$ that are solutions to $A(n,x_1,x_2,\ldots,x_m) = 0$ is infinite if the nth bit of the binary expansion of Ω is a one, and this number is finite otherwise. □

This can be interpreted as follows: Consider the sequence of Diophantine equations $D_1, D_2, \ldots$ such that $D_n(x_1,x_2,\ldots,x_m) = A(n,x_1,\ldots,x_m)$. Let us say that a formal theory "settles k cases" if it enables one to prove that the number of solutions of D_n is finite or is infinite for k specific values (not necessarily consecutive) of n. It is not difficult to show that no formal theory in which one can prove only true theorems and that is completely describable in n bits can settle more than $n + K(n) + O(1)$ cases. (Hint: use arguments about how many (scattered) bits of Ω can be determined (together with their positions) in a formal theory of given complexity.)

"This is a region in which mathematical truth has no discernible structure or pattern and appears to be completely random. These questions are completely beyond the power of human reasoning. Mathematics cannot deal with them. Quantum physics has shown that there is randomness in nature. I believe that we have demonstrated [...] that randomness is already present in pure Mathematics. This does not mean that the universe and Mathematics are completely lawless, it means that laws of a different kind apply: statistical laws. [...] Perhaps number theory should be pursued more openly in the spirit of an experimental science! To prove more one must assume more." [Chaitin]

Exercises

3.7.1. [42] (a) Show that K^+ (Exercise 3.4.1) is nonrecursive in the sense that there is no total recursive function f such that $K^+(x) = f(x) + O(1)$ for all x.

(b) Show that there is a recursive upper bound $f(x)$ of the function $K(x)$ with the property that $f(x) = K(x)$ holds for infinitely many x.

(c) Show that we cannot effectively find infinitely many x's for which some recursive upper bound on $K(x)$ (as in Item (b)) is sharp. (The same statement for $C(x)$ follows from Theorem 2.3.2, page 127.)

(d) Let $f(x)$ be a recursive upper bound on $K(x)$ as in Item (b). Show that in addition to Item (c), there is no recursive function $g(x)$ mapping each x to a finite set $X \subseteq \{y : y > x\}$ such that each X contains some y for which $f(y) \leq C(y)$. (Notice that the function $\log x$, or one almost equal to it, has this property for $C(x)$.)

Comments. The monotonic upper bound estimate of $\log x + K(\lceil \log x \rceil)$ on $K(x)$ is less satisfying than the sharp monotonic estimate $\log x$ on $C(x)$, because it is not a recursive function. The recursive upper bounds on $K(x)$ we have obtained previously, such as $\log x + 2 \log\log x$ and $\log^* x + l^*(x)$, are not precise up to a fixed additive constant for increasing x. The present Item (b) shows that nonetheless, we can find a recursive upper bound on $K(x)$ that is sharp infinitely often. However, we cannot find infinitely many x's for which this is the case, since Item (c) shows that every infinite set of x's for which the recursive upper bound coincides with $K(x)$ is not recursively enumerable. This also holds for $C(x)$. However, Item (d) shows a difference between $K(x)$ and $C(x)$ in effectiveness of a recursive upper bound that is sharp infinitely often. Source: R.M. Solovay, *Lecture Notes*, 1975, unpublished; P. Gács, Ph.D. thesis, Mathematics Department, Frankfurt am Main, 1978.

3.7.2. [35] Show that there is a recursive upper bound f on K and a constant c with the property that there are infinitely many x such that for all $k > 0$, the quantity of numbers $y \leq x$ with $K(y) < f(y) - k$ is less than $cx2^{-k}$.

Comments. Source: P. Gács, *Lecture Notes on Descriptional Complexity and Randomness*, Manuscript, Boston University, 1987.

3.8 *Complexity of Complexity

The complexity function K (or C for that matter) is nonrecursive (Theorem 2.3.2, page 127). If the number of bits one needs to know to compute $f(x)$ from x is not constant for all x, then f is nonrecursive. For instance, let f be defined by $f(x) = \Omega_{1:x}$, with Ω the halting probability in Section 3.6.2. Then $K(f(x)|x) \geq l(f(x)) + O(1)$. That is, knowledge of x does not help to describe $f(x)$ at all.

If f is recursive, then $K(f(x)|x) = O(1)$. But if f is 0,1-valued, then $K(f(x)|x) = O(1)$ too (since the programs for the constant functions of value 0 and 1 provide a bound). Thus, the complexity $K(f(x)|x)$ is a

property of f that tells something about the degree of nonrecursiveness of f, but the converse is not necessarily the case.

Definition 3.8.1 The *complexity* $K(f)$ of function f is defined as the function $K(f(x)|x)$.

We analyze $K(K(x)|x)$, the *complexity of the complexity function* K. There is a simple *upper bound* for all x. If $l(x) = n$, then $K(x) \leq n + 2\log n + O(1)$ and therefore

$$K(K(x)|x) \leq K(K(x)|n) + O(1) \leq \log n + O(1).$$

The first inequality follows from the fact that $l(x) = n$. The second inequality follows from the fact that if we know n, then to describe $K(x)$ it suffices to describe $|n - K(x)|$ and indicate whether $K(x) \geq n$ or $K(x) < n$. Since $|n - K(x)| < n$, and we know n, we can describe just the difference $d = n - |n - K(x)|$ self-delimitingly for $\log d + 2\log\log d + O(1) \leq \log n$, otherwise the plain value $|n - K(x)|$ self-delimitingly also in at most $\log n$ bits, indicating which is which in a constant number of bits. A similar, easier, argument yields the same upper bound for $C(C(x)|x)$. Therefore, we have that for every n and all strings x of length n we have $K(K(x)|x), C(C(x)|x) \leq \log n + O(1)$. It turns out that $K(K(x)|x)$ and $C(C(x)|x)$ can be very close to the upper bound.

Theorem 3.8.1 *For every n, there are strings x of length n such that $K(K(x)|x) \geq \log n - \log\log n + O(1)$. The same lower bound holds for $C(C(x)|x)$.*

Proof. The proof is a little tricky. Let U be the reference machine of Theorem 3.1.1. Fix a large enough n. In the sequel all x's considered have length n. Define the maximum value of $K(K(x)|x)$ by

$$s = \max_{l(x)=n} \min\{l(p) : U(p, x) = K(x)\}.$$

To prove the theorem we only need to show that

$$s \geq \log n - \log\log n + O(1). \tag{3.5}$$

Since $K(x) \leq n + 2\log n + O(1)$ for all x of length n,

$$K(K(x)|x) \leq s \leq \log n + O(1).$$

Let U be the reference prefix machine. A binary string p is called a *suitable* program for x if for some q we have

- $l(p) \leq s$;
- U computes $l(q)$ from p, given x; and

- U computes x from q.

In particular, there is a suitable program p for x, of length $l(p) = K(K(x)|x) \leq s$ (for instance, with corresponding q of length $l(q) = K(x)$). We denote by M_i the set of strings x for which there are at least i different suitable p. Now consider the sequence

$$\varnothing = M_{j+1} \subseteq M_j \subseteq \cdots \subseteq M_0 = \{0,1\}^n, \quad M_j \neq \varnothing.$$

There are $2^{s+1} - 1$ strings of length at most s. Therefore,

$$j \leq 2^{s+1}.$$

To prove the theorem, it suffices to show, for all $i \leq j$,

$$l(d(M_i)) \leq l(d(M_{i+1})) + 5 \log n. \tag{3.6}$$

Namely, this implies that $j \geq n/(5 \log n)$, which together with $j \leq 2^{s+1}$, proves Equation 3.5.

We prove Equation 3.6. There exists an $x \in M_i - M_{i+1}$, which is found by the following procedure with input $i, s, n, d(M_{i+1}), l(d(M_i))$, for all i with $0 \leq i \leq j$:

Step 1. Recursively enumerate all of M_{i+1}. **{**We know when we are done by the time we have found $d(M_{i+1})$ elements**}**

Step 2. Recursively enumerate enough of $M_i - M_{i+1}$ to make the sequel meaningful. For each z in $M_i - M_{i+1}$ we obtain, find by enumeration *all* i suitable programs for z. A suitable program p with $U(p, z)$ minimal satisfies $U(p, z) = K(z)$. We may assume that

$$\log d(M_i - M_{i+1}) \geq \log d(M_i) - 1, \tag{3.7}$$

since otherwise Equation 3.6 holds trivially. From Equation 3.7 it follows, by Theorem 3.3.1, that there exists a $z_{\max}$ among the z's for which

$$K(z_{\max}) \geq l(d(M_i)) - 1. \tag{3.8}$$

So we keep on enumerating z's until we find the first such $z_{\max}$.

Step 3. Set $x := z_{\max}$.

This algorithm provides a description of x containing the following items, each item in self-delimiting code:

- A description of this discussion in $O(1)$ bits;

- a description of $d(M_{i+1})$ in at most $l^*(n) + l(d(M_{i+1}))$ bits (since $K(l(d(M_{i+1}))) \leq l^*(n)$);
- a description of $l(d(M_i))$ in $l^*(n)$ bits;
- descriptions of i, n, and s, in $l^*(n)$, $l^*(n)$, and $l^*(\log n + 2\log\log n)$ bits, respectively.

The size of the description of x gives an upper bound on $K(x)$,

$$K(x) \leq 4l^*(n) + O(l^*(\log n)) + l(d(M_{i+1})) + O(1). \tag{3.9}$$

Equations 3.8 and 3.9 imply Equation 3.6, and hence the theorem. Precisely the same proof shows that $C(C(x)|x) \geq \log n - \log\log n + O(1)$. □

Since $C(C(x)) \leq \log n + O(1)$ for all x of length n, Theorem 3.8.1 indicates that for some x, knowledge of x only marginally helps to compute $C(x)$; most information in $C(x)$ is extra information. It turns out that these x's have lower complexity than random x's. Consider an x of length n with $C(x) \geq n - k$. Then $C(C(x)|x) \leq C(k) + O(1)$. If Theorem 3.8.1 holds for x, then it follows that $C(k) \geq \log n - \log\log n + O(1)$. Also, $C(k) \leq \log k + O(1)$. Then, $k = \Omega(n/\log n)$. Therefore, if Theorem 3.8.1 holds for x, then

$$C(x) \leq n - \Omega\left(\frac{n}{\log n}\right),$$

which implies that also $K(x) \leq n - \Omega(n/\log n)$. The same argument shows the following: Fix a constant c'. If Theorem 3.8.1 holds for x, then for each k, $0 \leq k \leq n$, such that $C(k|x) \leq c'$ (for example $k = \frac{1}{2}n$ or $k = \sqrt{n}$), we have $C(x) \notin [k - \delta, k + \delta]$, with $\delta = O(n/\log n)$.

3.9 *Symmetry of Algorithmic Information

Recall the *symmetry of algorithmic information* question raised in Sections 1.11 and 2.8. Up to what precision does the equality

$$K(x, y) = K(y|x) + K(x)$$

hold? It turns out that the precision is low, just as for C complexity, albeit for more fundamental reasons. It is at once obvious that

$$K(x, K(x)) = K(x) + O(1). \tag{3.10}$$

Namely, we can reconstruct both x and $K(x)$ from the shortest program for x. (Similarly $C(x, C(x)) = C(x) + O(1)$.) Now consider the difference of $K(x, y)$ from $K(y|x) + K(x)$ for $y = K(x)$. Combining Theorem 3.8.1 and Equation 3.10, for each n, there is a string x of length n,

$$K(x, K(x)) \leq K(x) + K(K(x)|x) - \log n + \log\log n + O(1). \tag{3.11}$$

For Equation 3.11 it does not really matter whether we use $K(x)$ or $C(x)$. However, $C(x)$ is nonadditive for reasons much simpler than for $K(x)$: additivity is violated already on random strings as shown in Section 2.8. But for $K(x)$, the possibly high-complexity $K(K(x)|x)$ is the only reason. Equation 3.10 is an obvious property that can be used for most variants of complexity to prove the corresponding version of Equation 3.11, which amounts to *asymmetry of information*.

This shows that analogues of the information-theoretic identities can hold only up to an additive term logarithmic in the complexity. In the case of $C(x)$ this is primarily caused by the randomness (incompressibility) of the strings concerned, Section 2.4. The overwhelming majority of strings is random enough for this effect to occur. One reason for focusing on the $K(x)$ measure was to eliminate this randomness-based effect. But it turns out that to obtain sharp analogues of the information-theoretic identities, a conditional term $K(\cdot|x)$ needs to be replaced by $K(\cdot|x, K(x))$. We necessarily require the extra information in $K(x)$, about the halting problem, that is not contained in x. This is the sole reason that the information-theoretic identities do not hold for $K(x)$ precisely. However, there are only relatively few x's with large $K(K(x)|x)$. This follows immediately from Example 3.8.

Another argument is as follows: Let χ be the characteristic function (equivalently, sequence) of a recursively enumerable set such that $K(\chi_{1:m}|n) \geq n$, where $m = 2^n$. Such sets exist by Barzdins's lemma, Theorem 2.7.2. Let $I(\chi : x) = K(x) - K(x|\chi)$, that is, the information in χ about x. For instance, we can choose χ as the characteristic function of the halting set $K_0 = \{\langle x, y\rangle : \phi_x(y) < \infty\}$. We can also take any other complete set. Then

$$K(K(x)|x) \leq I(\chi : x) + 3 \log I(\chi : x) + O(1). \tag{3.12}$$

This means that the extra information contained in $K(x)$, apart from x, is less than the information that any complete set contains about x. The a priori probability of x with high $K(K(x)|x)$ is low. Namely, it can be shown (but we omit the proof) that for any computable or lower semicomputable probability distribution (measure) on the x's, k bits of information concerning χ occur in x with probability 2^{-k}. Thus, we can derive, from the displayed inequality,

$$P\{x : K(K(x)|x) \geq i\} \leq i^3 2^{-i}, \tag{3.13}$$

for all computable and lower semicomputable probability distributions P. Equation 3.12 tells us that the extra information in $K(x)$, apart from x, is less than the information that any complete set contains about x. Equation 3.13 tells us that the a priori probability of all x with high $K(K(x)|x)$ is low. Source: attributed to L.A. Levin [P. Gács, *Soviet Math. Dokl.*, 15(1974), 1477–1480, and Ph.D. thesis, J.W. Goethe Univ., Frankfurt, 1978].

3.9.1 Algorithmic Information and Entropy

Let us recall some relations between Shannon entropy and algorithmic information, or complexity. Let P be a recursive probability distribution. In Example 2.8.1 on page 188 we have shown that for recursive probability distributions $P(\cdot)$ the expected value $C(\cdot)$ is asymptotic to $H(P)$ (in case both values grow unboundedly we have shown that the quotient of the compared quantities goes to 1). This implies the similar relation between expected $K(\cdot)$ value and $H(P)$; see Section 8.1.1.

A more direct proof uses the universal distribution in Chapter 4. Since the set of $K(x)$'s is the length set of a prefix code, the first inequality of the noiseless coding theorem, Theorem 1.11.2, shows that $H(P) \leq \sum_x P(x)K(x)$. Moreover, an effective version of the Shannon–Fano code in Example 1.11.2 on page 1.11.2 guarantees that $K(x) \leq \log 1/P(x) + O(1)$ (a formal proof is given later in Lemma 4.3.3 on page 274). Together this shows that the *entropy*

$$H(P) = \sum_x P(x) \log \frac{1}{P(x)}$$

of the distribution P is asymptotically equal to the *expected complexity*

$$\sum_x P(x)K(x)$$

with respect to probability $P(\cdot)$. However, we can make more precise calculations. The equality between expected prefix complexity and entropy is treated in detail in Section 8.1.1. It holds extremely precisely—up to an additive constant.

Instead of requiring $P(\cdot)$ to be recursive, it suffices to require that P be a lower semicomputable function. Together with $\sum P(x) = 1$, the latter requirement implies that $P(\cdot)$ is recursive (Example 4.3.2 on page 266).

The fact that the P-expectations of $\log 1/P(x)$ and $K(x)$ are close does not necessarily mean that those quantities are close together for all arguments. However, in Example 4.3.10 on page 282 we also show that for recursive $P(\cdot)$ the values $\log 1/P(x)$ and $K(x)$ are close to each other with high probability.

This establishes a tight quantitative relation between Shannon's statistical conception of entropy of a probability distribution and our intended interpretation of $K(x)$ as the information content of an individual object.

Let X and Y be two discrete random variables with a joint distribution. In Section 1.11 we defined the following notions: the *conditional entropy* $H(Y|X)$ of Y with respect to X, the *joint entropy* $H(X,Y)$ of X and Y, and the *information* $I(X:Y)$ in X about Y. The relations between these quantities are governed by Equations 1.13, 1.14, 1.15. The crucial one is Equation 1.13 on page 70: the additivity rule $H(X,Y) = H(X) + H(Y|X)$. In general we would like to derive the same relations for the complexity analogues. In Section 2.8 it turned out that the complexity analogue of Equation 1.13 holds in terms of C, within a logarithmic additive term (Theorem 2.8.2). The proof that Theorem 2.8.2 is sharp

used strings whose lengths were random. For K we also find that the exact analogue of Equation 1.13 on page 70 does not hold (Example 3.9 on page 244), not because it is violated on random x's with random lengths as is C, but for more subtle reasons.

3.9.2 Exact Symmetry of Information

By Equation 3.11, for each length n there are x of length n and y such that

$$|K(x,y) - K(x) - K(y|x)| = \Omega(\log K(x)),$$

showing that additivity can be satisfied only to within a logarithmic term. Since the complexity of the complexity function as expressed by Theorem 3.8.1 holds for all proper variants of complexity, additivity corresponding to Equation 1.13 cannot hold exactly for any proper variant of complexity.

While the complexity of the complexity function prevents an exact analogue to Equation 1.13, it turns out that nonetheless we can find an exact additivity property for K, by replacing the conditional x by $\langle x, K(x)\rangle$ (equivalently, by the shortest program for x).

Theorem 3.9.1 *Let x and y be finite binary strings. Then up to a fixed additive constant, $K(x,y) = K(x) + K(y|x, K(x))$.*

Proof. ($\leq$) Let p be a shortest program from which the reference prefix machine computes x, and let q be a shortest program for which it computes y given x and $K(x)$. But then we can find another prefix machine that with input pq uses p to compute x and $K(x)$ $(= l(p))$ and subsequently uses x and $K(x)$ to compute y from q.

($\geq$) Consider x and $K(x)$ as fixed constants. We need the following results, which are proven only later, in Chapter 4 (the conditional version Theorem 4.3.4 of Theorem 4.3.3 on page 275):

1. Recall Definition 1.7.4, page 35, of lower semicomputable functions $f : \mathcal{N} \to \mathcal{R}$. Let X consist of lower semicomputable functions $f_x(y)$ with $\sum_y f_x(y) < \infty$. There exists a $g \in X$ such that for all $f_x \in X$, $f_x(y) = O(g(y))$.
2. We can set this $g(y) = 2^{-K(y|x,K(x))}$.

Clearly, with x fixed the function $h_x(y) = 2^{K(x)-K(x,y)}$ is lower semicomputable. Moreover, with x fixed the set $\{K(x,y) : y \in \mathcal{N}\}$ is the length set of a prefix-free set of programs from which the reference prefix machine U computes $\langle x, y\rangle$. By the Kraft inequality, Theorem 1.11.1, we have

$$\sum_y 2^{-K(x,y)} \leq 1.$$

Therefore,

$$\sum_y h_x(y) \leq 2^{K(x)} < \infty.$$

Hence, $h_x \in X$, which implies by Items 1 and 2 that $h_x(y) = O(g(y))$. We obtain

$$K(x,y) \geq K(x) + K(y|x, K(x)) + O(1).$$

by substituting the expressions for g and h_x, taking the logarithm of both sides, and rearranging. □

This implies immediately the *subadditive property*

$$K(x,y) \leq K(x) + K(y|x) + O(1) \leq K(x) + K(y) + O(1),$$

which does not hold for C by Example 2.2.3 on page 118. This is a straightforward consequence of the fact that a prefix machine does not require extra information to see where one program ends and the other one begins—information that needs to be provided to the plain decoding algorithms in Chapter 2.

We cannot replace $\langle x, K(x)\rangle$ by $K(x)$ in Theorem 3.9.1. But we can replace x by $\langle x, K(x)\rangle$, as follows. Note that

$$K(x,y) = K(x, K(x), y) + O(1). \tag{3.14}$$

Substitution in Theorem 3.9.1 shows the following:

Corollary 3.9.1 K-complexity is additive in the form

$$K(\langle x, K(x)\rangle, y) = K(\langle x, K(x)\rangle) + K(y|\langle x, K(x)\rangle) + O(1).$$

Definition 3.9.1 The K-complexity of information in x about y is

$$I(x:y) = K(y) - K(y|x). \tag{3.15}$$

Define the conditional version as

$$I(x:y|z) = K(y|z) - K(y|x,z). \tag{3.16}$$

Information is symmetric if the information in x about y equals (up to additive constants) the information in y about x. Rewriting $K(x,y)$ in two different ways, it follows from Theorem 3.9.1 that

$$K(y) - K(y|x, K(x)) = K(x) - K(x|y, K(y)) + O(1).$$

Theorem 3.9.2 *Symmetry of information for K-complexity holds in the following form:* $I(\langle x, K(x)\rangle : y) = I(\langle y, K(y)\rangle : x) + O(1)$.

Example 3.9.1 We derive a (to our knowledge) new 'directed triangle inequality' that is needed later.

Lemma 3.9.1 *For all x, y, z,*

$$K(x|y^*) \leq K(x, z|y^*) + O(1) \leq K(z|y^*) + K(x|z^*) + O(1).$$

Proof. Using symmetry of information, Theorem 3.9.1, an evident inequality introducing an auxiliary object z, and twice symmetry of information again, we obtain

$$\begin{aligned} K(x, z|y^*) &= K(x, y, z) - K(y) + O(1) \\ &\leq K(z) + K(x|z^*) + K(y|z^*) - K(y) + O(1) \\ &\leq K(y, z) - K(y) + K(x|z^*) + O(1) \\ &= K(x|z^*) + K(z|y^*) + O(1). \end{aligned}$$

□

This lemma has bizarre consequences. These consequences are not simple unexpected artifacts of our definitions, but to the contrary, they show the power and the genuine contribution to our understanding represented by the deep and important mathematical relation represented by Theorem 3.9.1.

Define $k = K(y)$ and substitute $k = z$ and $K(k) = x$ to obtain the following counterintuitive corollary: To determine the complexity of the complexity of an object y it suffices to give both y and the complexity of y. This is counterintuitive, since in general we cannot compute the complexity of an object from the object itself; if we could, this would also solve the so-called halting problem, Section 1.7.2. This incomputability can be quantified in terms of $K(K(y)|y)$; which can rise to almost $K(K(y))$ for some y. But in the seemingly similar, but subtly different, setting below it is possible.

Corollary 3.9.2 As above, let k denote $K(y)$. Then,

$$\begin{aligned} K(K(k)|y, k) &= K(K(k)|y^*) + O(1) \\ &\leq K(K(k)|k^*) + K(k|y, k) + O(1) = O(1). \end{aligned}$$

We can iterate this idea. For example, the next step is that given y and $K(y)$ we can determine $K(K(K(y)))$ in $O(1)$ bits, which implies that $K(K(K(k)))|y, k) = O(1)$.

A direct construction works according to the following idea (where we ignore some important details): From k^* one can compute $\langle k, K(k)\rangle$, since k^* is by definition the shortest program for k and also by definition $l(k^*) = K(k)$. Conversely, from $k, K(k)$ one can compute k^*, by running all programs of length at most $K(k)$ in dovetailed fashion until the first program of length $K(k)$ halts with output k; this is k^*. The shortest program that computes the pair $\langle y, k\rangle$ has length $= k + O(1)$: We have $K(y, k) = k + O(1)$ (since the shortest program y^* for y carries both the information about y and about $k = l(y^*)$). Then, by Theorem 3.9.1 on page 247, we have $K(k) + K(y|k, K(k)) = k + O(1)$. In view of the information equivalence of $\langle k, K(k)\rangle$ and k^*, therefore $K(k) + K(y|k^*) = k + O(1)$. Let r be a program of length $l(r) = K(y|k^*)$ that computes y from k^*. Then, since $l(k^*) = K(k)$, there is a shortest program $y^* = qk^*r$ for y, where q is a fixed $O(1)$-bit self-delimiting program that unpacks and uses k^* and r to compute y. We are now in a position to show that $K(K(k)|y, k) = O(1)$. There is a fixed $O(1)$-bit program that includes knowledge of q and that enumerates two lists in parallel, each in dovetailed fashion: Using k it enumerates a list of all programs that compute k, including k^*. Given y and k it enumerates another list of all programs of length $k = l(y^*) + O(1)$ that compute y. One of these programs is $y^* = qk^*r$, which starts with qk^*. Since q is known, this self-delimiting program k^*, and hence its length $K(k)$, can be found by matching every element in the k list with the prefixes of every element in the y list in enumeration order. ◇

We cannot replace $\langle x, K(x)\rangle$ by $K(x)$ and $\langle y, K(y)\rangle$ by $K(y)$ in Theorem 3.9.2. The complexity version of individual information is asymmetric, in contrast to the expectation in the statistical entropy version of Equation 1.15, which is symmetric.

Lemma 3.9.2 *The error in symmetry of information using K-complexity is given by*

$$\begin{aligned}&\log l(x) - \log\log l(x) + O(1)\\ &\quad\le |I(x : y) - I(y : x)|\\ &\quad\le \log K(x) + \log K(y) + 2\log\log K(x) + 2\log\log K(y) + O(1).\end{aligned}$$

Proof. ($\le$) Use the conditional mutual information as in Equation 3.16 on page 248. Since $K(y|z) \le K(y|x, z) + K(x|z) + O(1)$,

$$I(x : y|z) \le K(x|z) + O(1). \tag{3.17}$$

We obtain, by first rewriting using Equations 3.14 and 3.16, and then using Equation 3.17, by the usual estimates of the relevant quantities,

$$K(x) + K(y|x) - K(x, y) = I(K(x) : y|x) + O(1)$$

$$\begin{aligned} &\leq K(K(x)|x) + O(1) \qquad (3.18)\\ &\leq \log K(x) + 2\log\log K(x) + O(1). \end{aligned}$$

Using Equation 3.18 in its turn we obtain

$$\begin{aligned} I(x:y) - I(y:x) &= I(K(y):x|y) - I(K(x):y|x) + O(1)\\ &\leq \log K(x) + \log K(y) + 2\log\log K(x)\\ &\quad + 2\log\log K(y) + O(1). \end{aligned}$$

($\geq$) Compute the error in additivity for the special case $y = \langle x, K(x)\rangle$. Rewrite using Definition 3.15, and apply Equation 3.10 (setting $x^* = \langle x, K(x)\rangle$):

$$\begin{aligned} I(x:x^*) - I(x^*:x) &= K(x^*) - K(x^*|x) - K(x) + K(x|x^*)\\ &= K(x,x^*) - K(x) - K(x^*|x) + O(1)\\ &= -K(K(x)|x) + O(1). \end{aligned}$$

Finally, for each n there is an x of length n such that

$$-K(K(x)|x) \leq -\log n + \log\log n + O(1),$$

by Theorem 3.8.1. □

We can use this to show that in fact $I(x:y)$ is *not even asymptotically symmetric*. Let $l(x) = n$. From $K(K(x)) \leq \log n + 2\log\log n + O(1)$ and Theorem 3.8.1 it follows that up to an additive constant,

$$I(x:K(x)) = K(K(x)) - K(K(x)|x) \leq 3\log\log n. \qquad (3.19)$$

By Theorem 3.8.1, using $K(K(x)|\langle x, K(x)\rangle) = O(1)$, there exist x of each length n, such that

$$I(\langle x, K(x)\rangle : K(x)) = K(K(x)) + O(1) \geq \log n - \log\log n + O(1).$$

By writing out the definitions, we have

$$I(K(x):x) = I(K(x):\langle x, K(x)\rangle) + O(1).$$

Consequently,

$$\begin{aligned} &\log n - 4\log\log n + O(1)\\ &\qquad \leq |I(\langle x, K(x)\rangle : K(x)) - I(K(x):\langle x, K(x)\rangle)|\\ &\qquad\quad + |I(x:K(x)) - I(K(x):x)|. \end{aligned}$$

This shows that the symmetry of information is violated strongly (in exponentially greater measure since $I(x:K(x)) = O(\log\log n)$) on at least one of the pairs $x, K(x)$ and $x^*, K(x)$. Source: attributed to L.A. Levin [P. Gács, *Soviet Math. Dokl.* 15(1974), 1477–1480].

Example 3.9.2 If a problem does not have a satisfactory solution as it is posed originally, it is a common mathematical ploy to change the definitions to accommodate the problem. Given $\langle x, K(x)\rangle$, we can enumerate all shortest programs for x. The first one we find is denoted by x^*. From x^* we can compute $\langle x, K(x)\rangle$. Hence, x^* and $\langle x, K(x)\rangle$ contain the same information although they are not identical strings. Below, we can replace x^* by $\langle x, K(x)\rangle$. Define $Kc(x|y) = K(x|y^*)$. When there is more than one object in the conditional then define $Kc(x|y,z) := Kc(x|\langle y,z\rangle) := K(x|\langle y,z\rangle^*)$, and so on. The unconditional versions don't change: define $Kc(x) := K(x)$ and $Kc(x,y) := K(x,y)$. We can formulate the additivity property as

$$Kc(x,y) = Kc(x) + Kc(y|x) + O(1).$$

Definition 3.9.2 Define the *mutual information* of two objects x and y by $I(x;y) = Kc(y) - Kc(y|x)$ (which we may view as the Kc analogue of $I(x:y)$).

Using Theorem 3.9.1, we find that the mutual information is a *symmetric quantity* as desired. Ignoring $O(1)$ terms,

$$I(x;y) = K(x) + K(y) - K(x,y) = I(y;x). \tag{3.20}$$

In general, the world looks prettier using Kc, and lots of basic identities can be derived (Exercise 3.9.3, page 253). One drawback about the conditional complexity $Kc(x|y)$ is that it is *not upper semicomputable* as a function of x and y. Namely, suppose the contrary and a function $Kc(x|y) = K(x,y) - K(y)$ is upper semicomputable. Then for fixed y, the function $2^{-Kc(x|y)}$ of variable x is lower semicomputable, and satisfies $\sum_x 2^{-Kc(x|y)} \leq 1$. Hence, by the Kraft inequality, for each fixed y the set $\{Kc(x|y) : x \in \{0,1\}^*\}$ is the length set of a prefix-code. But for fixed y, the set $\{K(x|y) : x \in \{0,1\}^*\}$ is also the length set of a prefix-code, and by the conditional variant of the invariance theorem, the shortest one among the upper semicomputable length sets. That is, for each y, there exists a constant $c_{Kc(\cdot|y)}$ such that for all x,

$$K(x|y) \leq Kc(x|y) + c_{Kc(\cdot|y)}.$$

Substituting $x = K(y)$, we obtain $Kc(K(y)|y) = O(1)$, while $K(K(y)|y)$ can be quite large by Theorem 3.9.1. ◇

Example 3.9.3 Using the Kc version of mutual information, the *conditional mutual information* is

$$\begin{aligned} I(x;y|z) &= K(x|z) - K(x|y, K(y|z), z) \\ &= K(x|z) + K(y|z) - K(x,y|z) + O(1). \end{aligned} \tag{3.21}$$

The elaborate conditionals are a consequence of the fact that x^* provides more information than x. Therefore, we have to be very careful when extending Theorem 3.9.1 on page 247. For example, the conditional version of it is

$$K(x,y|z) = K(x|z) + K(y|x, K(x|z), z) + O(1). \tag{3.22}$$

Note that a naive version

$$K(x,y|z) = K(x|z) + K(y|x^*, z) + O(1)$$

is incorrect: taking $z = x$, $y = K(x)$, the left-hand side equals $K(x^*|x)$, which can be as large as $\log n - \log\log n + O(1)$, and the right-hand side equals $K(x|x) + K(K(x)|x^*, x) = O(1)$.

But up to logarithmic precision we do not need to be that careful. In fact, in Section 8.1.1 it is shown that all linear (in)equalities that are valid for Kolmogorov complexity are also valid for Shannon entropy and vice versa—provided we require the Kolmogorov complexity (in)equalities to hold up to additive logarithmic precision only. ◇

Exercises

3.9.1. [22] Show that $\{K(x) : x = 1, 2, \ldots\}$ is the length set of an additively optimal universal code in the sense of Section 1.11.1.

3.9.2. [12] Let x^* be the first enumerated shortest program for x. Show that x^* and $\langle x, K(x)\rangle$ contain the same information: $K(x^*) = K(\langle x, K(x)\rangle) + O(1)$.

3.9.3. [20] Define Chaitin's conditional complexity as in Example 3.9.2 by $Kc(x|y) = K(x|y^*)$ with $y^* = \langle y, K(y)\rangle$, or y^* is the first enumerated shortest program for y. When there is more than one object in the conditional then define $Kc(x|y,z) := Kc(x|\langle y,z\rangle) := K(x|\langle y,z\rangle^*)$, and so on. Define $Kc(x) = K(x)$ and $Kc(x,y) = K(\langle x,y\rangle) = K(x,y)$. Prove the following identities (up to an additive constant):

(a) $Kc(x,y) = Kc(y,x)$.

(b) $Kc(x|x) = 0$.

(c) $Kc(Kc(x)|x) = 0$. (Contrast this with Theorem 3.8.1.)

(d) $Kc(x) \leq Kc(x,y)$.

(e) $Kc(x|y) \leq Kc(x)$.

(f) $Kc(x,y) = Kc(x) + Kc(y|x)$. (Contrast this with the case for $K(x,y)$, Theorem 3.9.1.)

(g) We have defined $I(x;y) = Kc(x) - Kc(x|y)$. Show that $I(x;y) \geq 0$, $I(x;x) = Kc(x)$, and $I(\epsilon;x) = I(x;\epsilon) = 0$.

(h) $I(x;y) = Kc(x) + Kc(y) - Kc(x,y) + O(1) = I(y;x) + O(1)$. (In fact, $I(x;y)$ is the symmetric quantity of mutual information of objects x and y; see Definition 3.9.2.)

(i) $Kc(x, Kc(x)) = Kc(x)$. (Hint: use (f) and (c). Contrast this derivation with the identical but differently formulated Equation 3.10.)

(j) $Kc(x,y) = Kc(x,y,Kc(x),Kc(y|x))$.

(k) $Kc(Kc(x),Kc(y|x)|x,y) = 0$.

(l) $Kc(Kc(x),Kc(y),Kc(y|x),Kc(x|y),Kc(x,y)|x,y) = 0$.

(m) $Kc(I(x;y)|x,y) = 0$.

(n) $Kc(x) \leq Kc(x|y) + Kc(y|z) + Kc(z)$.

(o) $Kc(x,y,z) = Kc(x|y,z) + Kc(y|z) + Kc(z)$.

(p) Show that $Kc(x|y)$ is not upper semicomputable. (Hint: use Theorem 3.8.1.) This contrasts with $K(x|y)$, which is upper semicomputable.

Comments. See also Example 3.9.2 in Section 3.9.1. Essentially, we have now at our disposal the entire calculus of information theory. In fact, the Kc calculus is somewhat richer, since it contains rules like Items (c) and (i) that have no counterpart in classical information theory. Note that the difference between K and Kc is the conditional form $K(x|y)$ versus $Kc(x|y)$. Kc complexity was introduced by G.J. Chaitin in *J. Assoc. Comp. Mach.* 22(1975), 329–340, and a recent exposé appears in *Algorithmic Information Theory*, Cambridge Univ. Press, 1987.

3.9.4. [12] Let $n = l(x)$ and $K(x) = n + K(n) + O(1)$. Show that $K(x,n) = K(x) + K(n|x) = K(n) + K(x|n) = K(x,n^*)$ up to additive constants.

Comments. This relates to the symmetry of information issue for K. The proof we gave that Theorem 2.8.2 on page 190 is sharp for C does not hold for K. Hence, to obtain that the analogue of Theorem 2.8.2 for K is sharp one has to use another argument. This argument uses the complexity of the complexity function, Theorem 3.8.1, which in fact holds for all variants of Kolmogorov complexity. Source: P. Gács, *Lecture Notes on Descriptional Complexity and Randomness*, Manuscript, Boston University, 1987.

3.9.5. [27] (a) Show that the mutual information $I(x;y) = K(x) + K(y) - K(x,y)$ according to Equation 3.20 on page 252 is symmetric: $I(x;y) = I(y;x)$.

(b) Show that the mutual information in Item (a) coincides with the mutual information $I(x:y) = K(y) - K(y|x)$ according to Equation 3.15 on page 248 up to an $O(\log K(x) + \log K(y))$ additive term.

3.10 History and References

Prefix complexity was first introduced in [L.A. Levin, *Problems Inform. Transmission*, 10:3(1974), 206–210; P. Gács, *Soviet Math. Dokl.*, 15(1974), 1477–1480; G.J. Chaitin, *J. Assoc. Comp. Mach.*, 22(1975), 329–340]. It resolves the technical problems of Solomonoff's original proposal for a universal a priori probability [R.J. Solomonoff, *A preliminary report on a general theory of inductive inference*, Tech. Rept. ZTB-138, Zator Company, Cambridge, Mass., November 1960; *Inform. Contr.*, 7(1964), 1–22, 224–254]. L.A. Levin [A.K. Zvonkin and L.A. Levin, *Russ. Math. Surv.*, 25:6(1970), 83–124] identified universal a priori probability with the maximal lower semicomputable semimeasure $\mathbf{M}$ over the sample space $\{0,1\}^\infty$, and it turns out that the negative logarithm of the $\mathbf{m}$ version of $\mathbf{M}$ coincides with complexity K (Theorem 4.3.3 on page 273 in Chapter 4). In some cases K is a more convenient complexity measure than C, or conversely, but for many applications one can use both equally well because they coincide to within a logarithmic additive term. Lemma 3.1.1 is due to L.A. Levin, *Soviet Math. Dokl.*, 17:2(1976), 522–526.

The material in Section 3.2 on the concrete implementation of a universal partial recursive function in combinatory logic to obtain explicit constants bounding how far $K(x)$ can exceed $l(x) + 2l(l(x))$, how far $K(x|l(x))$ can exceed $l(x)$, how far $C(x)$ can exceed $l(x)$, and so on, is due to J.T. Tromp [*Binary lambda calculus and combinatory logic*, Manuscript, CWI, Amsterdam, 2004]. Earlier, G.J. Chaitin [*Complexity*, 1:4(1995/1996), 55–59] used a LISP implementation of a reference universal prefix-machine to define concrete prefix complexity. He explicitly calculated constants involved in upper bounds on prefix complexity and with the halting probability (Section 3.6.2).

Estimates for the quantitative relation between C and K in Section 3.1 are from [L.A. Levin, *Problems Inform. Transmission*, 10:3(1974), 206–210; S.K. Leung-Yan-Cheong and T.M. Cover, *IEEE Trans. Inform. Theory*, IT-24(1978), 331–339; R.M. Solovay, *Lecture Notes*, UCLA, 1975, unpublished]. The numerical estimates on K-complexity and compressibility, like the upper bound on K and the distribution of description lengths in Theorem 3.3.1, are from [G.J. Chaitin, *J. Assoc. Comp. Mach.*, 22(1975), 329–340]. In the original submission of that paper Chaitin proposed to call an infinite binary sequence x random iff $K(x_{1:n}) \geq n - O(1)$. This was shown to be equivalent to Martin-Löf's notion of randomness, Theorem 3.6.1, by C.P. Schnorr, acting as a referee of that paper. That theorem is now known as Schnorr's theorem.

The halting probability Ω, Section 3.6.2, was popularized by Chaitin [*J. Assoc. Comp. Mach.*, 22(1975), 329–340] and C.H. Bennett [C.H. Bennett and M. Gardner, *Scientific American*, 241:11(1979), 20–34]. The relation between the halting probability and the solvability of whether

Diophantine equations have finitely many solutions or infinitely many solutions, Section 3.7.1, is due to Chaitin [*Adv. Appl. Math.*, 8(1987), 119–146; *Algorithmic Information Theory*, Cambridge Univ. Press, 1987]. The latter book also surveys prefix complexity and randomness of infinite sequences. It incorrectly attributes the presented results only to G.J. Chaitin, P. Martin-Löf, C.P. Schnorr, and R.M. Solovay; see also the book review by P. Gács [*J. Symb. Logic*, 54(1989), 624–627]. Related surveys are [A.N. Kolmogorov and V.A. Uspensky, *Theory Probab. Appl.*, 32(1987), 389–412; V.A. Uspensky, A.L. Semenov and A.K. Shen, *Russ. Math. Surv.*, 45:1(1990), 121–189; V.A. Uspensky, *J. Symb. Logic*, 57:2(1992), 385–412; An.A. Muchnik, A.L. Semenov, V.A. Uspensky, *Theoret. Comput. Sci.*, 2:207(1998), 1362–1376; J.S. Miller, A. Nies, Bull. Symb. Logic, 12:3(2006), 390–410.

The results of Yu.V. Matijasevich that every recursively enumerable set has a polynomial Diophantine representation, and that every recursively enumerable set has a singlefold exponential Diophantine representation, appeared in [*Soviet Math. Dokl.*, 11(1970), 354–357]. Matijasevich (in an email of April 10, 2003, to the authors) stated that this was proven by him in 1974 but published only later (with yet different proofs) in [J.P. Jones, Yu.V. Matijasevich, *J. Symbol. Logic*, 49(1984), 818–829; Yu.V. Matijasevich, *Hilbert's 10th Problem*, MIT Press, 1993].

Every universal prefix machine has an associated halting probability. A. Kučera and T.A. Slaman [*SIAM J. Comput.*, 31:1(2002], 199-211] have shown that the set of binary sequences corresponding to these halting probabilities equals precisely the set of Martin-Löf random binary sequences that are lower semicomputable. Moreover, they have also shown that the sum of the measures involved in a universal Martin-Löf test again coincide with the set of Martin-Löf random binary sequences that are lower semicomputable. Moreover, every uniformly computable sequence of lower semicomputable Martin-Löf random reals corresponds to the sequence of measures of the subsequent sets of some sequential Martin-Löf test, and vice versa, Exercise 3.6.17 on page 236. Developments in the theory at the crossroads of notions of individual randomness, Kolmogorov complexity, and recursion theory have blossomed in the last decades. Such work has been partially incorporated in the main text, and in the exercises, of Chapters 2 through 4. Detailed treatment is beyond the scope and physical size of this book, and is the subject of more specialized books: R.G. Downey, D.R. Hirschfeldt, *Algorithmic Randomness and Complexity*, Springer-Verlag, New York, to appear; A.K. Shen, V.A. Uspensky, N.K. Vereshchagin, *Kolmogorov Complexity and Randomness*, Elsevier, Amsterdam, to appear; A. Nies, *Computability and Randomness*, Oxford Univ. Press, to appear.

Theorem 3.8.1 on the complexity of the complexity function is due to P. Gács [*Soviet Math. Dokl.*, 15(1974), 1477–1480] and was later found

independently by R.M. Solovay [*Lecture Notes*, UCLA, 1975, unpublished]. This crucial result establishes the lower bound on the error term up to which information can be symmetric, for all possible variants of Kolmogorov complexity: Theorem 3.9.1, attributed to L.A. Levin in [P. Gács, *Soviet Math. Dokl.*, 15(1974), 1477–1480]. The equality of the expected algorithmic information (both the Kolmogorov complexity and prefix complexity variants) with Shannon's entropy is treated in detail in Section 8.1.1.

4

Algorithmic Probability

P.S. Laplace (1749–1827) pointed out the following reason why intuitively, a regular outcome of a random event is unlikely:

"We arrange in our thought all possible events in various classes; and we regard as *extraordinary* those classes which include a very small number. In the game of heads and tails, if heads comes up a hundred times in a row then this appears to us extraordinary, because the almost infinite number of combinations that can arise in a hundred throws are divided in regular sequences, or those in which we observe a rule that is easy to grasp, and in irregular sequences, that are incomparably more numerous." [Laplace]

If we define a regular object as an object with significantly less than maximal complexity, then the number of all regular events is small. This implies that the event that any one of them occurs has small probability (in the uniform distribution). Yet, the classical calculus of probabilities tells us that 100 heads is just as probable as any other sequence of heads and tails, even though our intuition tells us that it is less random than some others. Listen to the redoubtable Dr. Samuel Johnson:

"Dr. Beattie observed, as something remarkable which had happened to him, that he chanced to see both the No. 1 and the No. 1000, of the hackney-coaches, the first and the last; 'Why, Sir,' said Johnson, 'there is an equal chance for one's seeing those two numbers as any other two.' He was clearly right; yet the seeing of two extremes, each of which is in some degree more conspicuous than the rest, could not but strike one in a stronger manner than the sight of any other two numbers." [Boswell's *Life of Johnson*]

Laplace distinguishes between the object itself and a cause of the object:

"The regular combinations occur more rarely only because they are less numerous. If we seek a cause wherever we perceive symmetry, it is not that we

M. Li and P.M.B. Vitányi, *An Introduction to Kolmogorov Complexity and Its Applications*,
DOI: 10.1007/978-0-387-49820-1_4, © Springer Science + Business Media, LLC 2008

regard the symmetrical event as less possible than the others, but, since this event ought to be the effect of a regular cause or that of chance, the first of these suppositions is more probable than the second. On a table we see letters arranged in this order `C o n s t a n t i n o p l e`, and we judge that this arrangement is not the result of chance, not because it is less possible than others, for if this word were not employed in any language we would not suspect it came from any particular cause, but this word being in use among us, it is incomparably more probable that some person has thus arranged the aforesaid letters than that this arrangement is due to chance." [Laplace]

Let us try to turn Laplace's argument into a formal one. Suppose we observe a binary string x of length n and want to know whether we must attribute the occurrence of x to pure chance or to a cause. 'Chance' means that the literal x is produced by fair coin tosses. 'Cause' means that the reference prefix machine U computes x when its program is provided by fair coin tosses. The chance of generating x literally is about 2^{-n}. But the chance of generating x in the form of a short program from which U computes x is at least $2^{-K(x)}$. In other words, if x is regular, then $K(x) \ll n$, and it is about $2^{n-K(x)}$ times more likely that x arose as the result of computation from some simple cause (such as a short program) than literally by a random process.

This gives an objective and absolute definition of 'simplicity' as 'low Kolmogorov complexity.' Consequently, one obtains an objective and absolute version of the classic maxim of William of Ockham (1290?–1349?), known as Occam's razor: "If there are alternative explanations for a phenomenon, then, all other things being equal, we should select the simplest one." One identifies 'simplicity of an object' with 'an object having a short effective description.' In other words, a priori we consider objects with short descriptions more likely than objects with only long descriptions. That is, objects with low complexity have high probability, while objects with high complexity have low probability. Pursuing this idea leads to the remarkable probability distribution $2^{-K(x)}$ below.

4.1 Semicomputable Functions Revisited

We continue the treatment of semicomputable functions where we left it in Section 1.7.3. Nontrivial examples of functions that are upper semicomputable but not computable are $C(x)$, $C(x|y)$, $K(x)$, and $K(x|y)$ (Theorems 2.3.2, 2.3.3 on pages 127, 127, respectively). Examples of functions that are lower semicomputable but not computable are $-C(x)$, $-K(x)$, $2^{-K(x)}$, and the universal Martin-Löf test $\delta_0(x|L) = l(x) - C(x|l(x)) - 1$ with respect to the uniform distribution L.

Definition 4.1.1 A lower semicomputable function f is *universal* if there is an enumeration $f_1, f_2, \ldots$ of lower semicomputable functions, possibly with repetitions, such that $f(i,x) = f_i(x)$, for all $i, x \in \mathcal{N}$, where $f(i,x) = f(\langle i,x \rangle)$.

Lemma 4.1.1 *There is a universal lower semicomputable function.*

Proof. Let $\phi_1, \phi_2, \ldots$ be the effective enumeration of partial recursive functions in Section 1.7. Consider each partial recursive ϕ as a function $\phi : \mathcal{N} \times \mathcal{N} \to \mathcal{Q}$ by interpreting $\phi(\langle x,k\rangle) = \langle p,q\rangle$ as $\phi(x,k) = p/q$. Define for each i the function f_i by

$$f_i(x) = \sup_{k\in\mathcal{N}}\{\phi_i(x,k)\}, \tag{4.1}$$

or ∞ if such a maximum does not exist. Each such function f_i is lower semicomputable, since we can dovetail the computations of $\phi_i(x,k)$ for all $k \geq 1$. That is, the dovetailed computation proceeds by stages $1, 2, \ldots$. At each stage j, the overall computation executes step $j-k$ of the particular subcomputation of $\phi_i(x,k)$, for each k such that $j-k>0$. On the other hand, if f is a lower semicomputable function, then there exists a partial recursive function ϕ as in Equation 4.1 by Definition 1.7.4. In this way, we obtain an enumeration $f_1, f_2, \ldots$ of all and only partial functions that are lower semicomputable.

Define $\phi_0(i,\langle x,k\rangle) = \phi_i(x,k)$. Then ϕ_0 is a partial recursive function and there is an index j such that $\phi_j(\langle i,x\rangle,k) = \phi_0(i\langle x,k\rangle)$. The f_j-function corresponding to ϕ_j is also clearly lower semicomputable. Therefore, f_j is in the above enumeration. Then, $f_j(\langle i,x\rangle) = f_i(x)$, for all i and x. □

An analogous argument shows how to construct a function that is *universal upper semicomputable*. In contrast, there is no universal total computable function.

Example 4.1.1 If $f(x,y) \geq C(x|y)$, for all x and y, then we call $f(x,y)$ a *majorant* of $C(x|y)$. The minimum of any finite number of majorants is again a majorant. This is the way we combine different heuristics for recognizing patterns in strings. All upper semicomputable majorants of $C(x|y)$ share an interesting and useful property:

Lemma 4.1.2 *Let $f(x,y)$ be upper semicomputable. For all x,y we have $C(x|y) \leq f(x,y)+O(1)$ iff $d(\{x : f(x,y) \leq m\}) = O(2^m)$, for all y and m.*

Proof. (Only if) Suppose to the contrary that for each constant c, there exist y and m such that the number of x's with $f(x,y) \leq m$ exceeds $c2^m$. Then, by counting, there is an x such that $C(x|y) \geq m + \log c$. Therefore, $C(x|y) \geq f(x,y) + \log c + O(1)$. Letting $c \to \infty$ contradicts $C(x|y) \leq f(x,y) + O(1)$.

(If) Without loss of generality we can choose the minimal m satisfying the 'if' assumption, for each x and y. That is, given x and y, set $m := f(x,y)$. Let g be a partial recursive function such that $g(k,x,y) \geq g(k+$

$1, x, y)$ and $\lim_{k\to\infty} g(k, x, y) = f(x, y)$. Then we can describe x, given y and m, by the recursive function g approximating f from above, together with the index of x in enumeration order, namely, by enumerating all x's that satisfy $f(x, y) \leq m$.

Hence, $C(x|y, m) \leq m + O(1)$. Choose h such that $C(x|y, m) = m - h$. Using first the fact that we can reconstruct m from $C(x|y, m)$ and h and then substituting according to the definition of h gives

$$\begin{aligned} C(x|y) &\leq C(x|y, m) + 2\log h + O(1) \\ &\leq m - h + 2\log h + O(1). \end{aligned}$$

Since we have chosen $f(x, y) = m$, this proves the 'if' part. □ ◇

Definition 4.1.2 A real number $x = 0.x_1x_2\ldots$ is *lower semicomputable* if the set of rationals below x is recursively enumerable. A number x is *upper semicomputable* if $-x$ is lower semicomputable. A number x is *computable*, equivalently, *recursive*, if it is both lower semicomputable and upper semicomputable.

It is easy to show that x is lower semicomputable (respectively recursive) iff there is a lower semicomputable (respectively computable) function f such that $f(i) = x_i$ for all i. The halting probability $\Omega = \sum_{U(p)<\infty} 2^{-l(p)}$ (Section 3.6.2 on page 225) is a lower semicomputable real. Let us explicitly construct the approximation. Define $\phi(n) = \sum 2^{-l(p)}$, the sum taken over all programs p of the reference prefix machine U of Theorem 3.1.1 on page 202 with $l(p) \leq n$ that halt within n steps. Obviously, ϕ is a computable function, and $\phi(1), \phi(2), \ldots$ is a monotonic nondecreasing sequence of rational numbers with limit Ω. Similarly, $\sum_x 2^{-K(x)} < \Omega$, and moreover, the entire class of Ω-like reals introduced in Exercise 3.6.15 on page 234 consists of lower semicomputable reals.

4.2 Measure Theory

In this section, and in fact in the remainder of this entire chapter, we assume some knowledge of measure theory in the classical sense, say the basics in Section 1.6. In algorithmic probability theory it is customary to use a nonstandard approach to measures. For better or worse we will follow this usage. Let us first look at standard measure theory.

Classically, the framework is as follows: Let $\mathcal{B}$ be a finite or countably infinite set of *basic elements*. For example, $\mathcal{B} = \{0, 1\}$, $\mathcal{B} = \{0, 1\}^*$, or $\mathcal{B} = \mathcal{N}$ (the natural numbers). Consider the continuous sample space $S = \mathcal{B}^\infty$, that is, all one-way infinite sequences over $\mathcal{B}$.

We want to extend the idea of probability from finite sample spaces such as the outcomes {head, tail} for fair coin tosses to continuous sample spaces such as S. Probability cannot be properly defined for the individual elements of S. (The probability of selecting a particular real number r from the interval $[0, 1]$ is necessarily 0 for all but countably many elements.) Therefore, one defines the probability for subsets of S. Since there are too many subsets to describe, one first defines probability for countably many sets that are easily described. These sets are called 'cylinder' sets. Subsequently, by the operation of union, intersection, complement, and countable union, the probability definition is extended to many more subsets of S according to the Kolmogorov axioms in Section 1.6. (These 'Borel sets' are by no means all subsets of S.)

A *cylinder* is a set $\Gamma_x \subseteq S$ defined by

$$\Gamma_x = \{x\omega : \omega \in \mathcal{B}^\infty\}$$

with $x \in \mathcal{B}^*$. Let $\mathcal{G} = \{\Gamma_x : x \in \mathcal{B}^*\}$ be the set of all cylinders in S.

A function $\mu : \mathcal{G} \to \mathcal{R}$ defines a *probability measure* if

$$\mu(\Gamma_\epsilon) = 1,$$
$$\mu(\Gamma_x) = \sum_{b \in \mathcal{B}} \mu(\Gamma_{xb}).$$

(For general measures we can take $\mu(\Gamma_\epsilon) \in \mathcal{R}$ or even ∞.) In this definition we have defined the measures $\mu(\Gamma)$ only for all cylinders $\Gamma \subseteq S$. It induces measures for all subsets of S obtainable from the cylinders by intersection, union, complement, and countable union. It is a theorem of measure theory that μ uniquely induces a measure on the Borel sets. However, in the sequel we are primarily interested in the measures of the cylinders. For convenience of notation we replace the set function on cylinder sets by the isomorphic function on the defining initial segments.

Notation 4.2.1 A measure μ is defined by the function $\mu : \mathcal{G} \to \mathcal{R}$. Now consider the function $\mu' : \mathcal{B}^* \to \mathcal{R}$ defined by $\mu'(x) = \mu(\Gamma_x)$. Trivially, from μ' we can reconstruct μ. From now on we call the μ' functions 'measures' and drop the primes. Formally, we use the definition of measure below. One should keep in mind that our notation is shorthand for the original measure.

Definition 4.2.1 A function $\mu : \mathcal{B}^* \to \mathcal{R}$ is a *probability measure* if

$$\mu(\epsilon) = 1,$$
$$\mu(x) = \sum_{b \in \mathcal{B}} \mu(xb),$$

for all $x \in \mathcal{B}^*$. A semimeasure is a defective measure. A function $\mu : \mathcal{B}^* \to \mathcal{R}$ is a *semimeasure* if for all $x \in \mathcal{B}^*$,

$$\mu(\epsilon) \leq 1,$$
$$\mu(x) \geq \sum_{b \in \mathcal{B}} \mu(xb).$$

We can transform any semimeasure μ into a measure ρ by adding a distinguished element u not in $\mathcal{B}$, called the *undefined element*. We simply concentrate the surplus probability in Definition 4.2.1 on u by setting

$$\rho(\epsilon) = 1,$$
$$\rho(xu) = \rho(x) - \sum_{b \in \mathcal{B}} \mu(xb),$$

while for all $x \in \mathcal{B}^* - \{\epsilon\}$ we define $\rho(x) := \mu(x)$.

Example 4.2.1 For each $x \in \{0,1\}^*$ define the measure $\lambda(x) = 2^{-l(x)}$. This is the Lebesgue measure, or *uniform measure*, on the half-open unit interval $[0,1)$. It has a geometric interpretation. Consider the real numbers in $[0,1]$ as being represented by their binary representation after the binary point. A real like $\frac{1}{2}$ has two representations, namely, $0.100\ldots$ and $0.011\ldots$. We denote it by 0.1 and choose the representation with infinitely many zeros. The uniform measure $\lambda(x)$ of the cylinder Γ_x is the length $2^{-l(x)}$ of the half-open interval $[0.x, 0.x + 2^{-l(x)})$. ◇

This discussion leads to the central notion of this chapter: lower semicomputable and computable semimeasures.

Definition 4.2.2 A semimeasure μ is *lower semicomputable* (respectively *computable*) if the function μ is lower semicomputable (respectively computable).

Exercises

4.2.1. [18] Let μ be a semimeasure over $\mathcal{B}^*$. Show that if μ is computable, then we can find an algorithm to compute $\mu(x)$ and $\sum_{b \in \mathcal{B}} \mu(xb)$, for all $x \in \mathcal{B}^*$, to any degree of accuracy.

Comments. These properties are implicitly used throughout Section 4.5 on continuous semimeasures. Source: V.G. Vovk, *Soviet Math. Dokl.*, 35(1987), 656–660.

4.2.2. [14] (a) Let U be the reference prefix machine of Theorem 3.1.1 on page 202. Define $P(x) = \sum_{U(p)=x} 2^{-l(p)}$. Show that $\sum_x P(x) \leq 1$, so $P(x)$ qualifies as a probability mass function over the integers. (We use the term 'probability mass function' loosely here for nonnegative real-valued functions summing to *at most* 1.)

(b) Define $P(x) = 2^{-K(x)}$. Show that $\sum_x P(x) \leq 1$, the sum taken over all x, so $P(x)$ qualifies as a probability mass function over the integers.

(c) Define $P(x|y) = 2^{-K(x|y)}$. Show that $\sum_x P(x|y) \leq 1$, for each fixed y, so $P(x|y)$ qualifies as a conditional probability mass function over the integers.

(d) Define $P(x) = 2^{-K(x|l(x))}$. Show that $\sum_x P(x) = \infty$, so $P(x)$ does not qualify as a probability mass function.

Comments. Hint for Item (a): use the Kraft inequality, Theorem 1.11.1. Hint for Item (d): use $K(x|l(x) \leq l(x) + O(1)$.

4.3 Discrete Sample Space

We first develop the theory in the discrete domain. This is in a sense a first approximation to the theory in the continuous domain. One interpretation is to set $\mathcal{B} = \mathcal{N}$ and consider the sample space $S = \mathcal{N}$. (In terms of classical measure theory our sample space is $S = \{\Gamma_x : x \in \mathcal{N}^*, l(x) = 1\}$.) Since all elements of S are one-letter strings, the second item in Definition 4.2.1 on page 263 is not applicable. Since the elements of $\mathcal{N}$ are considered one-letter strings, none of them is a prefix of any other. All elements of $\mathcal{N}$ have prefix ϵ. Definition 4.2.1 requires such a measure μ to satisfy $\mu(\epsilon) = \sum_{x \in \mathcal{N}} \mu(x) = 1$. Except for the interpretation, there is no difference between a discrete measure and a probability distribution over a sample space $\mathcal{N}$. We use the same font (capital italics) to denote them.

Definition 4.3.1 A *discrete semimeasure* is a function P from $\mathcal{N}$ into $\mathcal{R}$ that satisfies $\sum_{x \in \mathcal{N}} P(x) \leq 1$. It is a *probability measure* if equality holds.

Example 4.3.1 (Relation Discrete and Continuous Measure)
The discrete Lebesgue measure L on the set of basic elements $\mathcal{B} = \mathcal{N}$ is a function $L : \mathcal{N} \to \mathcal{R}$ defined by $L(x) = 2^{-2l(x)-1}$. We verify that L is a probability measure:

$$\sum_{x \in \mathcal{N}} L(x) = \sum_{n \in \mathcal{N}} \left(2^{-n-1} \sum_{l(x)=n} 2^{-l(x)} \right) = \sum_{n \in \mathcal{N}} 2^{-n-1} = 1.$$

Here we use $l(x)$ not as the number of occurrences of basic symbols in x (there is only 1 occurrence of an element in $\mathcal{B}$) but rather as just a function.

The continuous Lebesgue measure λ on the set of basic elements $\mathcal{B} = \{0, 1\}$ is a function $\lambda : \{0, 1\}^* \to \mathcal{R}$ defined by $\lambda(x) = 2^{-l(x)}$ (from Example 4.2.1). In the discrete measure $L(x)$, the argument x is an element of $\mathcal{N}$ with the interpretation that no two different arguments

are prefixes of each other. In the continuous measure $\lambda(x)$, the argument x is an element of $\{0,1\}^*$ with the usual interpretation that arguments can be prefixes of other ones. An incorrect interpretation is seductive by confusing $\mathcal{B} = \mathcal{N}$, which is both the basic set and the domain in the discrete case, with the basic set $\mathcal{B} = \{0,1\}$ and/or the domain $\mathcal{B}^*$ in the continuous case.

We give a numerical example. According to our standard correspondence Equation 1.3, we have $1, 5, 6 \in \mathcal{N}$ correspond to $0, 10, 11 \in \{0,1\}^*$. But $L(1) > L(5) + L(6)$, since $L(1) = \frac{1}{8}$ and $L(5) = L(6) = \frac{1}{32}$. The interpretation in terms of cylinder sets for L is that $\Gamma_1, \Gamma_5, \Gamma_6$ are pairwise disjoint. As a comparison, for the continuous measure λ we have $\lambda(1) = \lambda(10) + \lambda(11)$, and the interpretation in cylinder sets is that $\Gamma_1 = \Gamma_{10} \bigcup \Gamma_{11}$.

For λ we have $\bigcup_{l(x)=n} \Gamma_x = S$ for each n. Thus, $\sum_{l(x)=n} \lambda(x) = 1$ for each n and

$$\sum_{x \in \{0,1\}^*} \lambda(x) = \infty.$$

In contrast, for the discrete measure L we have $\bigcup_{l(x)=n} \Gamma_x \subset S$, and we have $\sum_{l(x)=n} L(x) = 2^{-n-1}$ for each n, and hence $\sum_{x \in \mathcal{N}} L(x) = 1$. ◇

Example 4.3.2 If a lower semicomputable semimeasure P is a probability measure, then it must be computable. By Definition 1.7.4, there exists a recursive function $g(x,k)$, nondecreasing in k, with $P(x) = \lim_{k \to \infty} g(x,k)$. We can compute an approximation P^k of the function P from below for which $\sum_x P^k(x) > 1 - \epsilon$. This means that $|P(x) - P^k(x)| < \epsilon$, for all x. ◇

4.3.1 Universal Lower Semicomputable Semimeasure

In Section 4.1 we defined the notion of a universal two-argument function as being in an appropriate sense able to simulate each element in the class of one-argument lower semicomputable functions. This was similar to the universality notion in Turing machines. We now look at a slightly different notion of 'universality' meaning that some one-argument function is the 'largest' in a class of one-argument functions.

Definition 4.3.2 Let $\mathcal{M}$ be a class of discrete semimeasures. A semimeasure P_0 is *universal* (or maximal) for $\mathcal{M}$ if $P_0 \in \mathcal{M}$, and for all $P \in \mathcal{M}$, there exists a constant c_P such that for all $x \in \mathcal{N}$, we have $c_P P_0(x) \geq P(x)$, where c_P possibly depends on P but not on x.

We say that P_0 (multiplicatively) *dominates* each $P \in \mathcal{M}$. It is easy to prove that the class of all semimeasures has no universal semimeasure. This is also the case for the class of computable semimeasures (Lemma 4.3.1 below).

Theorem 4.3.1 *There is a universal lower semicomputable discrete semimeasure. We denote it by* $\mathbf{m}$.

Proof. We prove the theorem in two steps. In Stage 1 we show that the lower semicomputable discrete semimeasures can be effectively enumerated as

$$P_1, P_2, \ldots .$$

In Stage 2 we show that P_0 as defined below is universal:

$$P_0(x) = \sum_{j \geq 1} \alpha(j) P_j(x),$$

with $\sum \alpha(j) \leq 1$, and $\alpha(j) > 0$ and lower semicomputable for every j. Stage 1 consists of two parts. In the first part, we enumerate all lower semicomputable functions; and in the second part we effectively change the lower semicomputable functions to lower semicomputable discrete semimeasures, leaving the functions that were already discrete semimeasures unchanged.

STAGE 1 Let $\psi_1, \psi_2, \ldots$ be an effective enumeration of all real-valued partial recursive functions. Consider a function ψ from this enumeration (where we drop the subscript for notational convenience). Without loss of generality, assume that each ψ is approximated by a rational-valued two-argument partial recursive function $\phi'(x,k) = p/q$ (this is the interpretation of the literal $\phi'(\langle x,k\rangle) = \langle p,q\rangle$). Without loss of generality, each such $\phi'(x,k)$ is modified to a rational-valued two-argument partial recursive function $\phi(x,k)$ so as to satisfy the properties below. For all $x \in \mathcal{N}$, for all $k > 0$,

- if $\phi(x,k) < \infty$, then also $\phi(x,1), \phi(x,2), \ldots, \phi(x,k-1) < \infty$ (this can be achieved by the trick of dovetailing the computation of $\phi'(x,1)$, $\phi'(x,2)$, ... and assigning computed values in enumeration order to $\phi(x,1), \phi(x,2), \ldots$);
- $\phi(x,k+1) \geq \phi(x,k)$ (dovetail the computation of $\phi'(x,1)$, $\phi'(x,2)$, ... and assign the enumerated values to $\phi(x,1), \phi(x,2), \ldots$ satisfying this requirement and ignoring the other computed values); and
- $\lim_{k\to\infty} \phi(x,k) = \psi(x)$ (as does $\phi'(\cdot)$).

The resulting ψ-list contains all lower semicomputable real-valued functions, and is actually represented by the approximators in the ϕ-list.

Each lower semicomputable function ψ (rather, the approximating function ϕ) will be used to construct a discrete semimeasure P. In the algorithm below, the local variable array P contains the current approximation to the values of P at each stage of the computation. This is doable because the nonzero part of the approximation is always finite.

Step 1. Initialize by setting $P(x) := 0$ for all $x \in \mathcal{N}$; and set $k := 0$.

Step 2. Set $k := k+1$, and compute $\phi(1,k), \ldots, \phi(k,k)$. **{**If any $\phi(i,k)$, $1 \leq i \leq k$, is undefined, then P will not change any more and it is trivially a discrete semimeasure**}**

Step 3. If $\phi(1,k) + \cdots + \phi(k,k) \leq 1$ **then** set $P(i) := \phi(i,k)$ for all $i = 1, 2, \ldots, k$ **else** terminate.
{Step 3 is a test of whether the new assignment of P-values satisfies the discrete semimeasure requirements**}**

Step 4. Go to Step 2.

If ψ is already a discrete semimeasure, then $P = \psi$. If for some x and k with $x \leq k$ the value $\phi(x,k)$ is undefined, then the last assigned values of P do not change any more even though the computation goes on forever. Because the condition in Step 3 is satisfied by the values of P, it is a discrete semimeasure. If the condition in Step 3 gets violated, then the computation terminates and the P-approximation to P is a discrete semimeasure—even a computable one.

Executing this procedure on all functions in the list $\phi_1, \phi_2, \ldots$ yields an effective enumeration $P_1, P_2, \ldots$ of all lower semicomputable discrete semimeasures (and only lower semicomputable discrete semimeasures).

STAGE 2 Define the function P_0 as

$$P_0(x) = \sum_{j \geq 1} \alpha(j) P_j(x),$$

with $\alpha(j)$ chosen such that $\sum_j \alpha(j) \leq 1$, and $\alpha(j) > 0$ and lower semicomputable for all j. Then P_0 is a discrete semimeasure since

$$\sum_{x \geq 0} P_0(x) = \sum_{j \geq 1} \alpha(j) \sum_{x \geq 0} P_j(x) \leq \sum_{j \geq 1} \alpha(j) \leq 1.$$

The function P_0 is also lower semicomputable, since $P_j(x)$ is lower semicomputable in j and x. (Use the universal partial recursive function ϕ_0 and the construction above.) Finally, P_0 dominates each P_j since $P_0(x) \geq \alpha(j) P_j(x)$. Therefore, P_0 is a universal lower semicomputable discrete semimeasure. Obviously, there are countably infinitely many universal lower semicomputable semimeasures. We now fix a *reference*

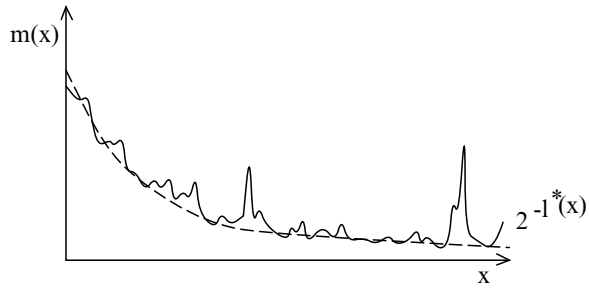


FIGURE 4.1. Graph of $\mathbf{m}(x)$ with lower bound $1/(x \cdot \log x \cdot \log\log x \cdots)$

universal lower semicomputable discrete semimeasure and denote it by $\mathbf{m}$, depicted in Figure 4.1. □

Example 4.3.3 In the definition of $\mathbf{m}(x)$ as $\sum_j \alpha(j)P_j(x)$ we can choose $\alpha(j) = 2^{-j}$ or $\alpha(j) = 6/(\pi j)^2$. In choosing α we must take care that the resulting $\mathbf{m}$ is lower semicomputable. In particular, we can define

$$\mathbf{m}(x) = \sum_{j \geq 1} 2^{-K(j)-O(1)} P_j(x),$$

the form of which will turn out to be convenient later. Namely, this assignment yields the domination relation $\mathbf{m}(x) \geq 2^{-K(j)-O(1)}P_j(x)$. The domination constant $2^{-K(j)}$ is for simple j much larger than the domination constant 2^{-j}. With $K(P) = \min\{K(j) : P = P_j\} + O(1)$ we write

$$\mathbf{m}(x) \geq 2^{-K(P)}P(x), \tag{4.2}$$

for all lower semicomputable discrete semimeasures $P = P_j$. ◇

Definition 4.3.3 Let $f(x, y)$ be a lower semicomputable function such that for each fixed y we have $\sum_x f(x, y) \leq 1$. Such functions f define *lower semicomputable conditional probability mass functions* $P(x|y) = f(x, y)$.

For example, y describes the relation $x \in A$, where A is a finite set. In the by now familiar manner, we can effectively enumerate this family of lower semicomputable probability mass functions as $P_1, P_2, \ldots$, for example by setting $P_j(x|y) = Q_j(\langle x, y\rangle)$ with Q_j the jth element in the list of lower semicomputable unconditional probability mass functions.

Definition 4.3.4 We can now define the *conditional* version of $\mathbf{m}(x)$ as

$$\mathbf{m}(x|y) = \sum_{j \geq 1} 2^{-K(j)-O(1)} P_j(x|y).$$

Theorem 4.3.2 *If $P(x|y)$ is a lower semicomputable conditional probability mass function, then $2^{K(P)}\mathbf{m}(x|y) \geq P(x|y)$, for all x, y.*

Proof. Similar to the argument in Example 4.3.3. □

Lemma 4.3.1 *The class of computable semimeasures has no universal element.*

Proof. Suppose a computable semimeasure P_0 is universal for the class of computable semimeasures. By its computability, $P_0(x)$ is approximable to any degree of accuracy; by its universality, $P_0(x)$ is strictly positive for every x (it multiplicatively dominates 2^{-x-1}); and since it is a semimeasure, then necessarily, $P_0(x) \to 0$ for $x \to \infty$. Consequently, we can compute an infinite sequence $x_1, x_2, \ldots$ such that x_i is the least value satisfying $P_0(x_i) < 2^{-i}/i$ $(i = 1, 2 \ldots)$. Therefore, the function Q, defined by $Q(x_i) := 2^{-i}$ for $i := 1, 2, \ldots$, and zero otherwise, is computable. Moreover, $\sum_x Q(x) = 1$, so Q is a (semi)measure. However, for every i there is an $x := x_i$ such that $Q(x) = 2^{-i} > iP_0(x)$, which contradicts the universality of P_0. □

This lemma is one of the reasons for introducing the notion of lower semicomputable semimeasures, rather than sticking to computable ones. Compare this to the introduction of lower semicomputable Martin-Löf tests in Sections 2.4, 2.5: among the recursive Martin-Löf tests there is no universal one.

Lemma 4.3.2 *The function $\mathbf{m}$ is incomputable and $\sum_x \mathbf{m}(x) < 1$.*

Proof. If $\mathbf{m}$ were computable, then it would be universal for the class of computable semimeasures, by Theorem 4.3.1. But there is no such universal element by Lemma 4.3.1.

Example 4.3.2 tells us that if a lower semicomputable semimeasure is also a probability distribution, then it is computable. Since $\mathbf{m}$ is lower semicomputable but not computable, this implies $\sum_x \mathbf{m}(x) < 1$. □

Let us look at the dependency between computability and measurehood of lower semicomputable semimeasures. One reason for introducing semimeasures, instead of just restricting consideration to probability mass functions summing to one, is due to the fact that on some input the reference prefix machine U runs forever. Normalizing $\mathbf{m}$ by dividing each value of it by $\sum_x \mathbf{m}(x)$ yields a proper probability mass function, say $\mathbf{P}$, such that $\sum_x \mathbf{P}(x) = 1$. We know from Example 4.3.2 that if a lower semicomputable semimeasure is also a probability mass function, then it is computable. So either $\mathbf{P}$ is computable or it is not lower semicomputable. Measure $\mathbf{P}$ is not computable by the same argument that works for $\mathbf{m}$ in Lemma 4.3.2. Hence, $\mathbf{P}$ is not even lower semicomputable.

Example 4.3.4 Consider the behavior of $\mathbf{m}(x)$ as x runs over the natural numbers. Let $v(x) = 6/(\pi x)^2$ and let w be defined by

$$w(x) = \begin{cases} 1/x & \text{for } x = 2^k \text{ and } k \in \mathcal{N}^+, \\ 0 & \text{otherwise.} \end{cases}$$

It can be shown that $\sum_x w(x) = \sum_x v(x) = 1$. Since both functions are lower semicomputable, they are both dominated by $\mathbf{m}(x)$. (Even though the series $\sum_x 1/x$ associated with the upper bound $1/x$ on $w(x)$ diverges, $\mathbf{m}(x)$ also dominates $w(x)$). From the Kraft inequality, Theorem 1.11.1, we know that the series $\sum_x 1/(x \log^2 x)$ converges. The function $\mathbf{m}(x)$ dominates $1/(x \log^2 x)$, but jumps at many places higher than what is shown in Figure 4.1, witnessed by the domination of $\mathbf{m}(x)$ over $w(x)$. ◇

4.3.2 A Priori Probability

Let $P_1, P_2, \ldots$ be the effective enumeration of all lower semicomputable semimeasures constructed in Theorem 4.3.1. There is another way to effectively enumerate the lower semicomputable semimeasures. Think of the input to a prefix machine T as being provided by an indefinitely long sequence of fair coin flips. The probability of generating an initial input segment p is $2^{-l(p)}$. If $T(p) < \infty$, that is, T's computation on p terminates, then presented with any infinitely long sequence starting with p, the machine T, being a prefix machine, will read exactly p and no further.

Let $T_1, T_2, \ldots$ be the standard enumeration of prefix machines of Theorem 3.1.1 on page 202. For each prefix machine T, define

$$Q_T(x) = \sum_{T(p)=x} 2^{-l(p)}. \tag{4.3}$$

In other words, $Q_T(x)$ is the probability that T computes output x if its input is provided by successive tosses of a fair coin. This means that Q_T satisfies

$$\sum_{x \in \mathcal{N}} Q_T(x) \leq 1.$$

Equality holds exactly for those T for which each one-way infinite input contains a finite initial segment constituting a halting program.

We can approximate Q_T as follows. (The algorithm uses the local variable $Q(x)$ to store the current approximation to $Q_T(x)$.)

Step 1. Initialize $Q(x) := 0$ for all x.

Step 2. Dovetail the running of all programs on T so that in stage k, step $k - j$ of program j is executed. Every time the computation of some program p halts with output x, increment $Q(x) := Q(x) + 2^{-l(p)}$.

The algorithm approximates the displayed sum in Equation 4.3 for each x by the contents of $Q(x)$. This shows that Q_T is lower semicomputable. Starting from a standard enumeration of prefix machines $T_1, T_2, \ldots$, this construction gives an enumeration of only lower semicomputable semimeasures

$$Q_1, Q_2, \ldots .$$

The P-enumeration of Theorem 4.3.1 contains all elements enumerated by this Q-enumeration. We only need to prove that the Q-enumeration contains all lower semicomputable measures (Lemma 4.3.4).

Definition 4.3.5 The *universal a priori probability* on the positive integers is defined as

$$Q_U(x) = \sum_{U(p)=x} 2^{-l(p)},$$

where U is the reference prefix machine of Theorem 3.1.1.

The use of prefix machines in the present discussion rather than plain Turing machines is necessary. By Kraft's inequality, Theorem 1.11.1, the series $\sum_p 2^{-l(p)}$ converges (to ≤ 1) if the summation is taken over all halting programs p of any fixed prefix machine. In contrast, if the summation is taken over all halting programs p of a universal plain Turing machine, then the series $\sum_p 2^{-l(p)}$ diverges.

In Section 3.6.2 we studied the real number $\Omega = \sum_x Q_U(x)$ and called it the 'halting probability.' No matter how we choose reference U, the halting probability is less than 1. Namely, U does not halt for some finite input q (the *halting problem* in Section 1.7). That is, $\sum_x Q_U(x) \leq 1 - 2^{-l(q)}$. If we normalize $Q_U(x)$ by $\mathbf{P}(x) = Q_U(x)/\Omega$, then the resulting function $\mathbf{P}$ is not lower semicomputable. Namely, if it were lower semicomputable, it would also be computable by Example 4.3.2. By Theorem 4.3.3, the function $\mathbf{P}$ would also be universal, and by Lemma 4.3.1, this is impossible.

4.3.3 Algorithmic Probability

It is common to conceive of an object as being simpler if it can be briefly described. But the shortness of description of an object depends on the description methods we allow, the 'admissible description syntax,' so to speak. We want to effectively reconstruct an object from its description. The shortest self-delimiting effective description of an object x is quantified by $K(x)$.

This leads to a recursively invariant notion of algorithmic probability, which can be interpreted as a form of Occam's razor: the statement that one object is simpler than another is equivalent to saying that the former object has higher probability than the latter.

Definition 4.3.6 The *algorithmic probability* $R(x)$ of x is defined as

$$R(x) = 2^{-K(x)}.$$

Let us see what this means. Consider a simple object. If x consists of a string of n zeros, then $K(x) \leq \log n + 2 \log\log n + c$, where c is a constant independent of n. Hence,

$$R(x) \geq \frac{1}{2^c n \log^2 n}.$$

Generate a binary sequence y by n tosses of a fair coin. With overwhelming probability, $K(y) \geq n$. For such complex objects y,

$$R(y) \leq 2^{-n}.$$

4.3.4 The Coding Theorem

Now we are ready to state the remarkable and powerful fact that the universal lower semicomputable discrete semimeasure $\mathbf{m}(x)$, the universal a priori probability $Q_U(x)$, and the algorithmic probability $R(x) = 2^{-K(x)}$ all coincide up to an independent fixed multiplicative constant. In mathematics the fact that quite different formalizations of concepts turn out to be equivalent is often interpreted as saying that the captured notion has an inherent relevance that transcends the realm of pure mathematical abstraction. We call the generic distribution involved the *universal distribution*. The following is called the *coding theorem*.

Theorem 4.3.3 *There is a constant c such that for every x,*

$$\log \frac{1}{\mathbf{m}(x)} = \log \frac{1}{Q_U(x)} = K(x),$$

with equality up to an additive constant c.

Proof. Since $2^{-K(x)}$ represents the contribution to $Q_U(x)$ by a shortest program for x, we have $2^{-K(x)} \leq Q_U(x)$, for all x.

Clearly, $Q_U(x)$ is lower semicomputable. Namely, enumerate all programs for x, by running reference machine U on all programs at once in dovetail fashion: in the first phase, execute step 1 of program 1; in the second phase, execute step 2 of program 1 and step 1 of program 2; in the ith phase ($i > 2$), execute step j of program k for all positive j and k such that $j + k = i$. By the universality of $\mathbf{m}(x)$ in the class of lower semicomputable discrete semimeasures, $Q_U(x) = O(\mathbf{m}(x))$.

It remains to show that $\mathbf{m}(x) = O(2^{-K(x)})$. This is equivalent to proving that $K(x) \leq \log 1/\mathbf{m}(x) + O(1)$, as follows. Exhibit a prefix-code E encoding each source word x as a code word $E(x) = p$, satisfying

$$l(p) \leq \log \frac{1}{\mathbf{m}(x)} + O(1),$$

together with a decoding prefix machine T such that $T(p) = x$. Then,

$$K_T(x) \leq l(p).$$

Then, by the invariance theorem, Theorem 3.1.1 on page 202,

$$K(x) \leq K_T(x) + O(1).$$

On the way to constructing E as required, we recall a construction for the Shannon–Fano code:

Lemma 4.3.3 *If P is a semimeasure on the integers, $\sum_x P(x) \leq 1$, then there is a binary prefix-code E such that the code words $E(1), E(2), \ldots$ can be length-increasing lexicographically ordered and $l(E(x)) \leq \log 1/P(x) + 2$. This is the Shannon–Fano code.*

Proof. Let $[0,1)$ be the half-open real unit interval, corresponding to the sample space $S = \{0,1\}^\infty$. Each element ω of S corresponds to a real number $0.\omega$. Let $x \in \{0,1\}^*$. The half-open interval $[0.x, 0.x + 2^{-l(x)})$ corresponding to the cylinder (set) of reals $\Gamma_x = \{0.\omega : \omega = x \ldots \in S\}$ is called a *binary interval*. We cut off disjoint, consecutive, adjacent (not necessarily binary) intervals I_x of length $P(x)$ from the left end of $[0,1)$, $x = 1, 2, \ldots$. Let i_x be the length of the longest binary interval contained in I_x. Set $E(x)$ equal to the binary word corresponding to the leftmost such interval. Then $l(E(x)) = \log 1/i_x$. It is easy to see that I_x is covered by at most four binary intervals of length i_x, from which the claim follows. □

In contrast to the proof of Theorem 1.11.1, the Kraft inequality, we cannot assume here that the sequence $\log 1/P(1), \log 1/P(2), \ldots$ is non-decreasing. This causes a loss of almost two bits in the upper bound.

We use this construction to find a prefix machine T such that $K_T(x) \leq \log 1/\mathbf{m}(x) + c$. That $\mathbf{m}(x)$ is not computable but only lower semicomputable results in $c = 3$.

Since $\mathbf{m}(x)$ is lower semicomputable, there is a partial recursive function $\phi(x,t)$ with $\phi(x,t) \leq \mathbf{m}(x)$ and $\phi(x,t+1) \geq \phi(x,t)$, for all t. Moreover, $\lim_{t\to\infty} \phi(x,t) = \mathbf{m}(x)$. Let $\psi(x,t)$ be the greatest partial recursive lower bound of special form on $\phi(x,t)$ defined by

$$\psi(x,t) := \{2^{-k} : 2^{-k} \leq \phi(x,t) < 2 \cdot 2^{-k} \text{ and } \phi(x,j) < 2^{-k} \text{ for all } j < t\},$$

and $\psi(x,t) := 0$ otherwise. Let ψ enumerate its range without repetition. Then,

$$\sum_{x,t} \psi(x,t) = \sum_x \sum_t \psi(x,t) \leq \sum_x 2\mathbf{m}(x) \leq 2.$$

The series $\sum_t \psi(x,t)$ can converge to precisely $2\mathbf{m}(x)$ only in case there is a positive integer k such that $\mathbf{m}(x) = 2^{-k}$.

In a manner similar to the proof of Lemma 4.3.3 on page 274, we chop off consecutive, adjacent, disjoint half-open intervals $I_{x,t}$ of length $\psi(x,t)/2$, in enumeration order of a dovetailed computation of all $\psi(x,t)$, starting from the left-hand side of $[0,1)$. We have already shown that this is possible. It is easy to see that we can construct a prefix machine T as follows: If Γ_p is the leftmost largest binary interval of $I_{x,t}$, then $T(p) = x$. Otherwise, $T(p) = \infty$ (T does not halt).

By construction of ψ, for each x there is a t such that $\psi(x,t) > \mathbf{m}(x)/2$. Each interval $I_{x,t}$ has length $\psi(x,t)/2$. Each I-interval contains a binary interval Γ_p of length at least one-half of that of I (because the length of I is of the form 2^{-k}, it contains a binary interval of length 2^{-k-1}) . Therefore, there is a p with $T(p) = x$ such that $2^{-l(p)} \geq \mathbf{m}(x)/8$. This implies $K_T(x) \leq \log 1/\mathbf{m}(x) + 3$, which was what we had to prove. □

Corollary 4.3.1 If P is a lower semicomputable discrete semimeasure, then there is a constant $c_P = K(P) + O(1)$ such that $K(x) \leq \log 1/P(x) + c_P$.

Definition 4.3.7 The *conditional universal distribution* is $Q_U(x|y) = \sum_{U(p,y)=x} 2^{-l(p)}$.

By Theorem 4.3.2, we have $2^{K(P)}\mathbf{m}(x|y) \geq P(x|y)$, for all x, y. Hence $\mathbf{m}(x|y)$ is a *universal conditional lower semicomputable semimeasure* in the sense of being the largest (within a constant factor) nonnegative lower semicomputable conditional discrrete semimeasure. Then as a corollary of Theorem 4.3.3 (rather, of its proof), we have the *conditional coding theorem*:

Theorem 4.3.4 *There is a constant c such that for all x, y,*

$$\log \frac{1}{\mathbf{m}(x|y)} = \log \frac{1}{Q_U(x|y)} = K(x|y),$$

with equality up to an additive constant c.

Proof. Since $Q_U(x|y)$ is a lower semicomputable discrete semimeasure, say the jth one in the enumeration $P_1(x|y), P_2(x|y), \ldots$, we can set $Q_U(x|y) = P_j(x|y)$. By Definition 4.3.4, we have $\mathbf{m}(x|y) \geq 2^{-K(j)}P_j(x|y)$, and hence $\log 1/\mathbf{m}(x|y) \leq \log 1/Q_U(x|y) + O(1)$. Moreover, for fixed y, the Shannon–Fano code associated with $Q_U(x|y)$ has code-word length $\log 1/Q_U(x|y)$, and this length is upper semicomputable. Since $K(x|y)$ is the length of the shortest upper semicomputable code for x given y, it follows that $\log 1/Q_U(x|y) + O(1) \leq K(x|y)$. So it remains to prove

$K(x|y) \leq \mathbf{m}(x|y) + O(1)$, which, for every fixed y, follows the proof of Theorem 4.3.3 □

Together, Theorem 4.3.4 and Theorem 4.3.2 yield the following:

Corollary 4.3.2 *If P is a lower semicomputable discrete semimeasure, then $K(x|y) \leq \log 1/P(x|y) + K(P) + O(1)$.*

Theorem 4.3.3 shows that the universal lower semicomputable discrete semimeasure $\mathbf{m}$ in the P-enumeration, defined in terms of a function computed by a prefix machine, and the universal a priori probability distribution Q_U in the Q-enumeration, defined as the distribution to which the universal prefix machine transforms the uniform distribution, are equal up to a multiplicative constant. This is a particular case of the more general fact that both sequences enumerate the same functions and there are recursive isomorphisms between the two.

Lemma 4.3.4 *There are recursive functions f, g such that $Q_j = \Theta(P_{f(j)})$ and $P_j = \Theta(Q_{g(j)})$.*

Proof. Firstly, we construct f. Let $Q = Q_j$ be the discrete semimeasure induced by prefix machine $T = T_j$ if its programs are generated by fair coin flips. We compute $Q(x)$ from below by a recursive function $\phi(x,t)$ such that $\phi(x,t+1) \geq \phi(x,t)$ and $\lim_{t\to\infty} \phi(x,t) = Q(x)$. The function ϕ is defined as follows:

Step 1. For all x, set $\phi(x,0) := 0$. Dovetail the computation of T on all of its programs p. Let variable t count the steps of the resulting computation. Set $t := 0$.

Step 2. Set $t := t + 1$.

Step 3. For all p, x with $l(p), l(x) \leq t$ **do**:
if the computation $T(p)$ terminates in the tth step with $T(p) = x$ **then** set $\phi(x,t+1) := \phi(x,t) + 2^{-l(p)}$ **else** set $\phi(x,t+1) := \phi(x,t)$.

Step 4. Go to Step 2.

We can make the described procedure rigorous in the form of a prefix machine T'. Let this T' be T_m in the standard enumeration of prefix machines. The construction in Theorem 4.3.1 that transforms every prefix machine into a prefix machine computing a semimeasure leaves P_m invariant. Therefore, machine T_m computes P_m from below in the standard way in the P-enumeration. Hence, $Q = P_m$. It suffices to set $f(j) = m$. Clearly, f is recursive: we have just outlined the algorithm to compute it.

Secondly, we construct g. Let $P = P_j$ be the jth element in the effective enumeration constructed in Theorem 4.3.1. We follow the construction in the proof of the Theorem 4.3.3, with P substituted for $\mathbf{m}$. Just as in that proof, since P is lower semicomputable, we can find a prefix machine T_P such that for each x, the following two items hold:

1. We can construct a function $\sum_t \psi_P(x,t) \leq 2P(x)$ with corresponding prefix machine T_P such that $\sum_{T_P(p)=x} 2^{-l(p)} \leq P(x)$.

2. Moreover, $P(x)/8 \leq 2^{-K_{T_P}(x)}$. Therefore, $K_{T_P}(x) \leq \log 1/P(x)+3$.

Let $Q = \sum_{T_P(p)=x} 2^{-l(p)}$ be the discrete semimeasure induced by the prefix machine T_P if its programs are generated by fair coin flips. Then, with $Q = Q_m$ in the Q-enumeration, and $K_{T_P}(x) = \min_p\{l(p) : T_P(p) = x\}$, we have

$$2^{-K_{T_P}(x)} \leq \sum_{T_P(p)=x} 2^{-l(p)} = Q_m(x).$$

By Items 1 and 2 this implies that $Q_m(x) = \Theta(P(x))$. Obviously, the function g defined by $g(j) = m$ is recursive. □

Example 4.3.5 A priori, an outcome x may have high probability because it has many long descriptions. The coding theorem, Theorem 4.3.3, tells us that in that case it must have a short description too. In other words, the a priori probability of x is dominated by the shortest program for x.

Just as we have derived the discrete semimeasure Q_U from U, we can derive the discrete semimeasure Q_T from the prefix machine T constructed in the proof of Theorem 4.3.3. Since Q_T is lower semicomputable, we have $Q_T(x) = O(\mathbf{m}(x))$. By definition, $2^{-K_T(x)} = O(Q_T(x))$. In the proof of Theorem 4.3.3 it was shown that $K_T(x) \leq \log 1/\mathbf{m}(x) + 3$. Using Theorem 4.3.3 once more, we have $K(x) \leq K_T(x)+O(1)$. Altogether, this gives

$$\log \frac{1}{Q_T(x)} = \log \frac{1}{\mathbf{m}(x)} = K_T(x)$$

up to additive constants. Again, therefore, if x has many long programs with respect to T, then it also has a short program with respect to T. In general, we can ask the question, how many descriptions of what length does a finite object have? This leads us to the statistics of description length. For instance, it turns out that there is a universal constant limiting the number of shortest descriptions of *any* finite object, Exercise 4.3.6 on page 287. ◇

We compare the coding theorem, Theorem 4.3.3, with Shannon's noiseless coding theorem, Theorem 1.11.2 on page 77. The latter states that given any discrete semimeasure P on the positive integers, we can construct a binary prefix-code E in such a way that on average, $l(E(x)) \leq \log 1/P(x) + 1$. Recall from Section 1.11.4 the notion of 'universal code' as a code that gives near-optimal encodings for any discrete semimeasure on the source alphabet. The coding theorem shows that there is a *single fixed* upper semicomputable universal code E' for *every* lower semicomputable discrete semimeasure. The code $E'(x)$ is the shortest program to compute x by the reference prefix machine. The code-word lengths satisfy $l(E'(x)) = K(x)$ up to a fixed additive constant independent of x. We have shown that for every lower semicomputable discrete semimeasure P, $P(x) \leq c_P \mathbf{m}(x)$, where c_P is a constant depending on the distribution P but not on x. By Theorem 4.3.3, the code-word lengths satisfy $l(E'(x)) \leq \log 1/P(x) + c_P$. This is far better than the performance of any universal code we have met in Section 1.11.1.

4.3.5 Randomness by Sum Tests

In Theorem 2.4.1, page 138, we have exhibited a universal P-test for randomness of a string x of length n with respect to an arbitrary computable distribution P over the sample set $S = \mathcal{B}^n$ with $\mathcal{B} = \{0, 1\}$.

The universal P-test measures how justified the assumption is that x is the outcome of an experiment with distribution P. We now use $\mathbf{m}$ to investigate alternative characterizations of random elements of the sample set $S = \mathcal{B}^*$ (equivalently, $S = \mathcal{N}$).

Definition 4.3.8 Let P be a computable discrete semimeasure on $\mathcal{N}$. A *sum P-test* is a lower semicomputable function δ satisfying

$$\sum_x P(x) 2^{\delta(x)} \leq 1. \tag{4.4}$$

A *universal sum P-test* is a test that additively dominates each sum P-test.

The sum tests of Definition 4.3.8 are slightly stronger than the tests according to Martin-Löf's original definition, Definition 2.4.1 on page 135.

Lemma 4.3.5 *Each sum P-test is a P-test. If $\delta(x)$ is a P-test, then there is a constant c such that $\delta'(x) = \delta(x) - 2\log(\delta(x) + 1) - c$ is a sum P-test.*

Proof. Define $P_n(x) = P(x|l(x) = n)$ for $l(x) = n$ and 0 otherwise. It follows immediately from the new definition that for all nonnegative n and k,

$$\sum \{P_n(x) : \delta(x) \geq k\} \leq 2^{-k}. \tag{4.5}$$

Namely, if Equation 4.5 is false, then we contradict Equation 4.4 by

$$\sum_{x\in\mathcal{N}} P(x)2^{\delta(x)} \geq \sum_{l(x)=n,\delta(x)\geq k} P_n(x)2^k > 1.$$

Conversely, if $\delta(x)$ satisfies Equation 4.5 for all n, then for some constant c, the function $\delta'(x) = \delta(x) - 2\log(\delta(x)+1) - c$ satisfies Equation 4.4. Namely,

$$\begin{aligned}\sum_{l(x)=n} P_n(x)2^{\delta'(x)} &= \sum_k \sum_{l(x)=n} \{P_n(x) : \delta(x) = k\}\, 2^{\delta'(x)} \\ &\leq \sum_k 1/(2^c(k+1)^2), \end{aligned} \tag{4.6}$$

where the summation over k is from 0 to ∞. Choose a constant c such that the last sum converges to at most 1. Note that $c = 1$ will do, since $\sum_{k=0}^{\infty} 1/(k+1)^2 = \pi^2/6 < 2$. Since $P_n(x) = P(x)/\sum_{l(x)=n} P(x)$,

$$\sum_n \left(\sum_{l(x)=n} P(x)\right) \sum_{l(x)=n} P_n(x)2^{\delta'(x)} = \sum_x P(x)2^{\delta'(x)}.$$

On the left-hand side of this equality, the expression corresponding to the left-hand side of Equation 4.6 is bounded from above by 1. Therefore, the right-hand side of the equality is at most 1, as desired. □

This shows that the sum test is not much stronger than the original test. One advantage of Equation 4.4 is that it is just one inequality instead of infinitely many, one for each n. We give an exact expression for a universal sum P-test in terms of complexity.

Theorem 4.3.5 *Let P be a computable probability distribution. The function $\kappa_0(x|P) = \log(\mathbf{m}(x)/P(x))$ is a universal sum P-test.*

Proof. Since $\mathbf{m}$ is lower semicomputable and P is computable, $\kappa_0(x|P)$ is lower semicomputable. We first show that $\kappa_0(x|P)$ is a sum P-test:

$$\sum_x P(x)2^{\kappa_0(x|P)} = \sum_x \mathbf{m}(x) \leq 1.$$

It remains only to show that $\kappa_0(x|P)$ additively dominates all sum P-tests. For each sum P-test δ, the function $P(x)2^{\delta(x)}$ is a semimeasure that is lower semicomputable. By Theorem 4.3.1, there is a positive constant c such that $c\cdot\mathbf{m}(x) \geq P(x)2^{\delta(x)}$. Hence, there is another constant c such that $c + \kappa_0(x|P) \geq \delta(x)$, for all x. □

Example 4.3.6 An important case is as follows. Define $P_A(x) = P(x)/\sum_{x\in A} P(x)$ for $x \in A$ and 0 otherwise, and $\mathbf{m}_A(x) = \mathbf{m}(x)/\sum_{x\in A} \mathbf{m}(x)$ for $x \in A$ and 0 otherwise. If we consider a distribution P restricted to a domain $A \subset \mathcal{N}$, then the universal sum P-test becomes $\log(\mathbf{m}_A(x)/P_A(x))$. For example, if L_n is the uniform distribution on $A = \{0,1\}^n$, then the universal sum L_n-test for $x \in A$ becomes

$$\kappa_0(x|L_n) = \log \frac{\mathbf{m}_A(x)}{L_n(x)} = n - K(x|n) - O(1).$$

Namely, $L_n(x) = 1/2^n$ and $\log \mathbf{m}_A(x) = -K(x|n) + O(1)$ by Theorem 4.3.4, since we can describe A by giving n. Alternatively, use the definition of $\mathbf{m}_A(x)$ above, Exercise 4.3.7 on page 288, and the symmetry of information theorem, Theorem 3.9.1. ◇

Example 4.3.7 The noiseless coding theorem, Theorem 1.11.2 on page 77, says that the Shannon–Fano code, which codes a source word x straightforwardly as a word of about $\log 1/P(x)$ bits (Example 1.11.2 on page 68 and Lemma 4.3.3 on page 274), nearly achieves the optimal expected code-word length. This code is based solely on the probabilistic characteristics of the source, and it does not use any characteristics of the object x itself to associate a code word with a source word x. The code that codes each source word x as a code word of length $K(x)$ also achieves the optimal expected code-word length. This code is independent of the probabilistic characteristics of the source, and uses solely the characteristics of the individual objects x to obtain shorter code words. Any difference in code-word length between these two encodings for a particular object x is due to exploitation of the probability of x versus the individual regularities in x. Taking a probability that accounts for the regularities in x, the two code-word lengths coincide. This is the case for the universal probability $\mathbf{m}(x)$ of x, which has as its associated Shannon–Fano code-word length the prefix complexity $K(x) = \log 1/\mathbf{m}(x) + O(1)$ of x. For other probabilities P of x, the P-expected Shannon–Fano code-word length differs from the P-expected prefix complexity $K(x)$ by at most the complexity $K(P)$ of P; see Section 8.1.1. This is tied up with the notion of randomness deficiency, which we met earlier, in a new setting.

Definition 4.3.9 Following Section 2.2.1, define the *randomness deficiency* of a finite object x with respect to P as

$$\delta(x|P) = \left\lfloor \log \frac{1}{P(x)} \right\rfloor - K(x) = \log \frac{\mathbf{m}(x)}{P(x)} + O(1).$$

Then, $\delta(x|P) = \kappa_0(x|P) + O(1)$ by Theorems 4.3.3, 4.3.5. That is, the randomness deficiency is the outcome of the universal sum P-test of

Theorem 4.3.5. Thus, for simple distributions, the expected prefix complexity is about equal to the expected Shannon–Fano code-word length, that is, the expectation of the randomness deficiency is close to zero. ◇

Example 4.3.8 Let us compare the randomness deficiency as measured by $\kappa_0(x|P)$ with that measured by the universal test $\delta_0(x|L)$, for the uniform distribution L, in Section 2.4. That test consisted actually of tests for a whole family L_n of distributions, where L_n is the uniform distribution such that each $L_n(x) = 2^{-n}$ for $l(x) = n$, and zero otherwise. Rewrite $\delta_0(x|L)$ as

$$\delta_0(x|L_n) = n - C(x|n) - 1,$$

for $l(x) = n$, and ∞ otherwise. This equals the reference universal test with respect to the uniform distribution we met in Definition 2.4.3 on page 140, and is close to the expression for $\kappa_0(x|L_n)$ obtained in Example 4.3.6 on page 280. From the relations between C and K we have established in Chapter 3, it follows that

$$|\delta_0(x|L_n) - \kappa_0(x|L_n)| \leq 2\log C(x) + O(1).$$

The formulation of the universal sum test in Theorem 4.3.5 can be interpreted as follows: An element x is random with respect to a distribution P, that is, $\kappa_0(x|P) = O(1)$, if $P(x)$ is large enough, not in absolute value but relative to $\mathbf{m}(x)$. If we did not have this relativization, then we would not be able to distinguish between random and nonrandom outcomes for the uniform distribution $L_n(x)$ above.

Let us look at an example. Let $x = 00\ldots0$ of length n. Then $\kappa_0(x|L_n) = n - K(x|n) + O(1) = n + O(1)$. If we flip a coin n times to generate y, then with overwhelming probability, $K(y|n) \geq n - O(1)$ and $\kappa_0(y|L_n) = O(1)$.

◇

Example 4.3.9 According to modern physics, electrons, neutrons, and protons satisfy the Fermi–Dirac distribution (Exercise 1.3.6, page 11). We distribute n particles among k cells, for $n \leq k$, such that each cell is occupied by at most one particle; and all distinguished arrangements satisfying this have the same probability.

We can treat each arrangement as a binary string: an empty cell is a zero and a cell with a particle is a one. Since there are $\binom{k}{n}$ possible arrangements, the probability for each arrangement x to happen, under the Fermi–Dirac distribution, is $FD_{n,k}(x) = 1/\binom{k}{n}$. Denote the set of possible arrangements by $A(n,k)$. According to Theorem 4.3.5,

$$\begin{aligned}\kappa_0(x|FD_{n,k}) &= \log \frac{\mathbf{m}_{A(n,k)}(x)}{FD_{n,k}(x)} \\ &= -K(x|n,k) + \log \binom{k}{n} + O(1)\end{aligned}$$

is a universal sum test with respect to the Fermi–Dirac distribution. It is easy to see that a binary string x of length k with n ones has complexity $K(x|n,k) \leq \log \binom{k}{n} + O(1)$, and $K(x|n,k) \geq \log \binom{k}{n} - O(1)$ for most such x. Hence, a string x with maximal $K(x|n,k)$ will pass this universal sum test. Each individual such string possesses all effectively testable properties of typical strings under the Fermi–Dirac distribution.

It is known that photons, nuclei, and some other elementary particles behave according to the Bose–Einstein distribution. Here, we distribute n particles in k cells, where each cell may contain many particles. Let the set of possible arrangements be $B(n,k)$. All possible arrangements are equally likely. By Exercise 1.3.6, the probability of each arrangement x under the Bose–Einstein distribution is $BE_{n,k}(x) = 1/d(B(n,k))$, where

$$d(B(n,k)) = \binom{k+n-1}{n} = \binom{k+n-1}{k-1}.$$

Similar to Example 4.3.9, use Theorem 4.3.5 to obtain a universal sum test with respect to the Bose–Einstein distribution:

$$\begin{aligned}\kappa_0(x|BE_{n,k}) &= \log \frac{\mathbf{m}_{B(n,k)}(x)}{BE_{n,k}(x))} \\ &= -K(x|n,k) + \log d(B(n,k)) + O(1).\end{aligned}$$

◇

Example 4.3.10 *Markov's inequality* says the following: Let P be a probability mass function; let f be a nonnegative function with P-expected value $\mathbf{E} = \sum_x P(x)f(x) < \infty$. Then, $\sum\{P(x) : f(x)/\mathbf{E} > k\} < 1/k$.

Let P be any probability distribution (not necessarily computable). The P-expected value of $\mathbf{m}(x)/P(x)$ is (ignoring the x's for which $P(x) = 0$)

$$\sum_x P(x)\frac{\mathbf{m}(x)}{P(x)} \leq 1.$$

Then, by Markov's inequality,

$$\sum_x \{P(x) : \mathbf{m}(x) \leq kP(x)\} \geq 1 - \frac{1}{k}. \tag{4.7}$$

Since $\mathbf{m}$ dominates all lower semicomputable semimeasures multiplicatively, we have for all x,

$$P(x) \leq c_P \mathbf{m}(x), \text{ with } c_P = 2^{K(P)}. \tag{4.8}$$

Equations 4.7 and 4.8 have the following consequences:

1. If x is a random sample from a simple computable distribution P, where 'simple' means that $K(P)$ is small, then $\mathbf{m}$ is a good estimate for P. For instance, if x is randomly drawn from distribution P, then the probability that

   $$c_P^{-1}\mathbf{m}(x) \leq P(x) \leq c_P\mathbf{m}(x)$$

   is at least $1 - 1/c_P$.

2. If we know or believe that x is random with respect to P, and we know $P(x)$, then we can use $P(x)$ as an estimate of $\mathbf{m}(x)$.

In both cases the degree of approximation depends on the index of P and the randomness of x with respect to P, as measured by the randomness deficiency $\kappa_0(x|P) = \log(\mathbf{m}(x)/P(x))$. For example, the uniform discrete distribution on $\mathcal{B}^*$ can be defined by $L(x) = 2^{-2l(x)-1}$. Then for each n we have $L_n(x) = L(x|l(x) = n)$. To describe L takes $O(1)$ bits, and therefore

$$\kappa_0(x|L) = l(x) - K(x) + O(1).$$

The randomness deficiency $\kappa_0(x|L)$ is $O(1)$ iff $K(x) \geq l(x) - O(1)$, that is, iff x is random. ◇

Example 4.3.11 The incomputable distribution $\mathbf{m}(x) = 2^{-K(x)}$ has the remarkable property that the test $\kappa_0(x|\mathbf{m})$ is $O(1)$ for all x: The test shows all outcomes x to be random with respect to it.

As an aside, $-\sum_x P(x)\kappa_0(x|P) = D(P \parallel \mathbf{m})$ is the Kullback–Leibler divergence between distributions P and $\mathbf{m}$, Equation 1.17 on page 72. This is a measure of how close $\mathbf{m}$ is to P. Rewriting, we see that $D(P \parallel \mathbf{m}) = \sum_x P(x)K(x) + \sum_x P(x)\log P(x) + O(1) = \sum_x P(x)K(x) - H(P) + O(1)$. This quantity is the difference between the expected prefix complexity and the entropy, the latter being the minimum possible P-expected prefix code-word length, Theorem 1.11.2 on page 77. In Section 8.1.1 this difference is shown to be bounded by $K(P)$, the complexity of describing distribution P, and in Section 1.10 a continuous version is used to show that the universal distribution is a good predictor for distributions of small complexity.

We can interpret Equations 4.7, 4.8 as saying that if the real distribution is P, then $P(x)$ and $\mathbf{m}(x)$ are close to each other with large P-probability. Therefore, if x comes from some unknown computable distribution P, then we can use $\mathbf{m}(x)$ as an estimate for $P(x)$. In other words, $\mathbf{m}(x)$ can be viewed as the universal a priori probability of x.

The universal sum P-test $\kappa_0(x|P)$ can be interpreted in the framework of hypothesis testing as the likelihood ratio between hypothesis P and the

fixed alternative hypothesis $\mathbf{m}$. In ordinary statistical hypothesis testing, some properties of an unknown distribution P are taken for granted, and the role of the universal test can probably be reduced to some tests that are used in statistical practice. ◇

4.3.6 Randomness by Universal Gambling

We toss a fair coin a hundred times and it shows heads every time. The argument that a hundred heads in a row is just as probable as any other outcome convinces us only that the axioms of probability theory do not solve all mysteries as they are supposed to. We feel that a sequence consisting of a hundred heads is not due to pure chance, while some other sequences with the same probability are.

Example 4.3.12 In some innominate country with a ruling party and free elections, the share of votes for the ruling party is $x_i.y_i\%$ in thirty successive elections, with $x_i \geq 50$ and y_i is the ith digit in the decimal expansion of $\pi = 3.1415\ldots$, $i = 0, 1, \ldots, 29$. However, if we complain about this, the election organizers tell us that some sequence has to come up, and the actual outcome is as likely as any other. We cannot criticize a regularity we discover *after the fact*, but only those regularities that we have excluded in advance. ◇

In probability theory one starts with the assumption that we have a sample space S of outcomes, with a probability distribution P. This P is either discovered empirically or simply hypothesized, for instance by analogy to similar processes or considerations of symmetry. It is customary to call properties that hold with P-probability one 'laws of probability.'

Consider a Bernoulli process $(\frac{1}{2}, \frac{1}{2})$ such as the repeated tossing of a fair coin. Each outcome x is an infinite sequence of zeros and ones. It is customary to predict that a random x will have each property that holds with probability one. But x cannot be predicted to have all such properties. To see this, consider the property of belonging to the complement of a given singleton set. Each such property has probability one, but jointly they have probability zero. That is, a random outcome x cannot be expected to withstand *all* statistical tests chosen afterward together. But we can expect x to satisfy a few standard laws, such as the law of large numbers, and presume them always chosen. However, the classical theory of probability gives us no criteria for selection of such standard laws.

Kolmogorov's solution is to select those randomness properties with probability close to one that are 'simply expressible.' The objects that do not satisfy such a property have a corresponding regularity and form a simply described set of small measure and correspondingly small cardinality. Then each such object is simply described by the set it is an element of and its position in that set. This allows substitution of the multiple requirement of "satisfying all regularities involved" by a single requirement of "not being a simple object." In the betting approach we place a single bet that gives a huge payoff in case the outcome is not complex, and which thereby safeguards us against all simple ways of cheating.

Definition 4.3.10 A nonnegative function $t : \mathcal{N} \to \mathcal{R}$ is a *P-payoff function* if

$$\sum_{x \in \mathcal{N}} P(x)t(x) \leq 1.$$

The definition says that the logarithm of a lower semicomputable P-payoff function is a sum P-test as in Definition 4.3.8. Among the lower semicomputable payoff functions there is a universal payoff function that incorporates all particular payoff functions: a universal betting strategy.

Definition 4.3.11 A lower semicomputable P-payoff function $t_0 : \mathcal{N} \to \mathcal{R}$ is (P-)*universal* if it multiplicatively dominates each lower semicomputable P-payoff function t (that is, $t(x) = O(t_0(x))$).

Lemma 4.3.6 *Let P be a computable probability distribution. The function $t_0(x|P) = \mathbf{m}(x)/P(x)$ is a universal lower semicomputable P-payoff function.*

Proof. Each lower semicomputable P-payoff function t can be expressed as $t(x) = 2^{\delta(x)}$ with δ a sum P-test, Definitions 4.3.8, 4.3.10. Since $t_0(x|P) = 2^{\kappa_0(x|P)}$ with κ_0 the universal sum P-test of Theorem 4.3.5, and κ_0 dominates each δ additively, it follows that t_0 dominates each t multiplicatively. □

Betting Against a Crooked Player

Suppose you meet a street gambler tossing a coin and offering odds to all passers-by on whether the next toss will be heads 1 or tails 0. He offers to pay you two dollars if the next toss is heads; you pay him one dollar if the next toss is tails. Should you take the bet? If the gambler is tossing a fair coin, it is a great bet. Probably you will win money in the long run. After all, you can expect that half of the tosses will come up heads and half tails. Losing only one dollar on each heads toss and getting two for each tails makes you rich fast. After some observation you notice that the sample sequence of outcomes looks like 01010101010 . . . . Perhaps the gambler manipulates the outcomes. Expecting foul play, you make the following offer as a bet for 1,000 coin tosses.

You pay \$1 first and propose that your opponent pays you $2^{1000-K(x)}$ dollars, with x the binary sequence of outcomes of the 1,000 coin flips. This is better than fair, since the gambler is expected to pay only

$$\sum_{l(x)=1000} 2^{-1000} 2^{1000-K(x)} < \$1,$$

by Kraft's inequality. So he should be happy to accept the proposal. But if the gambler cheats, then, for example, you receive $2^{1000-\log 1000}$ dollars for a sequence like 01010101010 . . .!

In the 1 versus 2 dollars scheme, you can also propose to add this as an extra bonus pay. In this way, you are guaranteed to win big: either polynomially increase your money (when the gambler does not cheat) or exponentially increase your money (when the gambler cheats).

Example 4.3.13 Suppose a gambler proposes the following wager to the election organizers in Example 4.3.12. He will bet one dollar in each election. The organizers claim that each outcome x associated with n elections has probability $L_n(x)$, where $L_n(x) = 10^{-n}$ is the uniform distribution on decimal strings of length n and zero otherwise. We formulate a payoff function that is a winning strategy against all simply describable malversations. To back up their claim, the organizers ought to agree to pay $t(x)$ dollars on outcome x on any payoff function t we propose. Namely, the expected amount of payoff is at most the gambler's total original wager of n dollars. Accordingly, we propose as payoff function $t = t_0(\cdot|L_n)$, the universal payoff function with respect to L_n defined by

$$t_0(x|L_n) = 2^{-\log L_n(x) - K(x|n)} = 2^{n \log 10 - K(x|n)}.$$

If x consists of the first n digits of the decimal expansion of π, the election organizers have to pay the gambler the staggering amount of

$$2^{n \log 10 - K(\pi_{1:n}|n)} \geq c10^n$$

dollars for some fixed constant c independent of n, even though the bet did not refer to π. Even if the organizers are smart and switch to some pseudorandom sequence algorithmically generated by their computer, they will have to pay such an amount. In other words, since we propose the payoff function beforehand, it is unlikely that we define precisely the one that detects a particular fraud. However, fraud implies regularity, and the number of regularities is so small that we can afford to make a combined wager on all of them in advance. ◇

The fact that $t_0(x|P)$ is a payoff function implies by Markov's inequality, Equation 4.7 on page 282, that for every $k > 0$ we have,

$$\sum_x \left\{ P(x) : K(x) \geq \log \frac{1}{P(x)} - k \right\} \geq 1 - \frac{1}{2^k}. \tag{4.9}$$

By Equation 4.2 and Theorem 4.3.3, for all x,

$$K(x) \leq \log \frac{1}{P(x)} + K(P) + O(1). \tag{4.10}$$

Setting $k := K(P)$, we find that with large probability, the complexity $K(x)$ of a random outcome x is close to its upper bound $\log 1/P(x) +$

$O(K(P))$. If an outcome x violates any 'law of probability,' then the complexity $K(x)$ falls far below the upper bound. Indeed, a proof of some law of probability like the law of large numbers or the law of the iterated logarithm always gives rise to some simple computable payoff function $t(x)$ taking large values on the outcomes violating the law.

We can phrase the relations as follows: Since the payoff function $t_0(\cdot|P)$ dominates all P-payoff functions that are lower semicomputable, $\kappa_0(\cdot|P)$ is a universal test of randomness—it measures the deficiency of randomness in the outcome x with respect to distribution P, or the extent of justified suspicion against hypothesis P given the outcome x.

Exercises

4.3.1. [12] Show that $\sum_x 2^{-K(x|y)} \leq 1$.

Comments. Hint: use the Kraft inequality, Theorem 1.11.1.

4.3.2. [15] Show that the class of computable measures does not contain a universal element.

4.3.3. [21] Show that the greatest monotonic nonincreasing lower bound on the universal distribution $\mathbf{m}$ (universal lower semicomputable discrete semimeasure) converges to zero more slowly than the greatest nonincreasing monotonic lower bound on any positive recursive function that goes to zero in the limit.

4.3.4. [28] Show that the universal distribution $\mathbf{m}$ has infinite entropy: $H(\mathbf{m}) = \sum_x \mathbf{m}(x) \log 1/\mathbf{m}(x) = \infty$, where the summation is over all $x \in \{0,1\}^*$.

Comments. Hint: by the coding theorem, Theorem 4.3.3 on page 273, it suffices to show that $\sum_x 2^{-K(x)}K(x) = \sum_n \sum_{l(x)=n} 2^{-K(x)}K(x) = \infty$. This follows because there are at least 2^{n-1} strings x of length n with $n-1 \leq K(x) \leq n + 2\log n + O(1)$.

4.3.5. [08] Show that if $K(x) \leq \log x$ then $\sum_{i=1}^{x} 2^{-K(i)} \leq x2^{-K(x)}$.

4.3.6. [32] We study the statistics of description length. By the coding theorem, Theorem 4.3.3, we have $K(x) = \log 1/Q_U(x)$ up to an additive constant. Informally, if an object has many long descriptions, then it also has a short one.

(a) Let $f(x,n)$ be the number of binary strings p of length n with $U(p) = x$, where U is the reference prefix machine of Theorem 3.1.1, page 202. Show that for all $n \geq K(x)$, we have $\log f(x,n) = n - K(x,n) + O(1)$.

(b) Use Item (a) to show that $\log f(x, K(x)) = K(x) - K(x, K(x)) + O(1) = O(1)$. The number of shortest programs of any object is bounded by a universal constant.

Comments. Hint: in Item (b), use $K(x) \leq K(x,n) + O(1)$; substitute $n = K(x)$ in the expression in Item (a) to obtain $\log f(x, K(x)) = K(x) - K(x, K(x)) + O(1) \leq O(1)$. Source: P. Gács, *Lecture Notes on Descriptional Complexity and Randomness*, Manuscript, Boston University, 1987.

4.3.7. [19] Show that $\sum_{l(x)=n} \mathbf{m}(x) = \mathbf{m}(n)$, up to a fixed multiplicative constant.

Comments. Source: P. Gács, *Ibid.*

4.3.8. [18] Give an example of a recursive sequence of rational numbers $a_n > 0$ such that the sum $\sum_n a_n$ is finite, but for each other recursive (or lower semicomputable) sequence $b_n > 0$, if $b_n/a_n \to \infty$ then $\sum_n b_n = \infty$.

Comments. Hint: Let r_n be a recursive increasing sequence of rational numbers with $\lim_n r_n = \sum_x \mathbf{m}(x)$ and let $a_n = r_{n+1} - r_n$. Source: P. Gács, *Ibid.*

4.3.9. [13] Prove the following: There exists a constant c such that for every k and l, if a string x has at least 2^l programs of length k, then $C(x|l) \leq k - l + c$.

Comments. Therefore, $C(x) \leq k - l + 2\log l + c$. So if the x has complexity k and there are 2^l shortest programs for x (programs of length k) then $k \leq k - l + 2\log l + c$, so that $l - 2\log l < c$ and l is bounded. Source: A.K. Shen, Kolmogorov mailing list, June 24, 2002.

4.3.10. [29] We can also express statistics of description length with respect to C. For every lower semicomputable function f with $\{f(k) : k \geq 1\}$ satisfying the Kraft inequality, there exist fewer than $2^{k+f(k)+O(1)}$ programs of length $C(x) + k$ for x.

Comments. Hint: consider a machine that assigns a code of length m to x iff x has at least $2^{k+f(k)}$ programs of length $m + k$. Then the number of strings that are assigned a code of length m is at most $\sum_k (2^{m+k}/2^{k+f(k)}) = \sum_k 2^{m-f(k)}$, which by Kraft's inequality is at most 2^m. Hence, this is a valid upper semicomputable code. Since x has no program of length less than $C(x)$, the string x has fewer than $2^{k+f(k)+O(1)}$ programs of length $C(x)+k$. Source: J.T. Tromp, personal communication, March 13, 1991.

4.3.11. [39] How many objects are there of a given complexity n? Let $g(n)$ be the number of objects x with $K(x) = n$, and let D_n be the set of binary strings p of length n such that $U(p)$ is defined. Define the moving average $h(n,c) = 1/(2c+1) \sum_{i=-c}^{c} g(n+i) + O(1)$.

(a) First show that $\sum_y \mathbf{m}(x,y) = \mathbf{m}(x) + O(1)$.

(b) Show that there is a natural number c such that $\log d(D_n) = n - K(n) + O(1)$ and also $\log h(n,c) = n - K(n)$.

Comments. Hint for Item (b): use Exercise 4.3.6 and Item (a). Since we are interested in equality only up to an additive constant, we can omit the normalizing factor $1/(2c+1)$ from the definition of h. But we do not know whether we can replace h by g. Namely, one can choose a reference prefix machine U' such that $g'(n) = 0$ for all odd n. For instance, $U'(00p) = U(p)$ of $l(p)$ is even, $U'(1p) = U(p)$ for $l(p)$ is odd, and $U'(p)$ is undefined otherwise. Then U' is defined only for inputs of even length, and for all x we have $K_{U'}(x) \leq K(x) + 2$. Source: R.M. Solovay, *Lecture Notes*, 1975, unpublished; and P. Gács, *Lecture Notes on Descriptional Complexity and Randomness*, Manuscript, Boston University, 1987.

4.3.12. [32] Suppose we want to obtain information about a certain object x. It is not a good policy to guess blindly. The mutual information of two objects x and y was given in Example 3.9.2 on page 252 as $I(x;y) = K(y) - K(y|x, K(x))$. Show that $\sum_y \mathbf{m}(y)2^{I(x;y)} = O(1)$.

Comments. In words, the expected value of $2^{I(x;y)}$ is small, even with respect to the universal distribution $\mathbf{m}(x)$. Hint: by the coding theorem, Theorem 4.3.3, we have $2^{I(x;y)} = 2^{-K(y|x,K(x))}/\mathbf{m}(x) + O(1)$. Source: P. Gács, *Lecture Notes on Descriptional Complexity and Randomness*, Manuscript, Boston University, 1987.

4.3.13. [34] Let $X = x_1, x_2, \ldots$ be a recursive sequence of natural numbers (in $\mathcal{N}$ or the corresponding binary strings). The *lower frequency* of some element x in the sequence is defined as

$$q_X(x) = \liminf_{n\to\infty} \frac{1}{n} d(\{i : i < n \text{ and } x_i = x\}).$$

(a) Show that there is a *universal recursive sequence* $U = u_1, u_2, \ldots$ such that for every recursive sequence $X = x_1, x_2, \ldots$ there is a constant $c > 0$ such that $cq_U(x) \geq q_X(x)$, for all x in $\mathcal{N}$.

(b) Show that if U and V are universal recursive sequences, then $q_U(x) = \Theta(q_V(x))$. Fix a *reference* universal recursive sequence U, and define the *a priori frequency* of x as $\mathbf{q}(x) = q_U(x)$.

(c) Show that $\mathbf{q}(x) \neq \Theta(\mathbf{m}(x))$. ($\mathbf{m}(x)$ is the *a priori probability* of x.)

(d) A set is *enumerable relative* to $0'$ if it is the range of a function computable by an algorithm with an oracle for some recursively enumerable set. An algorithm with an oracle for set A is an algorithm that (apart from the usual things) at each step can ask a question of the form, "is a in A?" and get the true answer "yes/no" for free. The recursively enumerable sets correspond to the $0'$-enumerable sets, where the oracle is restricted to recursive sets. Define the notion of $0'$-*enumerable semimeasures*, and show that there is a *universal* $0'$-enumerable semimeasure $\mathbf{p}$ such that for each $0'$-enumerable semimeasure ν there is a constant $c > 0$ such that $\mathbf{p}(x) \geq c\nu(x)$. We call $\mathbf{p}$ the *a priori probability relative to* $0'$.

(e) Show that $\mathbf{p}(x) = \Theta(\mathbf{q}(x))$. Compare this with Item (c).

Comments. Source: An.A. Muchnik, *SIAM Theory Probab. Appl.,* 32 (1987), 513–514; A.N. Kolmogorov, V.A. Uspensky, *SIAM Theory Probab. Appl.*, 32(1987), 389–412.

4.3.14. [43] (a) Show that the minimal length of a program enumerating a set A (prints all elements of A in lexicographic length-increasing order and no other elements; we do not require halting in case A is finite) is bounded above by 3 times the negative logarithm of the probability that a random program enumerates A. That is, the probability that if the input to the reference universal prefix machine is determined by flips of a fair coin, then the output is an enumeration of A.

(b) Show that the constant 3 in Item (a) can be reduced to 2 for finite sets A.

Comments. Source: for Item (a) R.M. Solovay, *Non-Classical Logics, Model Theory and Computability*, A.I. Aruda, N.C.A. da Costa and R. Chaqui, eds., North-Holland, 1977, 283–307; and for Item (b) N.K. Vereshchagin, *Inform. Process. Lett.*, 103:1(2007), 34–37.

4.4 Universal Average-Case Complexity

The universal distribution $\mathbf{m}$ is one of the foremost notions in the theory of Kolmogorov complexity. It multiplicatively dominates all lower semicomputable distributions (and therefore also all computable ones). Therefore, a priori it maximizes ignorance by assigning maximal probability to all objects. It has many remarkable properties and applications. Here we observe that the average-case computational complexity of *any algorithm whatsoever* under the universal distribution turns out to be of the same order of magnitude as the worst-case complexity. This holds both for time complexity and for space complexity.

For many algorithms the average-case running time under some distributions on the inputs is less than the worst-case running time. For instance, using (nonrandomized) Quicksort on a list of n items to be sorted gives under the uniform distribution on the inputs an average running time of $O(n \log n)$, while the worst-case running time is $\Omega(n^2)$. The worst-case running time of Quicksort is typically reached if the list is already sorted or almost sorted, that is, exactly in cases in which we actually should not have to do much work at all. Since in practice the lists to be sorted occurring in computer computations are often sorted or almost sorted, programmers often prefer other sorting algorithms that might run faster with almost sorted lists. Without loss of generality we identify inputs of length n with the natural numbers corresponding to binary strings of length n.

Definition 4.4.1 Consider a discrete sample space $\mathcal{N}$ with probability density function P. Let $t(x)$ be the running time of algorithm A on problem instance x. Define the *worst-case time complexity* of A as $T(n) = \max\{t(x) : l(x) = n\}$. Define the *$P$-average time complexity* of A as

$$T(n|P) = \frac{\sum_{l(x)=n} P(x)t(x)}{\sum_{l(x)=n} P(x)}.$$

Example 4.4.1 **(Quicksort)** We compare the average time complexity for Quicksort under the uniform distribution $L(x)$ and under the universal distribution $\mathbf{m}(x)$. Define $L(x) = 2^{-2l(x)-1}$ such that the conditional probability satisfies $L(x|l(x) = n) = 2^{-n}$. We encode the list of elements to be sorted as nonnegative integers in some standard way.

For Quicksort, $T(n|L) = \Theta(n \log n)$. We may expect the same complexity under $\mathbf{m}$, that is, $T(n|\mathbf{m}) = \Omega(n \log n)$. But Theorem 4.4.1 will tell us much more, namely, $T(n|\mathbf{m}) = \Omega(n^2)$. Let us give some insight into why this is the case.

With the low average time complexity under the uniform distribution, there can be only $o((\log n)2^n/n)$ strings x of length n with $t(x) = \Omega(n^2)$. Therefore, given n, each such string can be described by its sequence number in this small set, and hence for each such x we obtain $K(x|n) \leq n - \log n + 3 \log \log n$. (Since n is known, we can find each $n - k$ by coding k self-delimiting in $2 \log k$ bits. The inequality follows by setting $k \geq \log n - \log \log n$.)

Therefore, no really random x's, with $K(x|n) \geq n$, can achieve the worst-case running time $\Omega(n^2)$. Only strings x that are nonrandom, with $K(x|n) < n$, among which are the sorted or almost sorted lists, and lists exhibiting other regularities, can have $\Omega(n^2)$ running time. Such lists x have relatively low Kolmogorov complexity $K(x)$, since they are regular (can be compactly described), and therefore $\mathbf{m}(x) = 2^{-K(x)+O(1)}$ is very high. Therefore, the contribution of these strings to the average running time is weighted very heavily. ◇

Theorem 4.4.1 **(m-Average Complexity)** *Let A be an algorithm with inputs in $\mathcal{N}$. Let the inputs to A be distributed according to the universal distribution $\mathbf{m}$. Then the average-case time complexity is of the same order of magnitude as the corresponding worst-case time complexity.*

Proof. We define a probability distribution $P(x)$ on the inputs that assigns high probability to the inputs for which the worst-case complexity is reached, and zero probability for other cases.

Let A be the algorithm involved. Let $T(n)$ be the worst-case time complexity of A. Clearly, $T(n)$ is recursive (for instance by running A on all x's of length n). Define the probability distribution $P(x)$ as follows:

Step 1. **For** each $n = 0, 1, \ldots$, set $a_n := \sum_{l(x)=n} \mathbf{m}(x)$.

Step 2. **If** $l(x) = n$ and x is lexicographically least with $t(x) = T(n)$ **then** $P(x) := a_n$ **else** $P(x) := 0$.

It is easy to see that a_n is lower semicomputable, since $\mathbf{m}(x)$ is lower semicomputable. Therefore, $P(x)$ is lower semicomputable. Below we use $c_P \mathbf{m}(x) \geq P(x)$, where $\log c_P = K(P) + O(1)$ is a constant depending on P but not on x, Theorem 4.3.1 on page 267 and Example 4.3.3 on page 269. We have defined $P(x)$ such that $\sum_{x \in \mathcal{N}} P(x) = \sum_{x \in \mathcal{N}} \mathbf{m}(x)$, and $P(x)$ is a lower semicomputable probability distribution. The average-case time complexity $T(n|\mathbf{m})$ with respect to the $\mathbf{m}$ distribution on the inputs is now obtained by

$$\begin{aligned} T(n|\mathbf{m}) &= \sum_{l(x)=n} \frac{\mathbf{m}(x)t(x)}{\sum_{l(x)=n} \mathbf{m}(x)} \\ &\geq \frac{1}{c_P} \sum_{l(x)=n} \frac{P(x)}{\sum_{l(x)=n} \mathbf{m}(x)} T(n) \\ &= \frac{1}{c_P} \sum_{l(x)=n} \frac{P(x)}{\sum_{l(x)=n} P(x)} T(n) = \frac{1}{c_P} T(n). \end{aligned}$$

The inequality $T(n) \geq T(n|\mathbf{m})$ holds vacuously. □

Corollary 4.4.1 The analogue of the theorem holds for other complexity measures (such as *space* complexity) by about the same proof.

If the algorithm to approximate $P(x)$ from below is the kth algorithm in the standard effective enumeration of all algorithms, then $\log c_P = K(P) + O(1) < k \log^2 k$. To approximate the optimal value we must code the algorithm to compute P as compactly as possible. The ease with which we can describe (algorithmically) the strings that produce a worst-case running time determines the closeness of the average time complexity to the worst-case time complexity. Let $S \subseteq \{0,1\}^n$. Denote by $T(n|P, S)$ the P-average computation time as in Definition 4.4.1 but with the average taken over S instead of $\{0,1\}^n$ as with $T(n|P)$.

Lemma 4.4.1 *Let Q be a computable probability distribution. There is a set S of inputs with $Q(S) \geq 1 - 2^{-k}$ such that $T(n|Q, S) \geq T(n|\mathbf{m}, S)/2^{K(Q)+k+O(1)}$.*

Proof. If the probability distribution Q is lower semicomputable (which by Example 4.3.2 on page 266 means that it is computable), then by Markov's inequality, Equation 4.7 on page 282, and $2^{K(Q)+O(1)}\mathbf{m}(x) \geq Q(x)$, substitution in Definition 4.4.1 restricted to S proves the lemma. □

Example 4.4.2 **(Quicksort continued)** The average time complexity of Quicksort with the inputs distributed according to the uniform distribution is of order $n \log n$. By Lemma 4.4.1 and Theorem 4.4.1 we find (for simplicity ignoring $O(1)$ factors) that the same holds in the computational reality for a set of inputs of combined L-probability at least $1 - 2^k$ as long as

$$n \log n \geq n^2 / 2^{K(L)+K(P)+k}.$$

Here, L is the uniform distribution substituted for Q in Lemma 4.4.1 (generated by a program of length at least $K(L)$) and P is the particular distribution used in the proof of Theorem 4.4.1. For k large and n so large that $K(L) + K(P) + k < \log n - \log \log n$, the square average-case running time must take over. Clearly, increasing $K(L)$ (more complex algorithmic random number generator) increases the size of n at which the square running time starts to take over.

Frequently, algorithmically generated random numbers are used in order to reduce the average-case computation time to below the worst-case computation time. The above example gives evidence that for every input length, only sufficiently complex algorithmic random number generators can achieve reduced average-case computation time. ◇

Example 4.4.3 In learning applications in Section 5.3.3 we want to draw elements from the **m** distribution. Since **m** is not computable, we can't have a program for it. Suppose some powerful source deems it fit to give us a table with sufficiently many **m** values. We use this table to randomly draw according to **m** as follows:

Our prospective algorithm has access to an **m** table in the form of a division of the real open interval $[0, 1)$ into nonintersecting half-open subintervals I_x such that $\bigcup I_x = [0, 1)$. For each x, the length of interval I_x is $\mathbf{m}(x)/\sum_y \mathbf{m}(y)$. For each finite binary string r, the *cylinder* Γ_r is the set of all infinite binary strings starting with r. That is, Γ_r is a half-open interval $[0.r, 0.r + 2^{-l(r)})$ in $[0, 1)$. To draw a random example from **m**, the algorithm uses a sequence $r_1 r_2 \ldots$ of outcomes of fair coin flips until the cylinder Γ_r, $r = r_1 r_2 \ldots r_k$, is contained in some interval I_x. It is easy to see that this procedure of selecting x, using a table for **m** and fair coin flips, is equivalent to drawing an x randomly according to distribution **m**.

We are often interested in drawing an element of a subset D of $\mathcal{N}$. For instance, we want to draw an n-length binary vector ($D = \{0, 1\}^n$) when learning Boolean functions. To draw from $\mathbf{m}(\cdot|D)$, we simply draw examples from $\mathbf{m}(\cdot)$ and discard the ones not in D. If we need to draw m examples according to $\mathbf{m}(\cdot|D)$, then it suffices to draw $\Omega(2^{K(D)}m)$

examples under $\mathbf{m}(\cdot)$. Namely, for each $x \in D$,

$$\mathbf{m}(x|D) = \mathbf{m}(x)\frac{\sum_{y\in\mathcal{N}} \mathbf{m}(y)}{\sum_{y\in D} \mathbf{m}(y)} = \Theta(2^{K(D)}\mathbf{m}(x)).$$

◇

Exercises

4.4.1. [12] Show that the $\mathbf{m}$-average time complexity of Quicksort is $\Omega(n^2)$.

Comments. Source: M. Li and P.M.B. Vitányi, *Inform. Process. Lett.*, 42(1992), 145–149.

4.4.2. [12] Show that for each NP-complete problem, if the problem instances are distributed according to $\mathbf{m}$, then the average running time of any algorithm that solves it is superpolynomial unless P = NP.

Comments. Source: M. Li and P.M.B. Vitányi, *Ibid.*

4.5 Continuous Sample Space

Is there a universal lower semicomputable semimeasure in the continuous setting? In the discrete version we had only to satisfy that the probabilities summed to less than or equal one. Here we have to deal with the additional subadditive property. Let $\mathcal{B}$ be the finite set of basic elements. In the following we sometimes take $\mathcal{B} = \{0, 1\}$ for convenience, but this is not essential.

4.5.1 Universal Enumerable Semimeasure

The development of the theory for continuous semimeasures is quite similar to that for discrete semimeasures, except that the analogue of the coding theorem, Theorem 4.3.3, does not hold. Let $\mathcal{M}$ be a class of continuous semimeasures as in Definition 4.2.1. The definition of universal continuous semimeasure is analogous to Definition 4.3.2 for the discrete case.

Definition 4.5.1 A semimeasure μ_0 is *universal* (or maximal) for $\mathcal{M}$ if $\mu_0 \in \mathcal{M}$, and for all $\mu \in \mathcal{M}$, there exists a constant $c > 0$ such that for all $x \in \mathcal{B}^*$, we have $\mu_0(x) \geq c\mu(x)$.

Theorem 4.5.1 *There is a universal lower semicomputable continuous semimeasure. We denote it by* $\mathbf{M}$.

Proof. We prove the theorem in two stages. In Stage 1 we show that the lower semicomputable semimeasures can be effectively enumerated as

$$\mu_1, \mu_2, \ldots \ .$$

In Stage 2 we show that

$$\mu_0(x) = \sum_{j \geq 1} \alpha(j)\mu_j(x), \text{ with } \sum \alpha(j) \leq 1,$$

is a universal semimeasure. Stage 1 is broken up into two parts. In the first part we enumerate all lower semicomputable functions; and in the second part we effectively change the lower semicomputable functions to lower semicomputable semimeasures, leaving the functions that were already semimeasures unchanged.

STAGE 1 Let $\psi_1, \psi_2, \ldots$ be the effective enumeration of all lower semicomputable (real-valued) functions. Fix any ψ (we drop the subscript for notational convenience). Without loss of generality we can assume (as we have done before in similar cases) that we are actually dealing with rational-valued two-argument partial recursive functions $\phi(x, k) = p/q$ (rather $\phi(\langle x, k\rangle) = \langle p, q\rangle$) such that for all $x \in \mathcal{B}^*$, for all $k > 0$,

- if $\phi(x, k)$ is defined, then for all $y \leq x$ ($\leq$ in the sense of the natural lexicographic length-increasing order on $\mathcal{B}^*$), $\phi(y, 1), \ldots, \phi(y, k-1)$ are all defined;
- $\phi(x, k+1) \geq \phi(x, k)$;
- $\lim_{k\to\infty} \phi(x, k) = \psi(x)$;
- $\psi(x) > \phi(x, k)$ for every k. (This is achieved by replacing $\phi(x, k)$ by $\phi(x, k) := \phi(x, k)/(1 + 1/k)$. This replacement affects neither the monotonicity of ϕ nor the represented semimeasure—if any.)

Next we use each ϕ associated with ψ to compute a semimeasure μ by approximation from below. In the algorithm, at each stage of the computation the local variable μ contains the current approximation to the function μ. This is doable because the nonzero part of the approximation is always finite.

We describe a sequence of lower semicomputable semimeasures $\psi_k(x)$ computed from $\phi(x, k)$ such that if $\phi(x, k)$ represents a lower semicomputable semimeasure $\psi(x)$, then $\lim_{k\to\infty} \psi_k(x) = \psi(x)$.

Step 1. Initialize by setting $\mu(x) := \psi_k(x) := 0$, for all x in $\mathcal{B}^*$ and $k \in \mathcal{N}$; and set $k := 0$.

Step 2. Set $k := k + 1$. Compute $\phi(x, k)$ and set $\psi_k(x) := \phi(x, k)$ for all $x \in \mathcal{B}^k$. {If the computation does not terminate, then μ will not change any more and is trivially a semimeasure}

Step 3. **For** $i := k - 1, k - 2, \ldots, 0$ **do**

for each x of length i **do**

search for the least $K > k$ such that $\phi(x, K) > \sum_{b \in \mathcal{B}} \psi_k(xb)$;

set $\psi_k(x) := \phi(x, K)$;

if $\psi_k(\epsilon) \leq 1$ **then** $\mu := \psi_k$ **else** terminate.

{Step 3 tests whether the new values in Step 2 satisfy the semimeasure requirements; note that if ψ is a lower semicomputable semimeasure, then the K's always exist, since for each x we have $\sum_{b \in \mathcal{B}} \psi_k(xb) < \sum_{b \in \mathcal{B}} \psi(xb) \leq \psi(x)$ and $\lim_{k \to \infty} \phi(x, k) = \psi(x)$}

Step 4. **Go to** Step 2.

Since ϕ represents $\psi(x)$, by monotonicity of ϕ we have $\psi_k(x) \geq \phi(x, k)$ for all x of length at most k, which implies $\lim_{k \to \infty} \psi_k(x) = \psi(x)$. If ψ is already a semimeasure, then $\mu := \psi$ and the algorithm never finishes but continues to approximate μ from below. If for some k and $x \in \mathcal{B}^k$ the value of $\phi(x, k)$ is undefined, then the values of μ do not change any more even though the computation of μ goes on forever. If the condition in Step 3 is violated, then the algorithm terminates, and the constructed μ is a semimeasure—even a computable one. Clearly, in all cases, μ is a lower semicomputable semimeasure.

The current construction was suggested by J. Tyszkiewicz [personal communication of April 1996] and assumes a finite set $\mathcal{B}$ of basic elements. It can be made to handle $\mathcal{B} = \mathcal{N}$ if in the construction of ψ_k one considers and gives possibly nonzero measures to only sequences of length at most k and consisting of natural numbers $\leq k$.

Executing the above procedure on all functions in the list $\phi_1, \phi_2, \ldots$ yields an effective enumeration $\mu_1, \mu_2, \ldots$ of all lower semicomputable semimeasures.

STAGE 2 Let $\alpha : \mathcal{N} \to \mathcal{R}$ be any lower semicomputable function satisfying $\alpha(j) > 0$ for all j and $\sum_j \alpha(j) \leq 1$. Define the function μ_0 from $\mathcal{B}^*$ into $[0, 1)$ as

$$\mu_0(x) = \sum_j \alpha(j) \mu_j(x).$$

We show that μ_0 is a universal lower semicomputable semimeasure. The first condition in Definition 4.2.1 of being a semimeasure is satisfied,

since

$$\mu_0(\epsilon) = \sum_j \alpha(j)\mu_j(\epsilon) \leq \sum_j \alpha(j) \leq 1.$$

The second condition in Definition 4.2.1 of being a semimeasure is satisfied, since, for all x in $\mathcal{B}^*$,

$$\mu_0(x) = \sum_j \alpha(j)\mu_j(x) \geq \sum_j \alpha(j) \sum_{b\in\mathcal{B}} \mu_j(xb) = \sum_{b\in\mathcal{B}} \mu_0(xb).$$

The function μ_0 is lower semicomputable, since the $\mu_j(x)$'s are lower semicomputable in j and x. (Use the universal partial recursive function ϕ_0 and the construction above.)

Finally, μ_0 multiplicatively dominates each μ_j, since $\mu_0(x) \geq \alpha(j)\mu_j(x)$. Therefore, μ_0 is a universal lower semicomputable semimeasure. There is more than one such universal lower semicomputable semimeasure. We fix a *reference* universal lower semicomputable semimeasure μ_0 and denote it by $\mathbf{M}$. □

The universal lower semicomputable semimeasure $\mathbf{M}(x)$ captures the notion of a universal a priori probability needed for application to inductive reasoning (Chapter 5).

Above we can set $\alpha(j) = 2^{-j}$. But we can also choose $\alpha(j) = 2^{-K(j)}$. For $\mu = \mu_j$ we can define $K(\mu) = K(j)$. Therefore,

$$\mathbf{M}(x) \geq 2^{-K(\mu)}\mu(x), \tag{4.11}$$

for all $x \in \mathcal{B}^*$.

At the risk of beating a dead horse (Example 4.3.1), we belabor the distinction between 'continuous semimeasure' and 'discrete semimeasure,' and the relation between $\mathbf{M}$ and $\mathbf{m}$.

The discrete sample space theory is simply a restriction of the more sophisticated continuous approach we take in this section. Theorem 4.5.1 is a lifted version of Theorem 4.3.1. Namely, if we set $\mathcal{B} = \mathcal{N}$ and restrict the arguments of the measure functions to sequences of natural numbers of length one, and incorporate the resulting simplifications in the proof of Theorem 4.5.1, then we obtain the proof of Theorem 4.3.1, and instead of $\mathbf{M} : \mathcal{B}^* \to \mathcal{R}$, we obtain its discrete version $\mathbf{m} : \mathcal{B} \to \mathcal{R}$.

Lemma 4.5.1 *If a continuous lower semicomputable semimeasure is a measure, it is computable.*

Proof. Let μ be a lower semicomputable semimeasure with $\sum_{b\in\mathcal{B}} \mu(xb) = \mu(x)$ for all $x \in \mathcal{B}^*$ and $\mu(\epsilon) = 1$. Then, we can approximate all $\mu(x)$ to

any degree of precision starting with $\mu(a), \mu(b), \ldots$ ($\mathcal{B} = \{a, b, \ldots, z\}$) and determining $\mu(x)$ for all x of length n, for consecutive $n = 1, 2, \ldots$. □

Lemma 4.5.2 *The set of computable continuous semimeasures has no universal element.*

Proof. Set $\mathcal{B} = \mathcal{N}$. If there is a universal computable continuous semimeasure, then its restriction to domain $\mathcal{B}$ would by definition be a universal computable discrete semimeasure, contradicting Lemma 4.3.1. The case $2 \leq d(\mathcal{B}) < \infty$ is left to the reader. □

Lemma 4.5.3 *The function* $\mathbf{M}$ *is not computable and* $\mathbf{M}$ *is not a probability measure.*

Proof. If $\mathbf{M}$ were computable, then it would be universal for the class of computable continuous semimeasures, by Theorem 4.5.1. This contradicts Lemma 4.5.2.

For $\mathbf{M} : \mathcal{B}^* \to \mathcal{R}$ we prove $\sum_{b \in \mathcal{B}} \mathbf{M}(b) < 1$ by the same proof of Lemma 4.3.2 (with $\mathbf{M}$ instead of $\mathbf{m}$). □

4.5.2 A Priori Probability

As in the discrete case, we can interpret the lower semicomputable semimeasures in a different way. To do so we require an appropriate new Turing machine variant.

Definition 4.5.2 *Monotone machines* are Turing machines with a one-way read-only input tape, some work tapes, and a one-way write-only output tape. The input tape contains a one-way infinite sequence of 0's and 1's, and initially the input head scans the leftmost bit. The output tape is written one symbol in $\mathcal{B}$ at a time, and the output is defined as the finite binary sequence on the output tape if the machine halts, and the possibly infinite sequence appearing on the output tape in a never-ending process if the machine does not halt. For a (possibly infinite) sequence x we write $M(p) = x$ if M outputs x after reading p and no more. (Machine M either halts or computes forever without reading additional input.)

We define a sample space $S_{\mathcal{B}}$ consisting of all finite and infinite sequences over $\mathcal{B}$:

$$S_{\mathcal{B}} = \mathcal{B}^* \bigcup \mathcal{B}^\infty.$$

Definition 4.5.3 Monotone machines compute partial functions $\psi : \{0,1\}^* \to S_{\mathcal{B}}$ such that for all $p, q \in \{0,1\}^*$ we have that $\psi(p)$ is a prefix of $\psi(pq)$. The function ψ induces a mapping $\psi' : \{0,1\}^\infty \to S_{\mathcal{B}}$ as follows: Let $\omega = \omega_1\omega_2 \ldots \in \{0,1\}^\infty$.

Case 1 $\psi'(\omega)$ is the infinite sequence $\zeta = \zeta_1\zeta_2\ldots$ (with the ζ_i's strings), provided for each n, there is an m such that $\psi(\omega_{1:n}) = \zeta_{1:m}$, and m goes to infinity with n.

Case 2 If for some n, $\psi(\omega_{1:n})$ is defined, and for all $m > n$ we have $\psi(\omega_{1:m})$ equals $\psi(\omega_{1:n})$, then $\psi'(\omega) = \psi(\omega_{1:n})$. If for all n, $\psi(\omega_{1:n})$ is the empty word ϵ, then $\psi'(\omega) = \epsilon$.

Case 3 If there is an n such that $\psi(\omega_{1:n})$ is undefined, then $\psi'(\omega)$ is undefined.

We call such functions ψ' *monotone functions*. For convenience we drop the prime on the extension ψ' from now on.

Definition 4.5.4 A monotone machine M maps subsets of $\{0,1\}^\infty$ to subsets of $S_{\mathcal{B}}$. If ψ is the function computed by M, and $A \subseteq \{0,1\}^\infty$, then define

$$\psi(A) = \{x \in S_{\mathcal{B}} : \psi(\omega) = x, \omega \in A\}.$$

A *cylinder* set Γ_x of $S_{\mathcal{B}}$ is defined as

$$\Gamma_x = \{x\omega : \omega \in S_{\mathcal{B}}\},$$

with $x \in \mathcal{B}^*$. Each semimeasure μ is transformed by a monotone machine M, computing a monotone function ψ, to μ_ψ (also a semimeasure) as follows: For each $x \in \mathcal{B}^*$, let $X \subseteq \{0,1\}^*$ be the set of y's such that $\psi(y) \in \Gamma_x$. Then M maps $\bigcup_{y\in X}\Gamma_y \subseteq \{0,1\}^\infty$ to $\Gamma_x \subseteq S_{\mathcal{B}}$. It is possible that for some $y, z \in X$ we have $\Gamma_y \bigcap \Gamma_z \neq \varnothing$. This is the case precisely if y is a proper prefix of z or conversely. That is, either Γ_y is contained in Γ_z or vice versa. To obtain the total μ-measure of $\bigcup_{y\in X}\Gamma_y \subseteq \{0,1\}^\infty$, we sum the μ-measures of all the constituent cylinders that are not contained in other constituent cylinders. This is done by restricting X to a subset Y obtained from X by eliminating all strings that have a proper prefix in X and summing over the cylinders associated with elements in Y. The probability $\mu_\psi(x)$ that M computes a sequence starting with x on μ-random input from $\{0,1\}^\infty$ is given by

$$\mu_\psi(x) = \sum_{y\in Y} \mu(y). \tag{4.12}$$

Clearly, one can effectively enumerate all monotone machines $M_1, M_2, \ldots$ and therefore the associated monotone functions $\psi_1, \psi_2, \ldots$ they compute. We show that the corresponding enumeration $\mu_{\psi_1}, \mu_{\psi_2}, \ldots$, with the μ_ψ's defined as in Equation 4.12, is an enumeration of all and only lower semicomputable measures.

Definition 4.5.5 A monotone function ψ is *μ-regular* if the set of sequences $\omega \in \{0,1\}^\infty$ for which $\psi'(\omega)$ is defined by Case 1 of Definition 4.5.3 has μ-measure one. In other words, except for a set of negligible probability, $\psi(\omega)$ is defined by Case 1.

Lemma 4.5.4 (i) *For each computable measure μ and each μ-regular monotone function ψ, we have that μ_ψ is a computable measure as well.*

(ii) *For each computable measure μ there exists a λ-regular ψ, with λ the uniform measure, such that $\lambda_\psi = \mu$; moreover, there is a μ-regular ϕ such that $\mu_\phi = \lambda$ and ϕ is the inverse of ψ in the domain of definition of $\phi\psi$, and $\phi(\omega) \in \mathcal{B}^\infty$ except for possibly the computable ω's and ω's lying in countably many intervals of μ-measure 0.*

Proof. (i) For each $x \in \mathcal{B}^*$ and n we must be able to approximate $\mu_\psi(x)$ within accuracy 2^{-n}. Choose m such that

$$\sum\{\mu(y) : l(y) = m, l(\psi(y)) > l(x)\} > 1 - 2^{-(n+1)}. \tag{4.13}$$

Such an m exists, since ψ is μ-regular, and it is easy to find such an m effectively.

Let $Z \subseteq Y$ (with Y as in Equation 4.12) consist of the $y \in Y$ that additionally satisfy the condition in Equation 4.13. Since by assumption μ is a computable measure, the $\mu(y)$'s can be computed to within an accuracy of $2^{-(m+n+1)}$. Let $\hat{\mu}(y)$ be such an approximation, and let $\alpha(x,n) = \sum_{y\in Z} \hat{\mu}(y)$. There are at most 2^m many y's satisfying the condition in Equation 4.13, and therefore

$$|\mu_\psi(x) - \alpha(x,n)| < 2^{-(n+1)} + 2^m \cdot 2^{-(m+n+1)} = 2^{-n}.$$

(ii) Omitted. □

For the less-nice cases, namely, the lower semicomputable semimeasures, the following property is easy to prove:

Lemma 4.5.5 *If μ is a lower semicomputable semimeasure and ψ a monotone function, then μ_ψ is again a lower semicomputable semimeasure.*

Proof. By enumerating the semimeasure μ (from below) for all its arguments x, and dovetailing the computations of ψ for all these arguments as well, we approximate μ_ψ from below using Equation 4.12. Therefore, μ_ψ is lower semicomputable. It is straightforward to verify that μ_ψ is also a semimeasure. □

The theorem below shows that μ is a lower semicomputable semimeasure iff it can be obtained as the semimeasure on the output sequences of

some monotone machine whose input is supplied by independent tosses of a fair coin. In other words, each lower semicomputable semimeasure $\mu : \mathcal{B}^* \to \mathcal{R}$ can be obtained from some monotone function ψ and the uniform measure λ, in the sense of Equation 4.12:

$$\mu(x) = \lambda_\psi(x) = \sum_{y \in Y} 2^{-l(y)},$$

for the prefix-free set of programs y such that $\psi(y)$ starts with x.

Theorem 4.5.2 *A semimeasure μ is lower semicomputable if and only if there is a monotone function ψ such that $\mu = \lambda_\psi$, where λ is the uniform measure.*

Proof. (If) By Lemma 4.5.5.

(Only if) We construct a monotone function ψ such that $\mu = \lambda_\psi$, where $\lambda(x) = 2^{-l(x)}$ is the uniform measure (which is computable). For this construction, we have to decompose the interval $[0, 1)$ into nonintersecting sets of measure $\mu(x)$. We will represent $x \in \mathcal{B}^*$ by a set of intervals $\Phi(x)$ of $[0, 1)$ with the property that when x is a prefix of y then

$$\bigcup_{z \in \Phi(x)} \Gamma_z \supseteq \bigcup_{z \in \Phi(y)} \Gamma_z,$$

$$\mu(x) = \lambda \left(\bigcup_{z \in \Phi(x)} \Gamma_z \right).$$

This can be achieved incrementally. Analogous to the construction in the proof of the coding theorem, Theorem 4.3.3, since μ is lower semicomputable, there is a rational-valued recursive function $\phi(x, k)$, nondecreasing in k for fixed x, such that $\lim_{k \to \infty} \phi(x, k) = \mu(x)$. By definition, $\mu(x) \geq \sum_{b \in \mathcal{B}} \mu(xb)$. Without loss of generality, we can assume that $\phi(x, k) \geq \sum_{b \in \mathcal{B}} \phi(xb, k)$, for all t. (Whenever this inequality is not satisfied, we can decrease the $\phi(xb, k)$'s proportionally to the extent that the inequality becomes valid, without compromising the limiting behavior of ϕ for fixed x and t going to infinity.) To obtain $\mu(x)$ we approximate it by successive representations $\Phi_1(x), \Phi_2(x), \ldots$ such that $\Phi_k(x)$ is a prefix-free set with

$$\lambda \left(\bigcup_{z \in \Phi_k(x)} \Gamma_z \right) = \sum_{y \in \Phi_k(x)} 2^{-l(y)} = \phi(x, k),$$

satisfying the following: If x is a prefix of y, then

$$\bigcup_{z \in \Phi_k(x)} \Gamma_z \supseteq \bigcup_{z \in \Phi_k(y)} \Gamma_z.$$

If x and y are incomparable, then

$$\left(\bigcup_{z\in\Phi_k(x)}\Gamma_z\right)\bigcap\left(\bigcup_{z\in\Phi_k(y)}\Gamma_z\right)=\varnothing.$$

If $k<k'$, then

$$\bigcup_{z\in\Phi_{k'}(x)}\Gamma_z\supseteq\bigcup_{z\in\Phi_k(y)}\Gamma_z.$$

The construction of the Φ_k's is straightforward. Hence, $\lambda_\psi(x)=\mu(x)$. □

Definition 4.5.6 Let U be the reference monotone machine. Denote the function computed by U by ψ. The *universal a priori probability* that a binary sequence starts with x is $\lambda_\psi(x)$, with λ_ψ in the sense of Equation 4.12 (μ replaced by the uniform measure λ).

It turns out that the universal a priori probability and the universal lower semicomputable semimeasure are *equal*, just as in the discrete case (Theorem 4.3.3). If we provide U's input by tosses of a fair coin, then the probability that U's output starts with x is given by $\mathbf{M}(x)$. This $\mathbf{M}$ is a central concept in this area.

Theorem 4.5.3 $\log 1/\lambda_U(x)=\log 1/\mathbf{M}(x)+O(1)$.

Proof. This follows from the fact that for each monotone machine M in the effective enumeration $M_1,M_2,\ldots$, we have that $U(1^{n(M)}0p)=M(p)$. Namely, this shows that $\lambda_U(x)\geq 2^{-(n(M)+1)}\lambda_M(x)$. The λ_M's contain all lower semicomputable semimeasures by Theorem 4.5.2. Since λ_U multiplicatively dominates each λ_M, it qualifies as the universal lower semicomputable semimeasure $\mathbf{M}$. □

Corollary 4.5.1 If the monotone machine T defines the lower semicomputable semimeasure μ, then $\mathbf{M}(x)\geq 2^{-K(\mu)}\mu(x)$ with $K(\mu)$ the shortest self-delimiting description of T.

4.5.3 *Solomonoff Normalization

We have chosen to develop the theory essentially along lines reminiscent of Martin-Löf's theory of tests for randomness: we view it as mathematically elegant that the universal element, dominating all elements in a given class, belongs to that class. While our basic goal was to obtain a universal measure in the class of computable measures, satisfying the above criteria turned out to be possible only by weakening both the notion of measure and the notion of effective computability. Another

path was taken by R.J. Solomonoff. He viewed the notion of measure as sacrosanct. He normalized each semimeasure μ to a measure using a particular transformation.

Definition 4.5.7 The *Solomonoff normalization* of a semimeasure μ on the sample space $\mathcal{B}^\infty$ is defined by

$$\mu_{\rm norm}(\epsilon) = 1,$$
$$\mu_{\rm norm}(\omega_{1:n}b) = \mu_{\rm norm}(\omega_{1:n})\frac{\mu(\omega_{1:n}b)}{\sum_{a\in\mathcal{B}}\mu(\omega_{1:n}a)},$$

for all $n \in \mathcal{N}$ and $b \in \mathcal{B}$.

Write the (unnormalized) semimeasure μ as

$$\mu(\omega_{1:n}) = \prod_{i=1}^{n}\mu(\omega_i|\omega_{1:i-1}),$$

where we set $\omega_{1:0} = \epsilon$. Define a new symbol u, a special undefined element, with the property that $\mu(u|\omega_{1:n}) = 1 - \sum_{a\in\mathcal{B}}\mu(a|\omega_{1:n})$. To obtain $\mu_{\rm norm}$ from μ, we multiply the ith factor of μ written as the product above by

$$\frac{1}{1-\mu(u|\omega_{1:i-1})} = \frac{\mu(\omega_{1:i-1})}{\sum_{a\in\mathcal{B}}\mu(\omega_{1:i-1}a)}.$$

Then,

$$\begin{aligned}\mu_{\rm norm}(\omega_{1:n}) &= \frac{\mu(\omega_{1:n})}{\mu(\epsilon)}\prod_{i=1}^{n}\frac{1}{1-\mu(u|\omega_{1:i-1})} \\ &= \frac{\mu(\omega_{1:n})}{\mu(\epsilon)}\prod_{i=1}^{n}\frac{\mu(\omega_{1:i-1})}{\sum_{a\in\mathcal{B}}\mu(\omega_{1:i-1}a)}.\end{aligned} \tag{4.14}$$

It is straightforward to verify that if μ is a semimeasure, then $\mu_{\rm norm}$ is a measure. We call $\mathbf{M}_{\rm norm}$ the *Solomonoff measure*. It is at once clear that $\mathbf{M}_{\rm norm}$ dominates all lower semicomputable semimeasures as well. This comes at the price that $\mathbf{M}_{\rm norm}$ is *not* lower semicomputable itself by Lemmas 4.5.1, 4.5.2. Nonetheless, for many applications (as in Chapter 5) it may be important to have a universal *measure* available with most of the right properties, and the fact that it is not lower semicomputable may not really bother us. Another possible objection is that there are more ways to normalize a semimeasure to a measure. So why choose this one? The choice for $\mathbf{M}_{\rm norm}$ is not unique. Solomonoff justifies this choice by his particular interest in the interpretation of $\mathbf{M}$ as a

priori probability: the measure $\mathbf{M}(x)$ is the probability that the monotone reference machine outputs a sequence starting with x if its input is supplied by fair coin flips.

Suppose the output of the machine is known thus far, say x. One wants to know the relative probability that 1 rather than 0 will be the next symbol when we know that there is a next symbol. This relative probability is $P = \mathbf{M}(x1)/\mathbf{M}(x0)$. Many, if not most, applications of probability involve ratios of this sort. For example, this is the case in computing conditional probabilities, relative probabilities of various scientific hypotheses, and mathematical decision theory. If one wants to divide up $\mathbf{M}(xu)$ (the unnormalized probability that the machine does *not* print another symbol after emitting x), then only the normalization above leaves the ratio P invariant. The ratio P is sacred since it happens to be given by the reference universal monotone machine. The justification for $\mathbf{M}_{\rm norm}$ lies in the fact that it is the unique measure such that $\mathbf{M}_{\rm norm}(x1)/\mathbf{M}_{\rm norm}(x0) = \mathbf{M}(x1)/\mathbf{M}(x0)$.

This normalization eliminates all probability concentrated on the finite sequences and divides all $\mu(x)$'s by the corresponding remaining sums. In this way, we obtain an object related to the greatest measure contained in the semimeasure. Another possibility would be to use the unnormalized conditional probability $P_{\rm unnorm} = \mathbf{M}(x1)/\mathbf{M}(x)$ instead of $P_{\rm norm} = \mathbf{M}(x1)/(\mathbf{M}(x0) + \mathbf{M}(x1))$, the normalized conditional probability. Using $P_{\rm unnorm}$ is equivalent to considering xu to be a real possibility. In most (if not all) applications we have the *auxiliary information* that either $x0$ or $x1$ *did* occur, so the probability of xu is zero. In cases of this sort, $P_{\rm norm}$ is correct and $P_{\rm unnorm}$ is incorrect as values for the conditional probability of 1, given that x has occurred. Solomonoff argues (somewhat weakly) that since such probability ratios *need* to be used in applications, we somehow have to find ways to deal with them, and we cannot just refuse to accept normalized probability because it is not lower semicomputable. This tentative section is based on R.J. Solomonoff [*IEEE Trans. Inform. Theory*, IT-24(1978), 422–432]; discussions with L.A. Levin, P. Gács, and Solomonoff; and in good part on Solomonoff's arguments [Letter, October 4, 1991]. Every manner of normalizing $\mathbf{M}$ replaces $\mathbf{M}(x) - (\mathbf{M}(x0) + \mathbf{M}(x1))$ by 0, and Exercise 4.5.6 on page 325 shows that $1/\mathbf{M}(x)$ may be vastly different from $1/(\mathbf{M}(x0) + \mathbf{M}(x1))$. Therefore, the normalization greatly distorts the relation between $\mathbf{M}(x)$ and $\mathbf{M}(x0) + \mathbf{M}(x1)$ for some x. But Exercise 4.5.7 on page 325 tells us that if we use $\mathbf{M}$ to predict consecutive symbols of sequences actually distributed according to a computable measure μ, then this rarely happens. Namely, for every n the μ-expected lack of measurehood of $\mathbf{M}$ satisfies

$$\sum_{l(x)=n} \mu(x) \sum_{i=1}^{n} \frac{\mathbf{M}(x_{i-1}) - \mathbf{M}(x_{i-1}0) - \mathbf{M}(x_{i-1}1)}{\mathbf{M}(x_{i-1})} \leq K(\mu) \ln 2.$$

See also the discussion in 'History and References,' Section 4.7.

4.5.4 *Monotone Complexity and a Coding Theorem

There are two possibilities for associating complexities with machines. The first possibility is to take the length of the shortest program, while the second possibility is to take the negative logarithm of the universal probability. In the discrete case, using prefix machines, these turned out to be the same by the coding theorem, Theorem 4.3.3. In the continuous case, using monotone machines, it turns out that they are different.

Definition 4.5.8 The complexity KM is defined as

$$KM(x) = \log \frac{1}{\mathbf{M}(x)}.$$

In contrast with C and K complexities, in the above definition the greatest prefix-free subset of *all* programs that produce output starting with x on the reference monotone machine U is weighed.

Definition 4.5.9 Let U be the reference monotone machine. The complexity Km, called *monotone complexity*, is defined as

$$Km(x) = \min\{l(p) : U(p) = x\omega, \omega \in S_{\mathcal{B}}\}.$$

We omit the invariance theorems for KM complexity and Km complexity, stated and proven completely analogously to Theorems 2.1.1, 3.1.1 on pages 105, 202. By definition, $KM(x) \leq Km(x)$. In fact, all complexities coincide up to a logarithmic additive term, Section 4.5.5.

It follows directly from Definition 4.5.8 and Equation 4.11 that for each lower semicomputable semimeasure μ we have

$$KM(x) \leq \log \frac{1}{\mu(x)} + K(\mu).$$

The following theorem can be viewed as a coding theorem, continuous version, for computable measures. It states that for computable measures μ, each sample has a minimal description bounded by the logarithm of its probability plus a constant.

Theorem 4.5.4 *Let $\mu : \{0,1\}^* \to \mathcal{R}$ be a computable measure. Then*

$$Km(x) \leq \log \frac{1}{\mu(x)} + c_\mu,$$

where c_μ is a constant depending on μ but not on x.

Proof. Recall the terminology in the proof of Lemma 4.3.3 on page 274. We give an inductive procedure for assigning code words to samples starting with x (rather, the cylinder Γ_x consisting of finite and infinite

sequences starting with x). Start by setting the interval $I_\epsilon := [0, 1)$ (of length $\mu(\epsilon) = 1$).

Inductively, divide the interval I_x into two half-open intervals I_{x0} (the left one) and I_{x1} (the right one), of lengths $\mu(x0)$ and $\mu(x1)$, respectively. In each interval I_x determine the length of the largest binary interval. Since μ is a real-valued computable function, we cannot achieve that the lengths of the intervals I_{x0} and I_{x1} are exactly $\mu(x0)$ and $\mu(x1)$, but we can achieve a good rational approximation (good enough that even the product of the error factors is bounded).

Let the binary string representing the leftmost such interval be r. Select as code for a sample starting with x the code word r. Note that samples starting with x and $x0$ may be represented by the same code word. This is fine. According to this procedure, each sample of strings starting with x gets assigned a code word r with $l(r) \leq \log 1/\mu(x) + 2$ (analogous to the proof of Lemma 4.3.3).

By construction the code is monotone: if r is a code word for a sample starting with x, then so is each code word starting with r. Since also μ is computable, there is a monotone machine M as follows: If r is a code word for a sample starting with x, then M recovers x as prefix of an output string x' (possibly longer than x) by following the above procedure. For each prefix q of r it computes the longest y such that q is a code for a sample starting with y. Since the code is monotone, M can operate monotonically as well, by outputting subsequent digits of x' for each additional digit read from the one-way input r.

If $n(M)$ is the index of M in the standard enumeration of monotone machines, then input $\overline{n(M)}r$ is a program for the reference monotone machine U to output a string starting with x. Hence, the length $Km(x)$ of the shortest program for U for outputting a string with prefix x satisfies $Km(x) \leq \log 1/\mu(x) + c_\mu$ with $c_\mu = 2n(M) + 2$. Refinement of the argument lets us set $c_\mu = K(M) + 2$. □

Theorem 4.5.4 is the continuous analogue for the coding theorem, Theorem 4.3.3, concerning discrete lower semicomputable semimeasures. We know that $KM(x) \leq Km(x) + O(1)$. It has been shown that equality does not hold: the difference between $KM(x)$ $(= \log 1/\mathbf{M}(x))$ and $Km(x)$ is very small, but still rises unboundedly. This contrasts with the equality between $\log 1/\mathbf{m}(x)$ and $K(x)$ in Theorem 4.3.3. Intuitively, this phenomenon is justified by exposing the relation between $\mathbf{M}$ and $\mathbf{m}$.

The coding theorem states that $K(x) = \log 1/\mathbf{m}(x) + O(1)$. L.A. Levin [*Soviet Math. Dokl.*, 14(1973), 1413–1416] conjectured that the analogue would hold for the unrestricted continuous version. But it has been shown that

$$\sup_{x \in \mathcal{B}^*} |KM(x) - Km(x)| = \infty$$

[P. Gács, *Theoret. Comput. Sci.*, 22(1983), 71–93]. In particular (for each particular choice of basis $\mathcal{B}$ such as $\mathcal{B} = \mathcal{N}$, the natural numbers, or $\mathcal{B} = \{0, 1\}$),

$$KM(x) \leq Km(x) \leq KM(x) + K(l(x)) + O(1). \tag{4.15}$$

The left-hand inequality follows from the definitions. The right-hand inequality can be understood as follows: First look at $\mathcal{B} = \mathcal{N}$. The coding theorem can be proved for every recursively enumerable prefix-free subset of $\mathcal{N}^*$, and not only for $\mathcal{N}$. (It is enough to prove it for $\mathcal{N}$; the extension to prefix-free sets follows by encoding.) That is, if F is a recursively enumerable prefix-free subset of $\mathcal{N}^*$ (such as $\mathcal{N}^n$ or another cut through the prefix tree representation of $\mathcal{N}^*$), then $Km_F(x) = \log 1/\mathbf{m}_F(x)$. (Here, the subscript F means that both Km and $\mathbf{m}$ are defined with respect to an effective enumeration of monotone machines with programs in F.)

Now, $Km(x) \leq Km_F(x) + K(n(F)) + O(1)$, where we need a self-delimiting description of $K(n(F))$ additional bits to describe $n(F)$, the index of the partial recursive function describing F. Since one can show that $\mathbf{m}_F(x) \geq \mathbf{M}(x)$, we have proved the right-hand inequality of Equation 4.15. Similar reasoning can improve the estimate to

$$KM(x) \leq Km(x) \leq KM(x) + K(KM(x)) + O(1).$$

These results show that differences between $Km(x)$ and $KM(x)$ with $\mathcal{B} = \mathcal{N}$ cannot exceed those accounted for by the tree structure of $\mathcal{N}^*$. As stated before, the problem is equivalent for the binary basis $\mathcal{B} = \{0, 1\}$. Moreover, the estimate is the best possible one, since the following can be shown:

Claim 4.5.1 Let $\mathcal{B} = \mathcal{N}$. For every upper semicomputable function $g : \mathcal{N} \to \mathcal{N}$ for which

$$Km(x) - KM(x) \leq g(l(x)),$$

we have $Km(n) \leq g(n) + O(1)$.

Proof. We refer to the proof in [P. Gács, *Theoret. Comput. Sci.*, 22(1983), 71–93]. □

Note that for upper semicomputable functions $g(n)$, the statement $Km(n) \leq g(n) + O(1)$ is equivalent to $\sum_n 2^{-g(n)} < \infty$. Namely, on the one hand it follows from $KM(x) \leq Km(x) + O(1)$ that

$$\sum_n 2^{-Km(n)} \leq \sum_n \mathbf{M}(n) \leq \mathbf{M}(\epsilon) \leq 1.$$

Therefore, $Km(n) \leq g(n) + O(1)$ implies that $\sum_n 2^{-g(n)} < \infty$. Conversely, if $\sum_n 2^{-g(n)} < \infty$ then on the domain $\mathcal{N}$ we have that $2^{-g(n)}$ is multiplicatively dominated by $\mathbf{M}(n)$. Since on the domain $\mathcal{N}$ we have $KM(n) = Km(n)+O(1)$, it follows that $Km(n) \leq KM(n) + O(1) \leq g(n) + O(1)$ for $n \in \mathcal{N}$.

This shows that the differences between $Km(x)$ and $KM(x)$ must in some sense be very small. The next question to ask is whether the quantities involved are usually different, or whether this is a rare occurrence, in other words, whether for almost all infinite sequences ω, the difference between Km and KM is bounded by a constant. The following facts have been proven [P. Gács, *Theoret. Comput. Sci.*, 22(1983), 71–93].

Lemma 4.5.6 (i) *For random strings $x \in \mathcal{B}^*$ we have $Km(x) - KM(x) = O(1)$.*

(ii) *There exists a function $f(n)$ that goes to infinity as $n \to \infty$ such that $Km(x) - KM(x) \geq f(l(x))$, for infinitely many x.*

(a) *If x is a finite string of natural numbers ($\mathcal{B} = \mathcal{N}$), then we can choose $f(n) = \log n$.*

(b) *If x is a finite binary string ($\mathcal{B} = \{0,1\}$), then we can choose $f(n)$ as the inverse of some version of Ackermann's function (Exercise 1.7.18 on page 45).*

(c) *For almost all infinite ω, where 'almost all' is specified according to the universal lower semicomputable semimeasure, the difference $Km(\omega_{1:n}) - KM(\omega_{1:n})$ has an upper bound that is smaller than any unbounded recursive function.*

(d) *For almost all infinite ω, where 'almost all' is specified according to any computable measure, $Km(\omega_{1:n}) - KM(\omega_{1:n})$ is bounded by a constant.*

4.5.5 *Relation Between Complexities

Let $\mathcal{B} = \{0,1\}$. There are five complexity variants discussed in this book: the plain complexity $C(\cdot)$, the complexity $KM(\cdot) = \log 1/\mathbf{M}(\cdot)$ associated with the universal measure $\mathbf{M}$, the monotone complexity $Km(\cdot)$, Loveland's uniform complexity $C(\cdot;\cdot)$ of Exercise 2.3.2 on page 130, and the prefix complexity $K(\cdot)$. What is the quantitative difference between them? Partially, such as between $C(\cdot)$, $l(\cdot)$, and $K(\cdot)$, these relations follow from Chapters 2, 3. It is easy to see that $Km(x) \leq l(x) + O(1)$, and by definition, $KM(x) \leq Km(x)$. Moreover, $C(x; l(x)) \leq C(x) + O(1)$ by Exercise 2.3.2.

We include the table of relations, Figure 4.2, and omit the proofs. A table entry expresses bounds on the complexity naming the column minus the complexity naming the row. Since we cannot give the difference precisely, we split it into an upper bound that always holds and a bound that is exceeded infinitely often. Thus, every entry consists of an upper

	$C(x)$	$KM(x)$	$Km(x)$	$K(x)$
$l(x) = n$	$\leq O(1)$	$\leq O(1)$	$\leq O(1)$	$\ell^k(n,\epsilon)$
				$\ell^k(n,0)$
$C(x)$		$\ell^k(n,\epsilon)$	$\ell^k(n,\epsilon)$	$\ell^k(n,\epsilon)$
		$\ell^k(n,0)$	$\ell^k(n,0)$	$\ell^k(n,0)$
$KM(x)$	$\log n + O(1)$		$\ell^k(n,\epsilon)$	$\ell^k(n,\epsilon)$
	$\log n - O(1)$		$A^{-1}(n)$	$\ell^k(n,0)$
$Km(x)$	$\log n + O(1)$	< 0		$\ell^k(n,\epsilon)$
	$\log n - O(1)$	< 0		$\ell^k(n,0)$
$C(x;n)$	$\ell^k(n,\epsilon)$	$\ell^k(n,\epsilon)$	$\ell^k(n,\epsilon)$	$\log n + \ell^k(n,\epsilon)$
	$\ell^k(n,0)$	$\ell^k(n,0)$	$\ell^k(n,0)$	$\log n + \ell^k(n,0)$

FIGURE 4.2. Relations between five complexities with $l(x) = n$

entry and a lower entry. The upper entry gives an upper bound on the difference for *all* x. That is, the difference is always at most this bound. Thus, $C(x) - l(x) \leq O(1)$ (possibly nonconstantly negative) for every x. The lower entry gives a bound on the difference that is reached or exceeded for *infinitely many* x. That is, the difference reaches or exceeds this bound infinitely often. Thus, $C(x) - KM(x) \geq \log l(x) - O(1)$ for infinitely many x. We denote $l(x)+l(l(x))+\cdots+(1+\epsilon)l^k(x)$ by $\ell^k(x,\epsilon)$, where $l^1(x) = l(x)$ and $l^k(x) = l(l^{k-1}(x))$ for $k > 1$. The bound $\ell^k(x,\epsilon)$ holds for every fixed k and $\epsilon > 0$, and the bound $\ell^k(x,0)$ holds for every fixed k.

The difference $Km(x)-KM(x)$ is especially interesting. As noted above, this difference can be arbitrarily large, and hence $\log 1/\mathbf{M}(x) \neq Km(x)+O(1)$ (no coding theorem in this case). In particular, $Km(x)-KM(x) > A^{-1}(l(x))$ for infinitely many x. The function A^{-1} is a version of the very slow-growing inverse of the Ackermann function in Exercise 1.7.18 on page 45. (Interestingly, while A grows faster than any primitive recursive function, its inverse A^{-1} is primitive recursive.) The only known upper bound $Km(x)-KM(x) \leq \ell^k(l(x),\epsilon)$ holds for every x and k, and follows from Equation 4.15 on page 307. The difference between these bounds is large, and open for improvement. For background material, see the 'History and References,' Section 4.7.

4.5.6 *Randomness by Integral Tests

The randomness of infinite sequences with respect to the uniform measure has been treated in Sections 2.5 and 3.6. In the latter section, Corollary 3.6.1, page 223, gives the following exact expression characterizing the infinite sequences that are random with respect to the uniform measure λ:

$$\rho_0(\omega|\lambda) = \sup_{n\in\mathcal{N}}\{n - K(\omega_{1:n})\}. \tag{4.16}$$

That is, $\rho_0(\omega|\lambda) < \infty$ iff ω is random with respect to λ. We generalize this result to exact expressions testing the randomness of infinite sequences for arbitrary computable measures μ.

A universal sequential μ-test $\delta_0(\cdot|\mu)$ of Section 2.5 additively majorizes all other sequential μ-tests. It distinguishes the random infinite sequences from the nonrandom ones. However, we did not find an exact expression of the universal sequential μ-test in terms of complexity. But we can develop other types of tests that do have exact expressions in terms of Kolmogorov complexity for a universal test separating the random infinite sequences from the nonrandom ones.

Sequential μ-tests were essentially functions of a continuous variable in $S = \mathcal{B}^\infty$. Constructivity of such functions of infinite sequences was ensured by having a sequential test approximate its value by taking the supremum of all tests of finite initial segments of the infinite sequence.

We start by considering the notion of lower semicomputable unit integrable functions of a continuous variable. It is easy to show that the logarithms of such functions are slightly stronger versions of sequential tests. This parallels the development of the sum tests of Definition 4.3.8, page 278. Subsequently, we give the exact expression of a universal lower semicomputable unit integrable function with respect to a computable measure μ. The logarithm of this quantity is the exact expression for a universal test. This approach requires the application of a few elementary properties of integration.

Let $\mathcal{B} = \{0, 1, \ldots, k-1\}$ be a finite nonempty alphabet with $k \geq 2$. The continuous variable ranges over the set of one-way infinite sequences $S = \mathcal{B}^\infty$. The set S has the power of the continuum, since it can be mapped onto the set $\mathcal{R}$ of real numbers, Example 1.4.1. The set X below is a subset of S.

Definition 4.5.10 A nonnegative function $f : S \to \mathcal{R}$ is *unit integrable* (over a set X with respect to measure μ) if

$$\int_X f(\omega)\mu(d\omega) \leq 1.$$

Consider recursive functions of the form $g(x, k) = \langle r, s\rangle$. The interpretation is that g has an argument $x \in \mathcal{B}^*$ and a rational value r/s. The cylinder set Γ_x is the set of infinite sequences over $\mathcal{B}$ starting with x. This leads to a refinement of Definition 1.7.4 on page 35 of semicomputability:

Definition 4.5.11 A nonnegative function $f : S \to \mathcal{R}$ is *lower semicomputable* if there exists a recursive function $g(x, k)$ satisfying $g(x, k+1) \geq g(x, k)$ and $g(xy, k) \geq g(x, k)$ such that

$$f(\omega) = \sup_{\omega \in \Gamma_x,\, k \in \mathcal{N}} \{g(x, k)\}.$$

(Furthermore, f is upper semicomputable iff $-f$ is lower semicomputable, and f is computable iff it is both lower semicomputable and upper semicomputable.) We determine the relation between a lower semicomputable unit integrable function and a sequential test.

Lemma 4.5.7 *If f is a lower semicomputable unit integrable function over S with respect to a computable measure μ, then $\log f(\cdot)$ is a sequential μ-test. Moreover, if δ is a sequential μ-test, then the function f defined by $\log f(\omega) = \delta(\omega) - 2\log\delta(\omega) - c$ is a lower semicomputable unit integrable function.*

Proof. Let f be a lower semicomputable unit integrable function. Without loss of generality we can assume that the values $f(\omega)$ are of the form 2^m. Define

$$\gamma(x) = \inf_{\omega \in \Gamma_x} \lfloor \log f(\omega) \rfloor .$$

Since f is lower semicomputable, so is γ, even though its definition uses inf, by the way we defined lower semicomputable functions. Also, $\gamma(\omega_{1:n})$ is monotonic in n. It follows that $\log f(\omega) = \sup_{n \in \mathcal{N}} \{\gamma(\omega_{1:n})\}$. Moreover, for each n,

$$\sum \left\{ \mu(x) 2^{\gamma(x)} : l(x) = n, \gamma(x) > k \right\} \leq 2^{-k}.$$

Otherwise,

$$\int_S f(\omega)\mu(d\omega) > \sum_{l(x)=n} \mu(x) 2^{\gamma(x)} \geq \sum_{l(x)=n} \mu(x) 2^k \geq 1.$$

Hence, f is a sequential μ-test according to Definition 2.5.1, page 147.

Conversely, assume that δ is a sequential μ-test. By Definition 2.5.1 the function δ is lower semicomputable. Therefore, f defined as in the lemma is lower semicomputable. By Definition 2.5.1 we also have $\mu\{\omega : \delta(\omega) \geq m\} \leq 2^{-m}$, for each m. Define $f(\omega) = c 2^{\delta(\omega)} / \delta(\omega)^2$ for a constant $c = (\pi^2/3)^{-1}$ and $f(x) = \inf_{\omega \in \Gamma_x} \{f(\omega)\}$. Since $2^x/x^2$ is monotonic, these functions are also lower semicomputable. Define

$$E_m = \{\omega : m \leq \delta(\omega) < m + 1\}, \text{ and } c_m = \sup_{\omega \in E_m} \{f(\omega)\}.$$

Then (with $m \geq 1$),

$$\begin{aligned} \int_S f(\omega)\mu(d\omega) &\leq \sum_m \mu(E_m) \\ &\leq c \sum_m \frac{2^{m+1}}{m^2} \, 2^{-m} \leq 2c \sum_m \frac{1}{m^2} \leq 1. \end{aligned}$$

□

Definition 4.5.12 Let f be a unit integrable function over S with respect to μ. A function δ is an *integral μ-test* iff $\delta(\omega) = \log f(\omega)$. It is a *universal integral μ-test* if it additively dominates all integral μ-tests.

Lemma 4.5.7 states that sequential μ-tests and integral μ-tests correspond up to a logarithmic additive term. It remains to show that there exists a universal integral μ-test. We proceed by demonstrating the existence of a universal unit integrable function.

Definition 4.5.13 A function f_0 is *universal* for a class of unit integrable functions over a domain X if f_0 belongs to the class and for each f in the class there is a constant c such that for all $\omega \in X$ we have $cf_0(\omega) \geq f(\omega)$.

Theorem 4.5.5 *The class of lower semicomputable unit integrable functions (over X with respect to a computable measure μ) has a universal element, the function f_0 defined by*

$$\log f_0(\omega) = \sup_{\omega \in \Gamma_x} \left\{ \log \frac{1}{\mu(x)} - K(x|\mu) \right\}.$$

Proof. We need to show that f_0 is lower semicomputable, unit integrable, and that it multiplicatively dominates all lower semicomputable unit integrable functions. Since $K(\cdot)$ is upper semicomputable and $\mu(\cdot)$ is computable, it follows that f_0 is lower semicomputable.

Claim 4.5.2 The function f_0 is unit integrable.

Proof. First, write

$$f_0(\omega) = \sup_{\omega \in \Gamma_x} \left\{ 2^{-K(x|\mu) + \log 1/\mu(x)} \right\}.$$

We have defined f_0 only on elements of $\mathcal{B}^\infty$. If we want to define f_0 on finite sequences $x \in \mathcal{B}^*$, then the natural way is

$$f_0(x) \stackrel{\text{def}}{=} \inf_{\omega \in \Gamma_x} \{ f_0(\omega) \}.$$

This makes $f_0(\omega_{1:n})$ monotonic nondecreasing in n. Let

$$g(x) = 2^{-K(x|\mu) + \log 1/\mu(x)}.$$

Using Theorems 4.3.1 and 4.3.3, pages 267 and 273,

$$\begin{aligned} \int_X f_0(\omega)\mu(d\omega) &= \int_X \sup_n \{ g(\omega_{1:n}) \} \mu(d\omega) \\ &\leq \sum_n \int_X g(\omega_{1:n}) \mu(d\omega) \\ &= \sum_n \sum_{l(x)=n} g(x)\mu(x) \\ &= \sum_x g(x)\mu(x) = \sum_x 2^{-K(x|\mu)} \leq 1. \end{aligned}$$

□

Claim 4.5.3 The function f_0 multiplicatively dominates all lower semicomputable unit integrable functions.

Proof. Let g be a lower semicomputable unit integrable function. Without loss of generality, assume that g has values of the form 2^m only. Lower semicomputability of g is equivalent to saying that there exists a recursively enumerable set T of pairs (x, m) such that

$$g(\omega) = \sup\{2^m : \omega \in \Gamma_x \text{ and } (x, m) \in T\}.$$

For a set T of pairs (x, m), define the subset of elements stabbed by ω as

$$T(\omega) \stackrel{\text{def}}{=} \{(x, m) \in T : \omega \in \Gamma_x\}.$$

The proof goes by replacing T by a set T' such that $T'(\omega)$ contains at most one element (x, m) for each m. For this purpose, proceed as follows: Use T to define the recursively enumerable set T_m by

$$T_m = \{x : (x, m) \in T\}.$$

Each element $x \in T_m$ is associated with a cylinder $\Gamma_x \subseteq S$. Similarly, T_m is associated with the union of all these cylinders: $R = \bigcup_{x \in T_m} \Gamma_x$. However, the constituent cylinders may overlap: some infinite sequences ω have two different prefixes in T_m. By replacing T_m by a set T'_m in which no element is a prefix of another element, and such that ω has a prefix in T_m iff it has a prefix in T'_m, we achieve that R equals the union of the nonoverlapping cylinders of elements in T'_m.

From the enumeration of T_m we obtain an enumeration of such a prefix-free set T'_m by the following processing step. Starting with an empty set T'_m, we put each enumerated element x from T_m into T'_m as long as x is not a prefix of an element already in T'_m and there is no element in T'_m that is a prefix of x. If T'_m contains a prefix of x, then we simply discard x. If x is the prefix of one or more elements in T'_m, then we replace x by the smallest prefix-free set A of elements of the form xy such that each infinite sequence starting with x has a prefix in $T'_m \bigcup A$. Then the latter set is prefix-free and we set $T'_m := T'_m \bigcup A$. Since T'_m is finite at each stage, each such set A is finite and can be effectively determined. We give a formal description of this processing step:

Initialize: $T'_m := \varnothing$.

For each enumerated $x \in T_m$ **do**

if $\Gamma_x \bigcap \Gamma_y = \varnothing$ for all $y \in T'_m$ **then** $T'_m := T'_m \bigcup \{x\}$

else if $\Gamma_x \subseteq \Gamma_y$ for some $y \in T'_m$ **then** discard x

else $T'_m := T'_m \bigcup A$

{where we choose $A := \{xy_1, \ldots, xy_k\}$ the smallest set such that:

1. $\Gamma_z \bigcap \Gamma_{z'} = \varnothing$ for all $z, z' \in T'_m \bigcup A$; and
2. $\bigcup_{z \in T'_m \bigcup A} \Gamma_z = \bigcup_{z \in T'_m \bigcup \{x\}} \Gamma_z$ }.

By construction, T'_m is recursively enumerable and prefix-free. Because it is prefix-free, the subset $T'_m(\omega)$ stabbed by ω is a singleton set or $\varnothing$. We are now ready to define the promised T' as $T' = \bigcup_m \{(x, m) : x \in T'_m\}$. The subset $T'(\omega)$ stabbed by ω contains at most one element (x, m) for each m.

By the construction above,

$$g(\omega) = \sup\{2^m : T'_m(\omega) \neq \varnothing\}.$$

Define further

$$g(x) \stackrel{\text{def}}{=} \sup_{x \in T'_m} \{2^m\}, \text{and } g'(x) \stackrel{\text{def}}{=} \sum_{\{m : x \in T'_m\}} 2^m,$$

where $g(x) = g'(x) = 0$ if there is no m with $x \in T'_m$. In this way,

$$g(\omega) = \sup_{\omega \in \Gamma_x} \{g(x)\}, \text{ and } g(x) \leq g'(x). \tag{4.17}$$

Since each m occurs at most once in an element in $T'(\omega)$, we have

$$2g(\omega) \geq \sum_{T'_m(\omega) \neq \varnothing} 2^m = \sum_{\{x : \omega \in \Gamma_x\}} \sum_{\{m : x \in T'_m\}} 2^m = \sum_{\{x : \omega \in \Gamma_x\}} g'(x).$$

Therefore,

$$\begin{aligned} \sum_x g'(x)\mu(x) &= \sum_x \int_{\Gamma_x} g'(x)\mu(d\omega) \\ &= \int_X \sum_{\{x : \omega \in \Gamma_x\}} g'(x)\mu(d\omega) \\ &\leq 2 \int_X g(\omega)\mu(d\omega) \leq 2. \end{aligned}$$

Since $g'(x)\mu(x)/2$ is lower semicomputable and by the above sums to at most 1, it follows from Theorems 4.3.1 and 4.3.3 on pages 267 and 273, respectively, that there is a constant c such that

$$g'(x) \leq c \frac{2^{-K(x|\mu)}}{\mu(x)}.$$

Since $g(x) \leq g'(x)$, using Equation 4.17,

$$g(\omega) \leq \sup_{\omega \in \Gamma_x} \left\{ c 2^{-K(x|\mu) + \log 1/\mu(x)} \right\}.$$

□ □

Corollary 4.5.2 Let μ be a computable measure. The function

$$\rho_0(\omega|\mu) = \sup_{\omega \in \Gamma_x} \left\{ \log \frac{1}{\mu(x)} - K(x|\mu) \right\}$$

is a universal integral μ-test.

Example 4.5.1 With respect to the special case of the uniform distribution λ, Corollary 4.5.2 sets $\rho_0(\omega|\lambda) = \sup_{n \in \mathcal{N}} \{n - K(\omega_{1:n})\}$ up to a constant additional term. This is the expression we found already in Corollary 3.6.1, page 223. ◇

Example 4.5.2 To quantify the domination constants between f_0 and the lower semicomputable unit integrable functions, we can proceed by an argument we have met several times before (for example, Theorem 4.5.1, page 294). First one shows that the lower semicomputable unit integrable functions can be effectively enumerated as

$$f_1, f_2, \ldots .$$

Second, one shows that

$$f_0'(\omega) = \sum_{n \geq 1} \alpha(n) f_n(\omega), \text{ with } \sum \alpha(n) \leq 1,$$

is unit integrable. We can choose $\alpha(n) = 2^{-K(n)}$. Since the individual f_n's are lower semicomputable, f_0' is also lower semicomputable. Since the individual f_n's are unit integrable, f_0' is also unit integrable. Since $f_0'(\omega) \geq 2^{-K(n)} f_n(\omega)$, for all n and ω, the function f_0' is a universal lower semicomputable unit integrable function. By the above theorem, f_0 is also a universal lower semicomputable unit integrable function. Therefore, there is a constant c such that

$$c f_0(\omega) \geq f_0'(\omega) \geq 2^{-K(n)} f_n(\omega).$$

◇

Example 4.5.3 Consider a discrete sample space S and the class of lower semicomputable unit integrable functions over S with a computable measure μ.

For convenience we set $S \subseteq \mathcal{N}$. Then $\sum_{x\in S} f(x)\mu(x) \leq 1$. For each such function f we have that $f(x)\mu(x)$ is lower semicomputable and sums up to at most 1. By Theorem 4.3.1 on page 267, there is a constant c such that $f(x)\mu(x) \leq c \cdot \mathbf{m}(x|\mu)$ for all x. We have $K(x|\mu) = \log 1/\mathbf{m}(x|\mu) + O(1)$, by Theorem 4.3.3, page 273, and therefore

$$f(x) \leq c2^{-K(x|\mu)+\log 1/\mu(x)}.$$

Since $\mathbf{m}$ is lower semicomputable unit integrable, we find that the universal lower semicomputable unit integrable function over S with respect to μ is

$$f_0(x) = 2^{-K(x|\mu)+\log 1/\mu(x)}.$$

The discrete sample space approach is connected to the continuous sample space treatment as follows: If we induce from f_0 a function f_0' over $\mathcal{B}^\infty$ by defining $f_0'(\omega) = \sup_n\{f_0(\omega_{1:n})\}$, then we obtain the function of Theorem 4.5.5 again. ◇

4.5.7 *Randomness by Martingale Tests

This section parallels the discussion in Sections 4.3.5 and 4.3.6. We are presented with an infinite sequence of outcomes by an adversary who claims that the distribution of outcomes is the computable measure μ on $S = \mathcal{B}^\infty$, where $\mathcal{B}$ is a finite nonempty set of basic symbols. We would like to verify this claim under the assumption (guaranteed by the adversary) that the game is fair in the following sense: Let there be given an agreed-upon function f satisfying

$$\sum_{l(x)=n} f(x)\mu(x) \leq 1,$$

for all n. In every game, you pay the standard amount of \$1 to the adversary and he pays you $\$f(x)$ if the outcome of the game is x.

Below we define a type of test that embodies such a betting strategy. Let $x \in \mathcal{B}^*$ and $b \in \mathcal{B}$. You receive $f(x) = 2^{\gamma(x)}$ units on an outcome x. The first inequality of Definition 4.5.14 represents your stake in each play. The second inequality of Definition 4.5.14 says that if you continue playing another step after history x, then on average (using the conditional probability $\mu(xb)/\mu(x)$), your payoff will not increase.

Definition 4.5.14 Let $\mu : \mathcal{B}^* \to \mathcal{R}$ be a computable measure on the sample space $\mathcal{B}^\infty$. Let $\gamma : \mathcal{B}^* \to \mathcal{R}$ be a nonnegative, lower semicomputable function with the property

$$\mu(\epsilon)2^{\gamma(\epsilon)} \leq 1,$$
$$\mu(x)2^{\gamma(x)} \geq \sum_{b\in\mathcal{B}} \mu(xb)2^{\gamma(xb)}.$$

Then γ is a *martingale μ-test*. A martingale μ-test is *universal* for a class of martingale μ-tests if it additively dominates all martingale μ-tests.

This is a 'test' version of the martingale we will meet in Section 4.5.8 on page 319. We have formulated it here as a 'test' for ease of comparison with the original Martin-Löf test of Definition 2.4.1, page 135.

Lemma 4.5.8 *Each martingale μ-test is a μ-test. If $\delta(x)$ is a μ-test, then there exists a constant c such that $\delta(x) - 2\log\delta(x) - c$ is a martingale μ-test.*

Proof. It follows immediately from the new definition that for all n,

$$\sum_{l(x)=n} \{\mu(x) : \gamma(x) > k\} \leq 2^{-k}. \tag{4.18}$$

Conversely, if $\gamma(x)$ satisfies Equation 4.18, then for some constant c the function $\gamma(x) - 2\log\gamma(x) - c$ satisfies Definition 4.5.14. □

Thus, the universal (Martin-Löf) μ-test of Definition 2.4.2 on page 136, the universal sum μ-test of Definition 4.3.8 on page 278, and the universal martingale μ-test all yield the same values up to a logarithmic additive term.

Definition 4.5.15 A *sequential martingale μ-test* δ for $\omega \in \mathcal{B}^\infty$ is obtained from a martingale μ-test γ by defining

$$\delta(\omega) = \sup_{n\in\mathcal{N}} \{\gamma(\omega_{1:n})\}.$$

A sequential martingale μ-test $\sigma_0(\cdot|\mu)$ is *universal* if it additively dominates all sequential martingale μ-tests δ: there is a constant c such that for all $\omega \in \mathcal{B}^\infty$ we have $c \cdot \sigma_0(\omega|\mu) \geq \delta(\omega)$.

Theorem 4.5.6 *Let $\mu : \mathcal{B}^* \to \mathcal{R}$ be a computable measure on $\mathcal{B}^\infty$.*

(i) *The function $\gamma_0 : \mathcal{B}^* \to \mathcal{R}$ defined by $\gamma_0(x|\mu) = \log(\mathbf{M}(x)/\mu(x))$ is a universal martingale μ-test.*

(ii) *The function $\sigma_0 : \mathcal{B}^\infty \to \mathcal{R}$ defined by*

$$\sigma_0(\omega|\mu) = \sup_n \left\{ \log \frac{\mathbf{M}(\omega_{1:n})}{\mu(\omega_{1:n})} \right\}$$

is a universal sequential martingale μ-test.

Proof. Let V_k be the set of infinite sequences ω such that there is an index n with $k < \log 1/\mu(\omega_{1:n}) - Km(\omega_{1:n})$. Using $\log 1/\mathbf{M}(\omega_{1:n}) \leq Km(\omega_{1:n})$, this implies that

$$\gamma_0(\omega_{1:n}|\mu) = \log \frac{\mathbf{M}(\omega_{1:n})}{\mu(\omega_{1:n})} > k,$$

for all ω in V_k. The left-hand side of the inequality satisfies Definition 4.5.14. By Equation 4.18, therefore, $\mu(V_k) \leq 2^{-k}$. Since $\mathbf{M}$ is lower semicomputable and μ is computable, γ_0 is lower semicomputable. Hence it is a martingale μ-test, and σ_0 is a sequential martingale μ-test. By Definition 4.5.14 the function $\mu(x)2^{\gamma(x)}$ is a lower semicomputable semimeasure. Thus, for each martingale μ-test γ, there is a positive constant c_γ such that

$$\mathbf{M}(x) \geq c_\gamma \mu(x) 2^{\gamma(x)}.$$

Therefore, γ_0 and σ_0 additively dominate all martingale μ-tests and sequential martingale μ-tests, respectively. □

As before, we call $\gamma_0(x|\mu)$ the *randomness deficiency* of x. Theorem 4.5.6 yields yet another characterization of an infinite random sequence, equivalent to Martin-Löf's characterization of randomness in Section 2.5. Namely, for each such μ, an infinite binary sequence ω is μ-*random* iff

$$\sigma_0(\omega|\mu) < \infty.$$

By the proof of Theorem 4.5.6 this is equivalent to

$$\sup_n \left\{ \log \frac{1}{\mu(\omega_{1:n})} - Km(\omega_{1:n}) \right\} < \infty.$$

Corollary 4.5.3 An infinite sequence ω is random iff $\sup_n\{\log 1/\mu(\omega_{1:n}) - Km(\omega_{1:n})\} < \infty$. (The expression on the left side is yet another μ-randomness test.)

Altogether, we have found the following characterizations in this section. An infinite sequence ω is μ-random iff

$$KM(\omega_{1:n}) = Km(\omega_{1:n}) + O(1) = \log \frac{1}{\mu(\omega_{1:n})} + O(1).$$

(The $O(1)$ term is independent of n but may depend on ω.) The μ-random infinite sequences by definition have μ-measure one in the set of all infinite sequences. Such sequences, and only such sequences, satisfy all effective laws of probability theory (that is, withstand any sequential μ-test in Martin-Löf's sense). There is also a conditional version of Theorem 4.5.6:

Corollary 4.5.4 Let $f(\omega, \zeta)$ be a function with $\int_S 2^{-f(\omega,\zeta)}\lambda(d\omega) \leq 1$, where λ is the uniform measure. Such functions f define conditional measures as follows: Define $f(x, \zeta) = \sup_{\omega\in\Gamma_x}\{f(\omega, \zeta)\ \}$. Then,

$$\mu(x|\zeta) = \int_{\Gamma_x} 2^{-f(\omega,\zeta)}\lambda(d\omega).$$

If f is upper semicomputable (the corresponding $\mu(\omega|\zeta)$ is lower semicomputable), then there is a constant c such that $c \cdot \mathbf{M}(\omega|\zeta) \geq 2^{-f(x,\zeta)}$, for all ζ and all x with $\omega \in \Gamma_x$.

Example 4.5.4 Let $\mu = \lambda$, the uniform measure. The above discussion says that an infinite binary sequence ω is random with respect to the uniform measure (that is, in the original sense of Martin-Löf) iff

$$KM(\omega_{1:n}) = Km(\omega_{1:n}) + O(1) = n + O(1).$$

◇

4.5.8 *Randomness by Martingales

In gambling, the 'martingale' is a well-known system of play. With this system one bets in a casino that the outcome of a roulette run will be red each time, and after each loss the new bet is double the old one. No matter how long the run of losses, once red comes up, the gambler wins. The current gain minus past loss in this run equals the amount of the first bet. But even though red is bound to come up sometime, the gambler may go broke and lose all before this happens. In his 1853 book *The Newcomes*, W.M. Thackeray admonishes, "You have not played as yet? Do not do so; above all avoid the martingale if you do."

According to von Mises's Definition 1.9.1 on page 51, a random sequence in the sense of a collective defies a player betting at fixed odds, and in fixed amounts, on the tosses of a fair coin, to make unbounded gain in the long run. Von Mises does not require that the player can have no debt. Obviously, if the player cannot go broke, and he bets according to a martingale, his gain will *eventually* exceed each bound.

We can test for randomness by betting. Suppose a casino claims that the distribution of outcomes ω in sample space $S = \mathcal{B}^\infty$ is the measure μ. Then given any function $f(\omega)$, with $\int_S f(\omega)\mu(d\omega) < 1$, the casino should accept 1 unit for an obligation to pay $f(\omega)$ on outcome ω. (We have called such functions 'unit integrable' in Definition 4.5.10 on page 310.) A martingale is a payoff function that leads to such a global payoff.

Definition 4.5.16 Let $\mu : \mathcal{B}^* \to [0, 1)$ be a measure on $S = \mathcal{B}^\infty$. Let t be a nonnegative function from $\mathcal{B}^*$ into $\mathcal{R}$ such that with $x \in \mathcal{B}^*$,

$$\mu(\epsilon)t(\epsilon) \leq 1,$$
$$\mu(x)t(x) \geq \sum_{b\in\mathcal{B}} \mu(xb)t(xb).$$

Let ω be an element of S. We call $t(\omega_{1:n})$ a *μ-supermartingale*. We call $t(\omega_{1:n})$ a *μ-martingale* if the equations hold with equality.

Definition 4.5.17 A lower semicomputable μ-supermartingale t_0 is *universal* for the class of lower semicomputable μ-supermartingales if it multiplicatively dominates all lower semicomputable μ-supermartingales t'. (That is, there is a constant c such that $c \cdot t_0(x) \geq t'(x)$ for all $x \in \mathcal{B}^*$.)

Theorem 4.5.7 *There is a universal lower semicomputable μ-supermartingale. We denote it by $t_0(\cdot|\mu)$.*

Proof. Define $t_0(x|\mu) = \mathbf{M}(x)/\mu(x)$. Then $t_0(x|\mu) = 2^{\gamma_0(x|\mu)}$ with $\gamma_0(\cdot|\mu)$ as in Theorem 4.5.6. From Definition 4.5.15 and Definition 4.5.16, it follows that $\gamma(x)$ is a martingale μ-test iff $t(x) = 2^{\gamma(x)}$ is a lower semicomputable μ-supermartingale. Hence, the theorem follows from Theorem 4.5.6. □

Example 4.5.5 We give an alternative proof of Theorem 4.5.7. Comparison with Definition 4.2.1 shows that $t(x)$ is a lower semicomputable μ-supermartingale iff $t(x)\mu(x)$ is a lower semicomputable semimeasure. By Theorem 4.5.1 on page 294, the semimeasure $\mathbf{M}$ is a universal lower semicomputable semimeasure. Hence, for each lower semicomputable μ-supermartingale t there is a constant c such that $c \cdot \mathbf{M}(x) \geq t(x)\mu(x)$, for all x. Therefore,

$$t_0(x|\mu) = \frac{\mathbf{M}(x)}{\mu(x)}$$

dominates all lower semicomputable μ-supermartingales within a multiplicative constant. Thus $t_0(x|\mu)$ is a universal lower semicomputable μ-supermartingale. ◇

Intuition tells us to call an outcome ω nonrandom if it allows us to win against the adversary by choosing an appropriate payoff function, that is, if there is a lower semicomputable supermartingale t such that $t(\omega_{1:n})$ grows unboundedly. From the definitions it follows immediately that the characterization of random infinite sequences, as the complement of the nonrandom ones, is exactly the same as the one using sequential μ-tests. Since $\gamma_0(x|\mu)$ dominates all μ-tests, within additive constants, it follows that $t_0(x|\mu) = \mathbf{M}(x)/\mu(x)$ dominates all lower semicomputable μ-supermartingales within a multiplicative constant. Since γ_0 is a universal martingale μ-test, we have the following corollary to Theorem 4.5.7:

Corollary 4.5.5 An infinite sequence ω is μ-random iff

$$t_0(\omega|\mu) = \sup_{n \in \mathcal{N}} \{t_0(\omega_{1:n}|\mu)\} < \infty.$$

We recall that the set of such ω has μ-measure one.

4.5.9 *Relations Between Tests

There is a simple relation between supermartingales and unit integrable functions. The supermartingale condition of Definition 4.5.16 implies that for a μ-supermartingale t, for all prefix-free sets $I \subseteq \mathcal{B}^*$ we have

$$\sum_{x \in I} \mu(x) t(x) \leq 1.$$

This implies that t is a unit integrable function in the sense of Definition 4.5.10.

Unit integrability of a function f does not imply that it is a supermartingale. A supermartingale t is not necessarily monotonic nondecreasing but instead satisfies the restrictive supermartingale condition. We can interpret the difference as saying that the unit integrable functions represent betting strategies in a fair game where the expectation of profit in the overall infinite game is at most 1. The supermartingale condition says that the expectation of profit for a fair game of some finite length cannot be increased by playing longer. The condition on prefix-free sets states that the expectation with an arbitrary set of stopping times is still at most 1.

Assume that a function f is unit integrable. The integral can be expressed as a sum as follows: Taking the supremum over all prefix-free sets $I \subseteq Y$ we have

$$\int_X f_0(\omega) \mu(d\omega) = \sup_I \left\{ \sum_{x \in I} \mu(x) f_0(x) \right\}.$$

Then we can increase f to a μ-supermartingale t by defining

$$t(x) = \sup_I \left\{ \frac{1}{\mu(x)} \sum_{y \in I} f(xy) \mu(xy) \right\},$$

where I ranges over all prefix-free subsets of $\mathcal{B}^*$. The proof that t is a μ-supermartingale is by simply writing out both sides of the supermartingale condition of Definition 4.5.16 in terms of suprema and verifying the required relations. The fact that we increase f to obtain t accounts for the fact that the universal lower semicomputable unit integrable function f_0 is slightly smaller than the universal μ-supermartingale $t_0(\cdot|\mu)$. For fixed ω,

$$\sup_{\omega \in \Gamma_x} \left\{ 2^{-K(x|\mu) + \log 1/\mu(x)} \right\} \leq \sup_{\omega \in \Gamma_x} \left\{ \frac{\mathbf{M}(x)}{\mu(x)} \right\}.$$

To show that a supermartingale can be used to define a function over $\mathcal{B}^\infty$ we use the so-called supermartingale convergence theorem:

Claim 4.5.4 Consider a sequence of random variables $\omega_1, \omega_2, \ldots$. If $t(\omega_{1:n})$ is a μ-supermartingale and the μ-expectation $\mathbf{E}|t(\omega_{1:n})|$ is finite, then it follows that $\lim_{n\to\infty} t(\omega_{1:n})$ exists with μ-probability one.

Proof. See J.L. Doob, *Stochastic Processes* , Wiley, 1953, pp. 324–325. □

This implies that each μ-supermartingale $t(\omega_{1:n})$ converges μ-almost everywhere to a function $t(\omega)$. But this function may not be lower semicomputable. We obtain lower semicomputability if the function $t(\omega_{1:n})$ is monotonic in n.

The relation between the universal lower semicomputable supermartingale and the universal lower semicomputable unit integrable function is as follows. The function

$$\rho_0(\omega|\mu) = \sup_n \left\{ -K(\omega_{1:n}|\mu) + \log \frac{1}{\mu(\omega_{1:n})} \right\}$$

is a universal integral μ-test, and

$$\sigma_0(\omega|\mu) = \sup_n \left\{ \frac{\log \mathbf{M}(\omega_{1:n})}{\mu(\omega_{1:n})} \right\}$$

is a universal martingale μ-test. Hence they are either both finite (for μ-random ω) or both infinite (otherwise).

Lemma 4.5.9 *For each infinite sequence ω, we have up to fixed additive constants*

$$\sigma_0(\omega|\mu) - 2\log\sigma_0(\omega|\mu) \leq \rho_0(\omega|\mu) \leq \sigma_0(\omega|\mu).$$

Proof. The right inequality follows from the fact that we have to increase the universal unit integrable function to make it a universal supermartingale.

The left inequality is proved as follows: Let $\gamma^m(\omega) = m$ if ω has a prefix x with $\log(\mathbf{M}(x)/\mu(x)) \geq m$ and 0 otherwise. The function $\sigma(\omega) = \sup_m\{\gamma^m(\omega) - 2\log m\}$ satisfies $\sigma(\omega) \leq \rho_0(\omega|\mu) + O(1)$, since

$$2^{\sigma(\omega)} = \max_m \left\{ \frac{2^{\gamma^m(\omega)}}{m^2} \right\} = \sum_m \frac{2^{\gamma^m(\omega)}}{m^2}$$

is $(\pi^2/6)$ integrable (instead of unit integrable) over X with respect to μ, as is easy to verify. Clearly, $\sigma_0(\omega|\mu) - 2\log\sigma_0(\omega|\mu) \leq \sigma(\omega)$, from which the inequality follows. □

We can similarly analyze the relation between the universal sequential μ-test $\delta_0(\cdot|\mu)$ and the other tests.

Lemma 4.5.10 *For each infinite sequence ω, we have up to fixed additive constants*

$$\delta_0(\omega|\mu) - 2\log\delta_0(\omega|\mu) \leq \sigma_0(\omega|\mu) \leq \delta_0(\omega|\mu).$$

Proof. This follows from the definitions in Theorem 4.5.6 on page 317, and Lemma 4.5.8 on page 317. □

Exercises

4.5.1. [09] Show that with basis $\mathcal{B} = \{0,1\}$, we have $\mathbf{M}(\epsilon) < 1$ and $\mathbf{M}(x) > \mathbf{M}(x0) + \mathbf{M}(x1)$, for all x in $\mathcal{B}^*$ (strict inequality). Generalize this to arbitrary $\mathcal{B}$.

4.5.2. [31] Let the probability that an initial segment x of a binary sequence is followed by $a \in \{0,1\}^*$ be $\mathbf{M}(a|x) = \mathbf{M}(xa)/\mathbf{M}(x)$.

(a) Show that there is a constant $c > 0$ such that the probability (with respect to $\mathbf{M}$) of the next bit being 0 after 0^n is at least c.

(b) Show that there exists a constant c such that the probability (with respect to $\mathbf{M}$) of the next bit being 1 after 0^n is at least $1/(cn\log^2 n)$.

(c) Show that for every constant c and sufficiently large N, there are at most N/c initial segments 0^n $(1 \leq n \leq N)$ such that the probability (with respect to $\mathbf{M}$) of the next bit being 1 is larger than $(c\log^2 n)/n$.

(d) Conclude that the probability (with respect to $\mathbf{M}$) of the next bit following 0^n being 1 is $f(n)/n$ with $f(n) = \Omega(1/\log^2 n) \bigcap O(\log^2 n)$.

Comments. This exercise is a simple form of Occam's razor: the conditional $\mathbf{M}$ probability assigns high probability to the simple explanations and low probability to the complex explanations. The assertion Item (d) is an $\mathbf{M}$ prior probability variant of P.S. Laplace's well-known exercise to compute the expectation of a successful trial following n successful trials in a Bernoulli process $(p, 1-p)$ with unknown p. Using Bayes's rule with uniform prior probability, this expectation is $(n+1)/(n+2)$ and is a special case of Laplace's law of succession (Exercise 1.10.6 on page 65). Application of this law sets the probability of a 1 following n initial 0's at $1/(n+2)$. This is fairly close to the approximation we found in Item (d) using conditional $\mathbf{M}$ probability. In *A Philosophical Essay on Probabilities*, Laplace uses this rule to compute the probability that the sun will not rise tomorrow, given that it has been rising every morning since the creation of the world 10,000 years ago. Using the above, it follows that the probability that the sun will not rise tomorrow is approximately 1/3,650,000. This is correct, in case our information about the sun were exhausted by the fact stated. Hint for Items (a) through (c): use $|K(x) - \log 1/\mathbf{M}(x)| \leq 2\log K(x) + O(1)$. For all n we have $K(0^n 1) \leq \log n + 2\log\log n + O(1)$, and for the majority of n we

have $K(0^n 1) \geq \log n$. Source: L.A. Levin and A.K. Zvonkin, *Russ. Math. Surveys*, 25:4(1970), 83–124; see also T.M. Cover and J.A. Thomas, *Elements of Information Theory*, Wiley & Sons, New York, 1991.

4.5.3. [27] Even the most common measures can be not lower semicomputable if the parameters are not lower semicomputable. Consider a $(p, 1-p)$ Bernoulli process and the measure it induces on the sample space $\{0,1\}^\infty$.

(a) Show that if p is a computable real number such as $\frac{1}{2}$ or $\frac{1}{3}$ or $\frac{1}{2}\sqrt{2}$ or $\pi/4 = \frac{1}{4} \cdot 3.1415\ldots$, then the measure is computable.

(b) Show that if p is lower semicomputable then the measure can fail to be lower semicomputable

(c) Show that if p is a real that is not lower semicomputable, then the measure is not lower semicomputable either.

(d) Show that if p is a random real whose successive digits in its binary expansion are obtained by tosses of a fair coin, then with probability one the measure is not lower semicomputable.

Comments Hint for Item (b): if p is lower semicomputable but not computable, then $1-p$ is not lower semicomputable and it is the probability that the first element in the sequence is 0 [P. Gács, personal communication].

4.5.4. • [25] The *Solomonoff normalization* of a semimeasure μ, with $\mathcal{B} = \{0,1\}$, is $\mu_{norm}(x) = a(x)\mu(x)$, with $a(x)$ as defined in Definition 4.5.7 in Section 4.5.3. We call $\mathbf{M}_{\rm norm}$, the *normalized* version of the universal lower semicomputable semimeasure $\mathbf{M}$, the *Solomonoff measure*. This leads to an important parallel development of the theory, which may be mathematically less elegant, but is possibly preferable in some applications.

(a) Show that for each semimeasure μ, the function $\mu_{\rm norm}$ is a measure. Conclude that in particular, $\mathbf{M}_{\rm norm}$ is a measure.

(b) Show that the normalization factor $a(x)$ is at least 1 for all x in $\mathcal{B}^*$.

(c) Show that $\mathbf{M}_{\rm norm}$ dominates all lower semicomputable semimeasures μ. (That is, for each such μ, there is a positive constant c such that for all $x \in \mathcal{B}^*$, we have $\mathbf{M}_{\rm norm}(x) \geq c\mu(x)$.) Conclude that $\mathbf{M}_{\rm norm}$ dominates $\mathbf{M}$.

(d) Show that Solomonoff normalization is not the only normalization.

(e) Let μ be a lower semicomputable measure. Does $\mathbf{M}_{\rm norm}$ dominate all $\mu_{\rm norm}$'s?

Comments. Hint for Item (e): not all $\mu_{\rm norm}$'s are lower semicomputable, so we cannot use Item (c). This elaborates the discussion in Section 4.5.3.

Source: Items (a) through (d) R.J. Solomonoff [*IEEE Trans. Inform. Theory*, IT-24(1978), 422–432]; see also 'History and References,' Section 4.7.

4.5.5. [32] By Exercise 4.5.4, $\mathbf{M}_{\rm norm}$ dominates $\mathbf{M}$.

(a) Show that $\mathbf{M}$ does not multiplicatively dominate $\mathbf{M}_{\rm norm}$.

(b) Show that for *each* normalizer a defining the measure $\mathbf{M}'(x) = a(x)\mathbf{M}(x)$ we have $\mathbf{M}(\omega_{1:n}) = o(\mathbf{M}'(\omega_{1:n}))$, for some infinite ω.

(c) (Open) Item (b) with 'all' substituted for 'some.'

Comments. Item (b) implies that $\mathbf{M}$ does not dominate any of its normalized versions $\mathbf{M}'$. This is a special case of Exercise 4.5.6.

4.5.6. • [37] What is the difference between semimeasure $\mathbf{M}$ and *any* (not necessarily lower semicomputable) measure μ?

(a) Show that $1/\mathbf{M}(x)$ differs from $1/\mu(x)$, for infinitely many x, as the busy beaver function $BB(n)$ differs from n (Exercise 1.7.19 on page 45) for every measure μ.

(b) Show the same about $\mathbf{M}(x)$ versus $\mathbf{M}(x0) + \mathbf{M}(x1)$.

Comments. This shows that the normalization $\mathbf{M}_{\rm norm}$ must distort the measure $\mathbf{M}$. Exercise 4.5.7 shows thay this is rare. Hint: take the maximal running time $T(k)$ of those among the first k programs that halt. Note that $T(k)$ is the longest running time of a halting program among the first k programs and is related to $BB(k)$. The following procedure effectively generates the desired x from k (even though T is only lower semicomputable) such that $\mathbf{M}(x0) + \mathbf{M}(x1) < 1/T(k)$. Start with the empty prefix $y = \epsilon$ of x. Find its extension $y' = y0$ or $y' = y1$ (whichever you find first) such that $\mathbf{M}(y') > 1/(2T(k))$. Extend y to differ from y'. Repeat as long as you can (up to $2T(k)$ times). The resulting string x (of length $< 2T(k)$) will have both extensions of $\mathbf{M}$-semimeasure $< 1/(2T(k))$. Since x is described by k and an $O(1)$ program, for large enough random k we have $1/k^2 \leq \mathbf{M}(x)$. See also 'History and References,' Section 4.7. Source: one of us (PV) asked L.A. Levin; Levin asked R.M. Solovay, and returned Solovay's solution on September 12, 1989, by e-mail.

4.5.7. • [27] Let μ be a computable measure over $\mathcal{B}^\infty$. We use $\mathbf{M}$ to estimate the probabilities $\mu(a|y)$ for $a \in \mathcal{B}$ and $y \in \mathcal{B}^*$. Show that for every n, $\sum_{l(x)=n} \mu(x) \sum_{i=1}^{n} \mathbf{M}(u|x_{1:i-1}) \leq K(\mu) \ln 2$, where we define $x_{1:0} = \epsilon$ and $\mathbf{M}(u|x_{1:i-1}) = 1 - \sum_{a\in\mathcal{B}} \mathbf{M}(a|x_{1:i-1})$.

Comments. Let μ be an unknown computable measure, and suppose we are given a string $x = x_1 \ldots x_n$. We can use $\mathbf{M}(a|x_{1:i-1})$ to predict $\mu(a|x_{1:i-1})$ ($a \in \mathcal{B}$ and $1 \leq i \leq n$). But $\mathbf{M}$ being a semimeasure, we can have $\sum_{a\in\mathcal{B}} \mathbf{M}(a|x) < 1$. Exercise 4.5.6, with $\mathcal{B} = \{0, 1\}$, shows

that $\mathbf{M}(x)/(\mathbf{M}(x0)+\mathbf{M}(x1))$ can be very large, so the universal probability that after printing x the reference monotonic machine will never print again, $\mathbf{M}(xu) = \mathbf{M}(x) - \mathbf{M}(x0) - \mathbf{M}(x1)$ with u the next symbol being 'undefined,' is close to 1. The current exercise shows that this occurs only rarely, and for long sequences produced according to computable measure μ these occurrences do not have much μ-probability. Indeed, $\sum_{l(x)=n} \mu(x) \sum_{i=1}^{n} \mathbf{M}(u|x_{1:i-1}) < \sum_{i=1}^{n} 1/i$ for growing n. Hint: in Section 4.5.3 we defined the normalized version $\mathbf{M}_{\rm norm}(x)$ with $x = x_1 \ldots x_n$ by Equation 4.14 on page 303. Consider the negative Kullback–Leibler divergence $-D(\mu \parallel \mathbf{M}_{\rm norm}) \leq 0$ for strings of length n. Substituting $\mathbf{M}_{\rm norm}$ of Equation 4.14, we obtain $-D(\mu \parallel \mathbf{M}) + \sum_{l(x)=n} \mu(x) \sum_{i=1}^{n} \ln(1/(1-\mathbf{M}(u|x_{1:i-1}))) \leq 0$. Using Corollary 4.5.1 on page 302, we can show that $D(\mu \parallel \mathbf{M}) \leq K(\mu) \ln 2$. Adding the last two inequalities, we obtain $\sum_{l(x)=n} \mu(x) \sum_{i=1}^{n} \ln(1/(1-\mathbf{M}(u|x_{1:i-1}))) \leq K(\mu) \ln 2$. Since $-\ln(1-z) = z+(z^2/2)+(z^3/3)+\cdots$, we obtain $\sum_{l(x)=n} \mu(x) \sum_{i=1}^{n} \sum_{j=1}^{\infty} (\mathbf{M}(u|x_{1:i-1}))^j \leq K(\mu) \ln 2$. Since all terms are positive, we obtain $\sum_{l(x)=n} \mu(x) \sum_{i=1}^{n} \mathbf{M}(u|x_{1:i-1}) \leq K(\mu) \ln 2$. Source: R.J. Solomonoff, *Inform. Process. Lett*, 106:6(2008), 238–240.

4.5.8. [17] Let $\nu_1, \nu_2, \ldots$ be an effective enumeration of all discrete lower semicomputable semimeasures. For each i define $\gamma_i(x) = \sum_y \{\nu_i(xy) : y \in \{0,1\}^*\}$.

(a) Show that $\gamma_1, \gamma_2, \ldots$ is an effective enumeration of a subset of the set of lower semicomputable semimeasures. We call these the *extension* semimeasures.

(b) Define $\mathbf{M}c(x) = \sum_y \{\mathbf{m}(xy) : y \in \{0,1\}^*\}$. We call $\mathbf{M}c$ the *extension* semimeasure. Show that $\mathbf{M}c$ is universal in the γ-enumeration, that is, for all k, there is a positive constant c_k such that for all x we have $\mathbf{M}c(x) \geq c_k \gamma_k(x)$.

(c) Show that $\lim_{n\to\infty} \sum_{l(x)=n} \mathbf{M}c(x) = 0$.

(d) Show that the γ-enumeration doesn't contain all lower semicomputable semimeasures.

(e) Show that there is no positive constant c such that for all x we have $\mathbf{M}c(x) \geq c\mathbf{M}(x)$. Conclude that $\mathbf{M}$ dominates $\mathbf{M}c$, but $\mathbf{M}c$ does not dominate $\mathbf{M}$.

(f) Investigate $\mathbf{M}c$ and the normed measure $\mathbf{M}c_{\rm norm}$. (With $(>)\geq$ denoting (strict) domination we have obtained $\mathbf{M}_{\rm norm} \geq \mathbf{M} > \mathbf{M}c$.)

Comments. Hint: for Item (d) in particular, $\mathbf{M}$ is not in there. Consider the limit in Item (c) with $\mathbf{M}$ substituted for $\mathbf{M}c$. The relation between this class of (continuous) measures and the discrete measures has the following extra property: while $\gamma(\epsilon) \leq 1$, we have $\gamma(x) = \gamma(x0) + \gamma(x1) +$

$\nu(x)$ instead of just $\gamma(x) \geq \gamma(x0) + \gamma(x1)$. Moreover, in the γ semimeasures all probability is concentrated on the finite sequences and none on the infinite sequences. Source of definition of $\mathbf{M}c$: T.M. Cover, Universal gambling schemes and the complexity measures of Kolmogorov and Chaitin, Tech. Rept. 12, Statistics Dept., Stanford University, Stanford, CA, 1974. Source of Item (c): R.J. Solomonoff, *IEEE Trans. Inform. Theory*, IT-24(1978), 422–432.

4.5.9. [15] We compare $\mathbf{M}(x)$ and $\mathbf{M}c(x)$. Show that for some infinite ω (such as a recursive real) we have $\lim_{n\to\infty} \mathbf{M}(\omega_{1:n})/\mathbf{M}c(\omega_{1:n}) = \infty$.

Comments. Hint: since $0.\omega$ is a recursive real, $\lim_{n\to\infty} \mathbf{M}(\omega_{1:n}) > 0$.

4.5.10. [31] (a) Let ν be a probability measure and $G(n) = \mathbf{E}f(\omega_{1:n})$ with $f(\omega_{1:n}) = \log \nu(\omega_{1:n}) + \log 1/\mathbf{M}c(\omega_{1:n}))$. ($\mathbf{E}f(\omega_{1:n})$ denotes the ν-expectation $\sum_{l(x)=n} \nu(x)f(x)$.) Show that $\lim_{n\to\infty} G(n) = \infty$.

(b) Let ν be a computable probability measure, and let F be a recursive function such that $\lim_{n\to\infty} F(n) = \infty$. Show that there is a constant c such that for all binary x of length n we have $\log \nu(x) + \log 1/\mathbf{M}c(x) < c + F(n)$.

Comments. Hint: use Exercises 4.5.8, 4.5.9 to solve Item (a). We can interpret $\log 1/\mathbf{M}c(x)$ as the length of the Shannon–Fano code using distribution $\mathbf{M}c$, and $\log 1/\nu(x)$ as the length of the Shannon–Fano code using the actually reigning ν. Clearly, although $\log(\nu/\mathbf{M}c)$ approaches infinity with n, it does so more slowly than any recursive function. In contrast, $\log(\nu/\mathbf{M})$, or $\log(\nu/\mathbf{M}_{\rm norm})$, is bounded by a constant. Source: R.J. Solomonoff, *IEEE Trans. Inform. Theory*, IT-24(1978), 422–432.

4.5.11. [22] Let $\mu(x)$ be a lower semicomputable probability measure. Suppose we define the cooccurrence of events and conditional events anew as follows: The probability of cooccurrence $\mu(x, y)$ is $\mu(x, y) = \mu(x)$ if y is a prefix of x; it is $\mu(x, y) = \mu(y)$ if x is a prefix of y, and equals zero otherwise. The conditional probability is defined as $\mu(y|x) = \mu(x, y)/\mu(x)$. Complexities are defined as follows: the unconditional $K\mu(x) = \log 1/\mu(x)$; cooccurrence $K\mu(x, y) = \log 1/\mu(x, y)$; and conditional $K\mu(x|y) = \log 1/\mu(x|y)$.

Show that these complexities satisfy exactly the information-theoretic equality of symmetry of information: $K\mu(x, y) = K\mu(x) + K\mu(y|x)$ (Sections 1.11, 2.8, 3.9.1).

Comments. Similar (probability) definitions were used by D.G. Willis [*J. ACM*, 17(1970), 241–259]. Source: R.J. Solomonoff, *IEEE Trans. Inform. Theory*, IT-24(1978), 422–432. Solomonoff used $\mathbf{M}_{\rm norm}$ instead of an arbitrary measure μ.

4.5.12. [48] The analogue of Theorem 4.3.3 does not hold for continuous semimeasures. Therefore, the inequality in Theorem 4.5.4 cannot be improved to equality.

(a) Show that for every upper semicomputable function $g : \mathcal{N} \to \mathcal{N}$ for which $Km(x) - KM(x) \leq g(l(x))$, we have $Km(n) \leq g(n) + O(1)$.

(b) Show that for almost all infinite ω, $Km(\omega_{1:n}) - KM(\omega_{1:n})$ has an upper bound that is smaller than any unbounded recursive function.

Comments. See discussion in Section 4.5.4. Source: P. Gács, *Theoret. Comput. Sci.*, 22(1983), 71–93.

4.5.13. [15] Show that an infinite sequence ω is random with respect to a computable measure μ iff the probability ratio $\mu(\omega_{1:n})/\mathbf{M}(\omega_{1:n})$ is bounded below.

Comments. Hint: see Theorem 4.5.6. This ratio can be viewed as the likelihood ratio of hypothesis $\mu(x)$ and the fixed alternative hypothesis $\mathbf{M}(x)$. Source: L.A. Levin, *Soviet Math. Dokl.*, 14(1973), 1413–1416.

4.5.14. [42] Consider a finite or countably infinite basis $\mathcal{B}$, and define a *probability function* $p : \mathcal{B} \to [0, 1]$ such that $\sum_{b\in\mathcal{B}} p(b) \leq 1$. If equality holds, we call the probability function *proper*. The squared *Hellinger distance* $\rho(q, p)$ between two probability functions q and p is defined as $\sum_{b\in\mathcal{B}} \left(\sqrt{q(b)} - \sqrt{p(b)}\right)^2$. The χ^2 *distance*, denoted by $\rho_2(q, p)$, is defined as $\sum_{b\in\mathcal{B}} (q(b) - p(b))^2 / q(b)$. (Use $0/0 = 0$ and $\infty/\infty = 0$.) If μ is a semimeasure over $\mathcal{B}^*$, and $\omega \in \mathcal{B}^\infty$, then the probability function $\mu(\cdot|\omega_{1:n}) : \mathcal{B} \to [0, 1]$ is defined by $\mu(b|\omega_{1:n}) = \mu(\omega_{1:n}b)/\mu(\omega_{1:n})$. If μ is a measure and $\mu(\omega_{1:n}) \neq 0$, then $\mu(\cdot|\omega_{1:n})$ is proper. The randomness deficiency of $\omega_{1:n}$ with respect to μ is $\gamma_0(\omega_{1:n}|\mu) = \log(\mathbf{M}(\omega_{1:n})/\mu(\omega_{1:n}))$. An infinite sequence ω is μ-random iff $\gamma_0(\omega_{1:n}|\mu) = O(1)$.

(a) Suppose μ is a computable measure, σ is a computable semimeasure over $\mathcal{B}^*$, and $\omega \in \mathcal{B}^\infty$. Show that

$$\sum_{i=1}^{n-1} \rho(\sigma(\cdot|\omega_{1:i}), \mu(\cdot|\omega_{1:i})) - \gamma_0(\omega_{1:n}|\mu) - O(1) \leq \gamma_0(\omega_{1:n}|\sigma)$$
$$\leq \sum_{i=1}^{n-1} \rho_2(\sigma(\cdot|\omega_{1:i}), \mu(\cdot|\omega_{1:i})) + 2\gamma_0(\omega_{1:n}|\mu) + O(1).$$

(b) Suppose that μ and σ are computable semimeasures over $\mathcal{B}^*$, and $\omega \in \mathcal{B}^\infty$ is both μ-random and σ-random. Show that

$$\sum_{i=1}^{\infty} \left(\frac{\mu(\omega_{i+1}|\omega_{1:i})}{\sigma(\omega_{i+1}|\omega_{1:i})} - 1\right)^2 < \infty.$$

(c) Suppose μ is a computable measure, σ is a computable semimeasure over $\mathcal{B}^*$, $\omega \in \mathcal{B}^\infty$ is σ-random, and $\mu(\omega_{1:n}) > 0$ for all n. Show that ω is μ-random iff $\sum_{i=1}^{\infty} \rho(\sigma(\cdot|\omega_{1:i}), \mu(\cdot|\omega_{1:i})) < \infty$.

(d) Show that if ω is both μ-random and σ-random, then $\mu(0|\omega_{1:n}) - \sigma(0|\omega_{1:n}) \to 0$, as $n \to \infty$.

Comments. Hint for Item (c): use Items (a) and (b). Conclude from Item (a) that if ω is random relative to a computable measure μ, and the computable measure σ is chosen so that $\gamma_0(\omega_{1:n}|\sigma) = o(n)$, then the mean squared Hellinger distance $n^{-1} \sum_{i=1}^{n-1} \rho(\sigma(\cdot|\omega_{1:i}), \mu(\cdot|\omega_{1:i}))$ goes to 0. Item (c) gives a *randomness criterion* for objects with respect to a computable measure, in terms of Hellinger distance with a computable semimeasure with respect to which the object is known to be random. Source: V.G. Vovk, *Soviet Math. Dokl.*, 35(1987), 656–660. See also the estimate of prediction errors as in Theorem 5.2.1.

4.5.15. [28] Let $\mathcal{B}$ be a finite nonempty set of basic symbols. Let $\delta_0(\omega|\mu)$ be a universal sequential μ-test for sequences in $\mathcal{B}^\infty$ distributed according to a computable measure μ. Let ϕ be a monotone function as in Definition 4.5.3, page 298, that is μ-regular as in Definition 4.5.5.

(a) Show that $\delta_0(\omega|\mu) \geq \delta_0(\phi(\omega)|\mu_\phi) + K(\phi)$.

(b) Show that if μ_ϕ is a computable measure, then there exists a μ_ϕ-regular monotone function ψ such that $\mu_{\psi\phi} = \mu$ and $\delta_0(\phi(\omega)|\mu_\phi) \geq \delta_0(\omega|\mu) + K(\psi)$.

Comments. This generalizes Exercise 3.6.2, page 228. In particular it shows that a real number has a random binary representation iff it has a random representation in every base $r \geq 2$. Note that a sequence is not random in itself, but random with respect to a particular measure. Thus, if we recursively transform a sequence, then its randomness properties and the complexities of its initial segments are preserved up to an additive constant with respect to the transformed measure. Source: L.A. Levin, [*Problems Inform. Transmission*, 10:3(1974), 206–210; *Inform. Contr.*, 61(1984), 15–37].

4.5.16. [43] Sequences with maximal Kolmogorov complexity of the initial segments are random in Martin-Löf's sense of passing all effective statistical tests for randomness. Hence, they must satisfy laws like the law of the iterated logarithm.

(a) Show that if ω is an infinite sequence such that $Km(\omega_{1:n}) = n - o(\ln \ln n)$, then with $f_n = \omega_1 + \cdots + \omega_n$ we have

$$\limsup_{n\to\infty} \frac{|f_n - n/2|}{\sqrt{n \ln \ln n}} = \frac{1}{\sqrt{2}}.$$

(b) Show that if $Km(\omega_{1:n}) = n - o(n)$, then $\lim_{n\to\infty} f_n/n = \frac{1}{2}$. Conversely, for every $\epsilon > 0$ there is an ω with $Km(\omega_{1:n}) \geq (1-\epsilon)n$ for which the above doesn't hold.

(c) We say that an infinite binary sequence ω satisfies the infinite recurrence law if $f_n = \frac{1}{2}n$ infinitely often. Let $\epsilon > 0$. Prove the following:

(i) If $Km(\omega_{1:n}) \geq n - (\frac{1}{2} - \epsilon)\log n$, for all n, then ω is recurrent.

(ii) There is a nonrecurrent ω ($f_n/n > \frac{1}{2}$ for all but finitely many n) such that we have $Km(\omega_{1:n}) > n - (2+\epsilon)\log n$, for all n.

Comments. Item (a) is the law of the iterated logarithm. By Theorem 2.5.5, Item (a) holds for almost all infinite binary sequences ω. In other words, the law of the iterated logarithm holds for all infinite ω in a set that (obviously strictly) contains the Martin-Löf random sequences. Compare this with Equation 2.3 on page 167. There it was shown that for $C(x) \geq n - \delta(n)$, $n = l(x)$, we have $|f_n - \frac{1}{2}n| \leq \sqrt{n\delta(n)\ln 2}$. Item (b) is a form of the strong law of large numbers (Section 1.10). By Theorem 2.5.5 this gives an alternative proof that this law holds for almost all infinite binary sequences. Hint for the second part of Item (b): insert ones in an incompressible sequence at $1/\epsilon$-length intervals. Source: V.G. Vovk, *SIAM Theory Probab. Appl.,* 32(1987), 413–425.

4.6 Universal Average-Case Complexity, Continued

The discrete universal distribution has the remarkable property that the average computational complexity is of the same order of magnitude as the worst-case complexity, Section 4.4. What about the continuous version? This relates to algorithms for online computations that in principle never end. Such processes are abundant: sequence of key strokes from a computer keyboard; database insertions, deletions, and searches; network browser requests; and so on.

Formally, assume that the input sequence is infinite over the set $\mathcal{B}$ of basic symbols, say $\mathcal{B} := \{0,1\}$. Let the probability distribution of the inputs be a lower semicomputable semimeasure μ according to Definition 4.2.1 on page 263. Let A be an algorithm processing inputs $\omega = \omega_1\omega_2\ldots \in \{0,1\}^\infty$. The *computation time* of processing input $\omega_{1:n}$ up to the input of symbol ω_{n+1} is $t(\omega_{1:n})$.

Definition 4.6.1 Consider a continuous sample space $\{0,1\}^\infty$ with semimeasure μ. Let $t(\omega_{1:n})$ be the running time of algorithm A on initial segment instance $\omega_{1:n}$. Define the *worst-case time complexity* of A as $T(n) = \max\{t(\omega_{1:n}) : \omega_{1:n} \in \{0,1\}^n\}$. Define the *$\mu$-average time complexity* of A as

$$T(n|\mu) = \frac{\sum_{\omega_{1:n}} \mu(\omega_{1:n})t(\omega_{1:n})}{\sum_{\omega_{1:n}} \mu(\omega_{1:n})}.$$

Theorem 4.6.1 **(M-average complexity)** *Let A be an online algorithm with inputs in $\{0,1\}^\infty$. Let the inputs to A be distributed according to the universal semimeasure $\mathbf{M}$. Then, the average-case time complexity is of the same order of magnitude as the corresponding worst-case time complexity.*

Proof. Substitute μ for P and $\mathbf{M}$ for $\mathbf{m}$ in the proof of Theorem 4.4.1 on page 291. □

4.7 History and References

Napoleon's contemporary and friend Pierre-Simon Laplace, later Marquis de Laplace, may be regarded as the founder of the modern phase in the theory of probability. His anticipation of Kolmogorov complexity reasoning that we quoted at the beginning of this chapter occurs in the "sixth principle: the reason why we attribute regular events to a particular cause" of his *Essai philosophique sur les probabilités*. Similar sentiments were already formulated by the eccentric mathematician Girolamo Cardano (1501–1576). He seems to be the first to have made the abstraction from empiricism to theoretical concept for probability, in [G. Cardano, *Liber de Ludo Alae*, published posthumously in *Hieronymi Cardani Mediolanensis philosophi ac medici celeberrimi opera omnia, cura Car. Sponii*, Basle, 1663]. "To throw in a fair game at Hazards only three spots, when something great is at stake, or some business is the hazard, is a natural occurrence and deserves to be so deemed; and even when they come up the same way for a second time, if the throw be repeated. If the third and fourth plays are the same, surely there is occasion for suspicion on the part of a prudent man," [G. Cardano, *De Vita Propria Liber*, Milano, 1574].

The remarks of Dr. Samuel Johnson (1709–1784) are taken from the monumental J. Boswell, *Life of Johnson*, 1791, possibly the most famous biography written in the English language. For the basics of probability theory (as in Section 1.4) we have primarily used [A.N. Kolmogorov, *Grundbegriffe der Wahrscheinlichkeitsrechnung*, Springer-Verlag, 1933; English translation published by Chelsea, 1956].

The notion of semicomputable functions originates, perhaps, from L.A. Levin and A.K. Zvonkin [*Russ. Math. Surveys*, 25(1970), 83–124], but is so natural that it might have been used earlier (unknown to either of the authors of this book, or Levin). In the earlier editions of the current book we used 'enumerable function' for 'lower semicomputable function,' 'coenumerable function' for 'upper semicomputable function,' and 'computable function' for 'recursive function.' The notion of 'semimeasure' is most similar to 'lower measure' for nonmeasurable sets. They are also called 'defective measure' in W. Feller, *An Introduction to Probability Theory and Its Applications, Vol. II,* Wiley, 1970. In the current setting,

semimeasures were apparently used first by Levin and Zvonkin [*Russ. Math. Surveys*, 25(1970), 83–124], not by this name but in terms of an equivalent (but awkward) measure on a non-Hausdorff space denoted there by Ω^*. The name 'semimeasure,' together with the explicit definition as we used it, may originate from [L.A. Levin and V.V. Vyugin, pp. 359–364 in: Lect. Notes Comput. Sci. Vol. 53, Springer-Verlag, 1977]. The combination 'lower semicomputable probability distribution' (measure) as a framework for Solomonoff's approach below is due to Levin and Zvonkin [*Russ. Math. Surveys*, 25(1970), 83–124], but a related approach using computable probability distributions (measures) was given by D.G. Willis [*J. ACM*, 17(1970), 241–259]. R.J. Solomonoff used the notion of computable measures and informally noticed lower semicomputable semimeasures previously in [*Inform. Contr.*, 7(1964), 1–22].

Kolmogorov's introduction of complexity was motivated by information theory and problems of randomness. Solomonoff introduced algorithmic complexity independently and for a different reason: inductive reasoning. Universal a priori probability, in the sense of a single prior probability that can be substituted for each actual prior probability in Bayes's rule, was invented by Solomonoff, with Kolmogorov complexity as a side product, several years before anybody else did. R.J. Solomonoff obtained a Ph.B. (bachelor of philosophy) and M.Sc. in physics at the University of Chicago. He was already interested in problems of inductive inference and exchanged viewpoints with the resident philosopher of science Rudolf Carnap, who taught an influential course in probability theory [*Logical Foundations of Probability*, Univ. Chicago Press, 1950].

In 1956, Solomonoff attended the Dartmouth Summer Study Group on Artificial Intelligence, at Dartmouth College in Hanover, New Hampshire, organized by M. Minsky, J. McCarthy, and C.E. Shannon, and in fact stayed on to spend the whole summer there. (This meeting gave AI its name.) There Solomonoff wrote a memo on inductive inference. McCarthy had the idea that given every mathematical problem, it could be brought into the form of "given a machine and a desired output, find an input from which the machine computes that output." Solomonoff suggested that there was a class of problems that was not of that form: "given an initial segment of a sequence, predict its continuation." McCarthy then thought that if one saw a machine producing the initial segment, and then continuing past that point, would one not think that the continuation was a reasonable extrapolation? With that the idea got stuck, and the participants left it at that.

Later, Solomonoff presented the paper "An Inductive Inference Machine" at the IEEE Symposium on Information Theory, 1956, describing a program to unsupervisedly learn arithmetic formulas from examples. At the same meeting, there was a talk by N. Chomsky, based on his

paper "Three models for the description of language" [*IRE Trans. Inform. Theory*, IT-2(1956), 113–126]. The latter talk started Solomonoff thinking anew about formal machines in induction. In about 1958 he left his half-time position in industry and joined Zator Company full time, a small research outfit located in some rooms at $140\frac{1}{2}$ Mount Auburn Street, Cambridge, Massachusetts, which had been founded by Calvin Mooers sometime around 1954 for the purpose of developing information retrieval technology. Floating mainly on military funding, Zator Co. was a research front organization, employing Mooers, Solomonoff, Mooers's wife, and a secretary, as well as at various times visitors such as Marvin Minsky. It changed its name to the more martial sounding Rockford Research (Rockford, Illinois, was a place where Mooers had lived) sometime around 1962. In 1968 Solomonoff left and founded his own (one-man) company, Oxbridge Research, in Cambridge in 1970, and has been there ever since, apart from spending nine months as research associate at MIT's Artificial Intelligence Laboratory, 1990–1991 at the University of Saarland, Saarbrücken, Germany, and a more recent sabbatical at IDSIA, Lugano, Switzerland.

In 1960 Solomonoff published a report "A preliminary report on a general theory of inductive inference" [Tech. Rept. ZTB-138, Zator Company, Cambridge, Mass.] in which he gave an outline of the notion of universal a priori probability and how to use it in inductive reasoning (rather, prediction) according to Bayes's rule (Chapter 5). This was sent out to all contractors of the Air Force who were even vaguely interested in this subject. In his paper of 1964 [A formal theory of inductive inference, Part 1, *Inform. Contr.*, 7(1964), 1–22], Solomonoff developed these ideas further and defined the notion of enumeration of monotone machines and a notion of universal a priori probability based on the universal monotone machine.

In this way, it came about that the original incentive to develop a theory of algorithmic information content of individual objects was Ray Solomonoff's invention of a *universal* a priori probability that can be used instead of the actual a priori probability in applying Bayes's rule. His original suggestion in 1960 was to set the universal a priori probability $P(x)$ of a finite binary string x as $\sum 2^{-l(p)}$, the sum taken over all programs p with $U(p) = x$, where U is the reference Turing machine of Theorem 2.1.1 on page 105. However, using plain Turing machines this is improper, since not only does $\sum_x P(x)$ diverge, but $P(x)$ itself diverges for each x. To counteract this defect, Solomonoff in 1964 introduced a machine model tantamount to prefix machines/monotone machines. This left the problem of the corresponding $P(x)$ not being a probability measure. For this Solomonoff in 1964 suggested, and in 1978 exhibited, a normalization. However, the resulting probability measure is not even lower semicomputable. According to Solomonoff this is a small price

to pay. In fact, in some applications we may like the probability measure property and not care about semicomputability (Section 4.5.3 and Chapter 5). The universal distribution has remarkable properties and applications. Such applications are 'simple pac-learning' in Section 5.3; the MDL principle in statistical inference and learning, Section 5.4; and the notion and development of 'logical depth,' Section 7.7.

The remarkable property of $\mathbf{m}(\cdot)$, that the average computational complexity of every algorithm is always of the order of magnitude of the worst-case complexity, Sections 4.4 and 4.6, is from [M. Li and P.M.B. Vitányi, *Inform. Process. Lett.*, 42(1992), 145–149]. For computable versions of $\mathbf{m}(\cdot)$ this phenomenon is treated in Section 7.6. These considerations involve the maximal gain of average-case complexity over worst-case complexity for (algorithm, distribution) pairs and associated families such as polynomial-time algorithms and polynomial-time computable distributions, [P.B. Miltersen, *SIAM J. Comput.*, 22:1(1993), 147–156; K. Kobayashi, *IEICE Trans. Inform. Systems*, E76-D:6(1993), 634–640; K. Kobayashi, Transformations that preserve malignness of universal distributions, *Theoret. Comput. Sci.*, 181(1997), 289–306; A. Jakoby, R. Reischuk, and C. Schindelhauer, *Proc. 12th Symp. Theoret. Aspects Comput. Sci.*, 1995, pp. 628–639]. A.K. Jagota and K.W. Regan ["Performance of MAX-CLIQUE approximation heuristics under description-length weighed distributions," UB-CS-TR 92-24, SUNY at Buffalo, 1992] have extended Theorem 4.4.1 to approximation ratios for approximation algorithms. In this paper, they have also used a distribution $q(x)$ to approximate $\mathbf{m}(x)$ and performed extensive experiments for the MAX-CLIQUE problem. In their experiments, it was found that three out of nine algorithms perform much worse under $q(x)$ than under the uniform distribution, confirming the theory; six other algorithms showed not much difference.

Leonid A. Levin in 1970 gave a mathematical expression of a priori probability as a universal (that is, maximal) lower semicomputable semimeasure, Theorem 4.3.1, and showed that $\log 1/\mathbf{m}(x)$ coincides with $C(x)$ to within an additive term of $2\log C(x)$. In 1974 he explicitly introduced the notion of prefix machines and prefix complexity, and proved the remarkable Theorem 4.3.3, which can be designated as the *coding theorem*. To be able to formulate this theorem properly, we first recall the discrete form of the universal a priori probability, using the prefix complexity $K(x)$. In their 1970 paper [A.K. Zvonkin and L.A. Levin, *Russ. Math. Surveys*, 25(1970), 83–124], L.A. Levin analyzes the case of continuous semimeasures related to monotone machines and presents the construction of the universal lower semicomputable semimeasure, its equality with the universal a priori probability, and the universal randomness μ-test for arbitrary measures μ. See also [L.A. Levin, *Soviet Math. Dokl.*, 14(1973), 1477–1480]. The interpretation of this in terms

of discrete semimeasures and the restriction of monotone machines to prefix machines are also due to L.A. Levin [L.A. Levin, *Problems Inform. Transmission*, 10:3(1974), 206–210; P. Gács, *Soviet Math. Dokl.*, 15(1974), 1477–1480]. Prefix complexity was also introduced, independently, by G.J. Chaitin [*J. ACM*, 22(1975), 329–340], including Theorem 4.3.3. The whole theory is brought to majestic (and hard to understand) heights in [L.A. Levin, *Inform. Contr.*, 61(1984), 15–37]. The theory of universal continuous semimeasure $\mathbf{M}(\cdot)$ is used for induction and prediction in Section 5.2 and in the development of the notion of 'algorithmic entropy' in Section 8.6.

Solomonoff insists on the use of traditional probabilistic measures (such that $\mu(x0) + \mu(x1) = \mu(x)$). This led to some difficulties with his 1964 paper. $\mathbf{M}$ cannot be uniquely increased to a measure, and it is hard to choose a natural one among possible extensions (so optimality is lost). In the relevant (commissioned) Exercise 4.5.6, R.M. Solovay has shown that *every* such extension would both change $\mathbf{M}$ by more than a constant factor and destroy its algorithmic properties. However, in Exercise 4.5.7 on page 325, Solomonoff has shown that if we predict a measure μ using $\mathbf{M}_{\rm norm}$, then the changes with respect to $\mathbf{M}$ induced by $\mathbf{M}_{\rm norm}$ happen only with μ expectation going fast to 0 with growing length of the predicted sequence. Moreover, Solomonoff considers the increase by $1/o(1)$ to be a merit, and loss of lower semicomputability a small price to pay for it and for avoiding the heresy of redefining the notion of probability. It is not clear from his 1964 paper which extension he wanted (though it is clear that he meant to consider only ordinary probability measures), but in his 1978 paper he rigorously specifies an extension motivated as discussed in Section 4.5.3.

C.P. Schnorr [*Lect. Notes Math.*, Vol. 218, Springer-Verlag, 1971] introduced the use of martingales, due to P. Levy and used advantageously by J. Ville in 1939 [*Étude Critique de la Notion de Collectif*, Gauthier-Villars, 1939] in the study of Martin-Löf tests. Independently, C.P. Schnorr [*J. Comput. System Sci.*, 7(1973), 376–388] for the uniform distribution, and L.A. Levin [*Sov. Math. Dokl.*, 14(1973), 1413–1416] for arbitrary computable distributions, introduced the monotone variant of complexity $Km(x)$ in about 1973. The monotone complexity Km smoothes out the oscillations in order to characterize randomness. Monotone complexity obliterates all quantitative differences among Martin-Löf random sequences, and hence does not allow us to make distinctions in randomness properties, in contrast to K complexity [M. van Lambalgen, *Random Sequences*, Ph.D. thesis, University of Amsterdam, 1987; *J. Symb. Logic*, 54(1989), 1389–1400]. L.A. Levin [*Soviet Math. Dokl.*, 14(1973), 1413–1416] introduced monotone complexity, and C.P. Schnorr [*J. Comput. System Sci.*, 7(1973), 376–388] introduced another complexity, which he called 'process complexity.' The difference between those

two complexities is not bounded by any constant [V.V. Vyugin, *Semiotika i Informatika*, 16(1981), 14–43 (p. 35); English translation: *Selecta Mathematica* formerly *Sovietica*, 13:4(1994), 357–389]. C.P. Schnorr in [*Basic Problems in Methodology and Linguistics*, R.E. Butts and J. Hintikka, eds., Reidel, 1977, pp. 193–210] introduced a variant of monotone complexity coinciding up to an additive constant with Levin's variant. For further historical notes see [A.N. Kolmogorov and V.A. Uspensky, *SIAM J. Theory Probab. Appl.*, 32(1987), 387–412].

C.P. Schnorr's later definition is as follows: A partial recursive function $\phi : \{0,1\}^* \times \mathcal{N} \to \{0,1\}^*$ is called a *monotone* interpreter if

(i) for all (p,n) in the domain of ϕ we have that $l(\phi(p,n)) = n$, and

(ii) for all $(p,n), (pq, n+k)$ in the domain of ϕ we have that $\phi(p,n)$ is a prefix of $\phi(pq, n+k)$.

We can think of monotone interpreters as being computed by *monotone machines* according to Schnorr, which are Turing machines with two one-way read-only input tapes containing p and n, respectively; some work tapes; and a one-way write-only output tape. The output tape is written in binary, and the machine halts after it outputs n bits.

Define $Km_\phi(x) = \min\{l(p) : \phi(p, l(x)) = x\}$, and $Km_\phi(x) = \infty$ if such p does not exist. There is an additively optimal monotone interpreter ϕ_0 such that for any other monotone interpreter ϕ there is a constant c such that $Km_{\phi_0}(x) \leq Km_\phi(x)+c$ for all x. Select one such ϕ_0 as reference and set the *monotone complexity* according to Schnorr as $Km(x) = Km_{\phi_0}(x)$. Similarly, we can define the conditional monotone complexity $Km(x|y)$.

L.A. Levin used another definition. Instead of a function, the definition uses a recursively enumerable relation $A(p,x)$ with the property that if p is a prefix of q and $A(p,x)$, $A(q,y)$ hold, then x, y must be compatible (one is a prefix of the other). The meaning is that our machine on input p outputs a (possibly infinite) string with prefix x. The minimum length of such an input p is the monotone complexity $Km_A(x)$ with respect to A according to Levin. Among all such recursively enumerable relations there is a universal one, say U, such that for each A above there is a constant c such that for all x, we have $Km_U(x) \leq Km_A(x) + c$. We fix such a U and define the *monotone complexity according to Levin* as $Km(x) = Km_U(x)$. Let us call this a type 1 monotone machine.

The following definition of monotone machines is not equivalent but also appropriate. We require that if p is a prefix of q and $A(p,x)$ and $A(q,y)$ hold then x is a prefix of y. Let us call this a type 2 monotone machine.

There is yet another definition, the apparently most obvious one. The machine T has a one-way input tape and a one-way output tape. It keeps reading input symbols and emitting output symbols. For a (possibly

infinite) string x we write $T(p) = x$ if T outputs x after reading p and no more. Let us call this a type 3 monotone machine. This type is used in the main text.

The monotone complexities arising from these three different kinds of machine are not necessarily the same. But all interesting upper bounds work for type 3 (the largest), and P. Gács's theorem distinguishing Km from KM works for type 1 (the smallest). See [C.P. Schnorr, *J. Comput. Syst. Sci.*, 7(1973), 376–388; A.K. Zvonkin and L.A. Levin, *Russ. Math. Surveys*, 25:6(1970), 83–124, attributed to L.A. Levin; L.A. Levin, *Sov. Math. Dokl.*, 14(1973), 1413–1416; P. Gács, *Theoret. Comput. Sci.*, 22(1983), 71–93] and on generalization of monotone complexity [A.K. Shen, *Sov. Math. Dokl.*, 29:3(1984), 569–573].

The relation between different complexities in the table of Figure 4.2 is asserted (many relations without proofs) in V.A. Uspensky's survey [*Kolmogorov Complexity and Computational Complexity*, O. Watanabe, ed., Springer-Verlag, 1992, pp. 85–101]. Many of the missing proofs or references are provided in [V.A. Uspensky and A.K. Shen, *Math. Systems Theory*, 29(1996), 271–292]. The most difficult relation, the lower bound on $Km(x) - KM(x)$, which infinitely often exceeds a variant of the nonprimitive-recursive slow-growing inverse of the Ackermann function (Exercise 1.7.18, page 45), is due to P. Gács [*Theoret. Comput. Sci.*, 22(1983), 71–93].

In the dictionary the word 'martingale' is defined as (a) a betting system; (b) part of a horse's harness; (c) part of a sailing rig. The delightful remark of Thackeray was quoted second hand from [J. Laurie Snell, *Mathematical Intelligencer*, 4:3(1982)]. The mathematical study of martingales was started by P. Levy and continued by J. Ville [*Etude Critique de la Concept du Collectif*, Gauthier-Villars, 1939] in connection with von Mises's notion of a random sequence. Ville showed that von Mises–Wald–Church random sequences defined in Section 1.9 do not satisfy all randomness properties; see the exercises in Section 1.9. But he also developed martingale theory as a tool in probability theory. A successful application of martingales was by J.L. Doob in the theory of stochastic processes in probability theory. In connection with random sequences in the sense of Martin-Löf, the martingale approach was first advocated and developed by C.P. Schnorr [*Lect. Notes Math.*, Vol. 218, Springer-Verlag, 1971]. Schnorr gives an overview of his work in [pp. 193–210 in: *Basic Problems in Methodology and Linguistics*, R.E. Butts, J. Hintikka, eds., D. Reidel, 1977]. See also [R. Heim, *IEEE Trans. Inform. Theory*, IT-25(1979), 558–566] for relations between computable payoff functions, martingales, algorithmic information content, effective random tests, and coding theorems. The material used here is gleaned from [P. Gács, *Lecture Notes on Descriptional Complexity and Randomness*, Manuscript, Boston University, 1987; T.M. Cover, P. Gács and

R.M. Gray, *Ann. Probab.*, 17:3(1989), 840–865]. A survey of the basics of algorithmic probability and its relation to universal betting and to prefix complexity is given in [T.M. Cover and J.A. Thomas, *Elements of Information Theory*, Wiley, 1991]. The election example is perhaps due to L.A. Levin. In general, the material on exact expressions of universal randomness μ-tests is partially due to unpublished work of L.A. Levin, and is based on [P. Gács, *Z. Math. Logik Grundl. Math.*, 28(1980), 385–394; *Theoret. Comput. Sci.*, 22(1983), 71–93] and personal suggestions of P. Gács. For the great developments in the last decades on the crossroads of randomness, Kolmogorov complexity, and recursion theory, we refer to the specialized treatments mentioned in the history and references sections of Chapters 2 and 3.

5

Inductive Reasoning

5.1 Introduction

The *Oxford English Dictionary* defines **induction** as "the process of inferring a general law or principle from the observations of particular instances."

inductive inference. On the other hand, we regard *inductive reasoning* as a more general concept than inductive inference, as a process of reassigning a probability (or credibility) to a law or proposition from the observation of particular instances.

In other words, inductive inference draws conclusions that *accept or reject* a proposition, possibly without total justification, while inductive reasoning only changes the degree of our belief in a proposition. In *deductive reasoning* one derives the absolute truth or falsehood of a proposition, such as when a mathematical proposition is proved from axioms. In deduction one often discards information: from the conclusion one cannot necessarily deduce the assumptions. In induction one generally increases information but does not discard information: the observed data follow from the induced law. In this view, deduction may be considered a special form of induction.

5.1.1 Epicurus's Principle

Inductive reasoning dates back at least to the Greek philosopher of science Epicurus (342?–270? B.C.), who proposed the following approach:

> **Principle of Multiple Explanations.** If more than one theory is consistent with the observations, keep all theories.

M. Li and P.M.B. Vitányi, *An Introduction to Kolmogorov Complexity and Its Applications*,
DOI: 10.1007/978-0-387-49820-1_5, © Springer Science + Business Media, LLC 2008

In his *Letter to Pythocles*, Epicurus motivates this as follows. There are cases, especially of events in the heavens such as the risings and settings of heavenly bodies and eclipses, where it is sufficient for our happiness that several explanations be discovered. In these cases, the events "have multiple causes of coming into being and a multiple predication of what exists, in agreement with the perceptions."

When several explanations are in agreement with the (heavenly) phenomena, we must keep all of them for two reasons. Firstly, the degree of precision achieved by multiple explanations is sufficient for human happiness. Secondly, it would be unscientific to prefer one explanation to another when both are equally in agreement with the phenomena. This, he claims, would be to "abandon physical inquiry and resort to myth." His follower Lucretius (95–55 B.C.) considered multiple explanations as a stage in scientific progress. According to him, to select one explanation from several equally good ones is not appropriate for the person who would "proceed step by step:"

"There are also some things for which it is not enough to state a single cause, but several, of which one, however, is the case. Just as if you were to see the lifeless corpse of a man lying far away, it would be fitting to state all the causes of death in order that the single cause of this death may be stated. For you would not be able to establish conclusively that he died by the sword or of cold or of illness or perhaps by poison, but we know that there is something of this kind that happened to him." [Lucretius]

In the calculus of probabilities it has been customary to postulate the 'principle of indifference' or the 'principle of insufficient reason.' The principle of indifference considers events to be equally probable if we have not the slightest knowledge of the conditions under which each of them is going to occur. When there is an absolute lack of knowledge concerning the conditions under which a die falls, we have no reason to assume that a certain face has a higher probability of coming up than another. Hence, we assume that each outcome of a throw of the die has probability $\frac{1}{6}$.

[Bertrand's paradox] The principle of indifference is not without difficulties. Consider the following elegant paradox. We are given a glass containing a mixture of water and wine. All that is known is that $1 \leq \text{water/wine} \leq 2$. The principle of indifference then tells us that we should assume that the probability that the ratio lies between 1 and $\frac{3}{2}$ is 0.5 and the probability that the ratio lies between $\frac{3}{2}$ and 2 is also 0.5. Let us take a different approach. We know $\frac{1}{2} \leq \text{wine/water} \leq 1$. Hence by the same principle the probabilities that this new ratio lies in the intervals of $\frac{1}{2}$ to $\frac{3}{4}$ and $\frac{3}{4}$ to 1 should each be 0.5. Thus, according to the second calculation, there is 0.5 probability such that the water/wine ratio lies between 1 to $\frac{4}{3}$ and 0.5 probability such that the water/wine ratio lies between $\frac{4}{3}$ to 2. But the two hypotheses are incompatible.

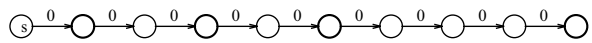


FIGURE 5.1. Trivial consistent automaton

5.1.2 Occam's Razor

The second and more sophisticated principle is the celebrated Occam's razor principle commonly attributed to William of Ockham (1290?–1349?). This was formulated about fifteen hundred years after Epicurus. In sharp contrast to the principle of multiple explanations, it states:

> **Occam's Razor Principle.** Entities should not be multiplied beyond necessity.

According to Bertrand Russell, the actual phrase used by William of Ockham was, "It is vain to do with more what can be done with fewer." This is generally interpreted as, 'among the theories that are consistent with the observed phenomena, one should select the simplest theory.' Isaac Newton (1642–1727) states the principle as rule 1 for natural philosophy in the *Principia*:

"We are to admit no more causes of natural things than such as are both true and sufficient to explain the appearances. To this purpose the philosophers say that Nature does nothing in vain, and more is in vain when less will serve; for Nature is pleased with simplicity, and affects not the pomp of superfluous causes." [Newton]

In Newton's time, 'the Philosopher' meant Aristotle (ca. 384–322 B.C.), who states in his *Posterior Analytics*, anticipating Ockham, as presumably known by the latter:

"We may assume the superiority ceteris paribus [other things remaining equal] of the demonstration which derives from fewer postulates or hypotheses—in short, from fewer premises." [Aristotle]

Example 5.1.1 A *deterministic finite automaton* (DFA) A has a finite number of states, including a starting state and some accepting states. At each step, A reads the next input symbol and changes its state according to the current state and the input symbol. Let us measure simplicity by the number of states in the automaton. The sample data are

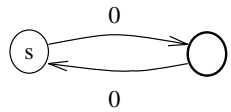


FIGURE 5.2. Smallest consistent automaton

Accepted inputs: 0, 000, 00000, 000000000;

Rejected inputs: ϵ, 00, 0000, 000000.

There are infinitely many finite automata that are consistent with these data. Figure 5.1 shows the trivial automaton, which simply encodes the data. Figure 5.2 shows the simplest automaton. The marker S indicates the starting state, and the bold circles are the accepting states.

Since the automaton in Figure 5.1 simply literally encodes the data, we do not expect that the machine anticipates future data. On the other hand, the automaton in Figure 5.2 makes the plausible *inference* that the language accepted consists of strings of an odd number of 0's. It selects the simplest described division of the positive and negative data. It therefore also anticipates data it has not yet seen and that do not logically follow from the observed data. The latter appeals to our intuition as a reasonable inference. ◇

A too simplistic application of Occam's razor may also lead to nonsense, as the following story illustrates. Once upon a time, there was a little girl named Emma. Emma had never eaten a banana, nor had she ever been on a train. One day she had to journey from New York to Pittsburgh by train. To relieve Emma's anxiety, her mother gave her a large bag of bananas. At Emma's first bite of her banana, the train plunged into a tunnel. At the second bite, the train broke into daylight again. At the third bite, Lo! into a tunnel; the fourth bite, La! into daylight again. And so on all the way to Pittsburgh. Emma, being a bright little girl, told her grandpa at the station, "Every odd bite of a banana makes you blind; every even bite puts things right again." Freely adapted from [N.R. Hanson, *Perception and Discovery*, 1969, Freeman and Cooper, p. 359].

In the learning automaton example, it turns out that one can *prove* the following: If sufficient data are drawn randomly from any fixed distribution, then with high probability the smallest consistent automaton (or a reasonably small automaton) will with high probability correctly predict acceptance or rejection of most data that are drawn afterward from this distribution.

Example 5.1.2 In spite of common intuitive acceptance of Occam's razor, the notion of simplicity remains a highly controversial and elusive idea. Things are subtler than they seem. For example, consider the following seemingly innocent rule:

> Select a hypothesis that is as well in agreement with the observed value as possible; if there is any choice left, choose the simplest possible hypothesis.

Let there be an unknown number of white balls and black balls in a sealed urn. We randomly draw one ball at a time, note its color and replace it, and shake the urn thoroughly. After n draws we must decide what fraction of the balls in the urn is white. The possible hypotheses state that some rational fraction r of balls in the urn is white, where $0 < r < 1$. By the above rule, if in n draws m white balls are selected, then we should formulate the hypothesis $r = m/n$. Let there be $\frac{1}{3}$ white and $\frac{2}{3}$ black balls. Then the probability of getting the true hypothesis $r = \frac{1}{3}$ is zero if n is not divisible by 3, and it tends to 0, even under the assumption that n is divisible by 3. Even for a sequence of draws for which the process does converge, convergence may be too slow for practical use. ◇

We still have not defined 'simplicity.' How does one define it? Is $\frac{1}{4}$ simpler than $\frac{1}{10}$? Is $\frac{1}{3}$ simpler than $\frac{2}{3}$? Note that saying that there are $\frac{1}{3}$ white balls in the urn is the same as that there are $\frac{2}{3}$ black balls. If one wants to infer polynomials, is $x^{100}+1$ more complicated than $13x^{17}+5x^3+7x+11$?

Can a thing be simple under one definition of simplicity and not simple under another? The contemporary philosopher Karl R. Popper (1902–1994) has said that Occam's razor is without sense, since there is no objective criterion for simplicity. Popper states that every such proposed criterion will necessarily be biased and subjective.

It is widely believed that the better a theory compresses the data concerning some phenomenon under investigation, the better we learn, generalize, and the better the theory predicts unknown data. This is the basis of the Occam's razor paradigm about simplicity. Making these ideas rigorous involves the length of the shortest effective description of the theory, its Kolmogorov complexity, which is the size in bits of the shortest binary program to compute a description of the theory on a universal computer. This complexity, although defined in terms of a particular machine model, is independent up to an additive constant and acquires an asymptotically universal and absolute character through Church's thesis, from the ability of universal machines to simulate one another and execute any effective process. This train of thought will lead us to a rigorous mathematical relation between data compression and learning.

5.1.3 Bayes's Rule

In contrast to Epicurus and Ockham, Thomas Bayes took a probabilistic view of nature. Assume that we have observational data D.

> **Bayes's Rule.** The probability of hypothesis H being true is proportional to the learner's initial belief in H (the prior probability) multiplied by the conditional probability of D given H.

The two fundamental components in the general inductive reasoning theory we are developing are Bayes's formula and the universal prior probability. They both bear the same characteristics: superficially trivial but philosophically deep. We have studied the mathematical theory of the universal distribution in Chapter 4. In Section 5.2 we will develop the underlying mathematics and validation of Solomonoff's predictive theory. But first we develop Bayesian theory starting from the motivation in Section 1.10 and the formal definition in Section 1.6.

Consider a situation in which one has a set of observations of some phenomenon and also a (possibly countably infinite) set of hypotheses that are candidates to explain the phenomenon. For example, we are given a coin and we flip it 100 times. We want to identify the probability that the coin has outcome 'heads' in a single coin flip. That is, we want to find the bias of the coin. The set of possible hypotheses is uncountably infinite if we allow each real bias in $[0,1]$, and countably infinite if we allow each rational bias in $[0,1]$.

For each hypothesis H we would like to assess the probability that H is the true hypothesis, given the observation of D. This quantity, $\Pr(H|D)$, can be described and manipulated formally in the following way:

Let S be a discrete sample space, and let D denote a sample of outcomes, say experimental data concerning a phenomenon under investigation. Let $H_1, H_2, \ldots$ be an enumeration of countably many hypotheses concerning this phenomenon, say each H_i is a probability distribution over S. The list $\mathcal{H} = \{H_1, H_2, \ldots\}$ is called the *hypothesis space*. The hypotheses H_i are exhaustive (at least one of them is true) and mutually exclusive (at most one of them is true).

For example, say the hypotheses enumerate the possible rational (or computable) biases of the coin. As another possibility there may be only two possible hypotheses: hypothesis H_1, which says that the coin has bias 0.2, and hypothesis H_2, which puts the bias at 0.8.

Let the prior distribution of the probabilities $P(H)$ of the various possible hypotheses in $\mathcal{H}$, and the data sample D, be given or prescribed. Because the list of hypotheses is exhaustive and mutually exclusive we have $\sum_i P(H_i) = 1$. In the context of Bayesian reasoning we will distinguish between the notation P for probabilities that are prescribed (can be chosen freely), and Pr for a probabilities that are determined by (and often can be computed from) the prescribed items. Thus, we assume that for all $H \in \mathcal{H}$ we can compute the probability $\Pr(D|H)$ that sample D arises if H is the case. Then we can also compute (or approximate in case the number of hypotheses with nonzero probability is infinite) the probability $\Pr(D)$ that sample D arises at all

$$\Pr(D) = \sum_i \Pr(D|H_i)P(H_i).$$

From the definition of conditional probability it is easy to derive Bayes's rule (Example 1.6.3, page 19),

$$\Pr(H_i|D) = \frac{\Pr(D|H_i)P(H_i)}{\Pr(D)}, \tag{5.1}$$

and substitution yields

$$\Pr(H_i|D) = \frac{\Pr(D|H_i)P(H_i)}{\sum_i \Pr(D|H_i)P(H_i)}.$$

Despite the fact that Bayes's rule essentially rewrites the definition of conditional probability, and nothing more, it is its interpretation and application that are profound and controversial. The different H's represent the possible alternative hypotheses concerning the phenomenon we wish to discover. The term D represents the empirically or otherwise known data concerning this phenomenon. The term $\Pr(D)$, the probability of data D, is considered as a normalizing factor so that $\sum_i \Pr(H_i|D) = 1$.

The term $P(H_i)$ is called the *a priori, initial*, or *prior* probability of hypothesis H_i. It represents the probability of H_i being true before we have obtained any data. The prior probability is often considered as the learner's *initial degree of belief* in the hypothesis concerned.

The term $\Pr(H_i|D)$ is called the *final*, *inferred*, or *posterior* probability, which represents the adapted probability of H_i after seeing the data D. In essence, Bayes's rule is a mapping from prior probability $P(H_i)$ to posterior probability $\Pr(H_i|D)$ determined by data D.

Continuing to obtain more and more data, and repeatedly applying Bayes's rule using the previously obtained inferred probability as the current prior, eventually the inferred probability will concentrate more and more on the true hypothesis. It is important to understand that one can find the true hypothesis also, using many examples, by the law of large numbers. In general, the problem is not so much that in the limit the inferred probability would not concentrate on the true hypothesis, but that the inferred probability gives as much information as possible about the possible hypotheses from only a limited number of data. Given the prior probability of the hypotheses, it is easy to obtain the inferred probability, and therefore to make informed decisions.

In many learning situations, if the data are consistent with the hypothesis H_i, in the strict sense of being forced by it, then $\Pr(D|H_i) = 1$. Example: outcome of a throw with a die is 'even' while the hypothesis says 'six.' If the data are inconsistent with the hypothesis, then $\Pr(D|H_i) = 0$. We assume that there is no noise that distorts the data.

Example 5.1.3 We reconsider the example given in Section 1.9. An urn contains many dice with different biases of outcome 6 in a random throw. The set of

biases is A. Assume that the difference between each pair of biases is greater than 2δ with $\delta > 0$ and the biases are properly between 0 and 1. Randomly drawing a die from the urn, our task is to determine its bias. This is done by experimenting. We throw the die n times, independently. If 6 shows up m times, then our learning algorithm chooses the least bias in A that is nearest to m/n.

Let H_p be the event of drawing a die with bias p for outcome 6 from an urn. A *success* is a throw with outcome 6. For $q \in A$, let D_q be an example of experimental data such that m successes (6's) were observed out of n throws and $|(m/n) - q| \leq \delta$. Then,

$$\Pr(H_p|D_q) = \frac{\Pr(D_q|H_p)P(H_p)}{\Pr(D_q)},$$

where $\Pr(D_q) = \sum_{i \in A} \Pr(D_q|H_i)P(H_i)$. With H_p being true, the probability of m successes out of n throws is given by the binomial distribution, Equation 1.6 on page 61, as

$$\binom{n}{m} p^m (1-p)^{n-m}.$$

The deviation ϵ (where $0 \leq \epsilon \leq 1$) from the average number of successes pn in a series of n experiments is analyzed by estimating the combined tail probability

$$P(|m - pn| > \epsilon pn) = \sum_{|m-pn|>\epsilon pn} \binom{n}{m} p^m (1-p)^{n-m}$$

of the binomial distribution. The estimates are given by Chernoff's bound of Lemma 1.10.1 on page 61,

$$P(|m - pn| > \epsilon pn) \leq 2e^{-\epsilon^2 pn/3}.$$

Let p be the true attribute of the die we have drawn, and define $A(p) = A - \{p\}$. For every $q \in A(p)$, the value of m/n to force us to infer a bias of q instead of p must deviate from p by more than δ. Thus, to select some hypothesis D_q instead of the true hypothesis D_p, we must have $|m/n - p| > \delta$. By Chernoff's bound, with $\epsilon = \delta/p$, we obtain

$$P\left(\left|\frac{m}{n} - p\right| > \delta\right) \leq 2^{-\delta^2 n/3p}.$$

Therefore, $\sum_{q \in A(p)} \Pr(D_q|H_p) < 2^{-\Omega(n)}$. (We have assumed that $|p - q| \geq 2\delta$ for every $q \in A(p)$. and $0 < p, q < 1$.) Hence, the probability $\Pr(D_p|H_p) \geq 1 - 1/2^{\Omega(n)}$. Altogether this means that the posterior probability $\Pr(H_p|D_p)$ goes to 1 exponentially fast with n (because $\Pr(H_q|D_p)$ goes to 0 as fast, for $q \in A(p)$).

From the Bayesian formula one can see that if the number of trials is small, then the posterior probability $\Pr(H_p|D_q)$ may strongly depend on the prior probability $P(H_p)$. When n grows large, the posterior probability condenses more and more around m/n. ◇

In real-world problems, the prior probabilities may be unknown, incomputable, or conceivably may not exist. (What is the prior probability of use of words in written English? There are many different sources of different social backgrounds living in different ages.) This problem would be solved if we could find a *single* probability distribution to use as the prior distribution in each different case, with approximately the same result as if we had used the real distribution. Surprisingly, this turns out to be possible up to some mild restrictions on the class of prior distributions being taken into account.

5.1.4 Hume on Induction

The philosopher D. Hume (1711–1776) argued that true induction is impossible because we can reach conclusions only by using known data and methods. Therefore, the conclusion is logically already contained in the start configuration. Consequently, the only form of induction possible is deduction. Philosophers have tried to find a way out of this deterministic conundrum by appealing to probabilistic reasoning such as using Bayes's rule. One problem with this is where the prior probability one uses has to come from. Unsatisfactory solutions have been proposed by philosophers such as R. Carnap (1891–1970) and K.R. Popper.

However, R.J. Solomonoff's inductive method of Section 5.2, of which we have already seen a glimpse in Section 1.10, may give a rigorous and satisfactory solution to this old problem in philosophy.

Essentially, combining the ideas of Epicurus, Ockham, Bayes, and modern computability theory, Solomonoff has successfully invented a perfect theory of induction. It incorporates Epicurus's multiple explanations idea, since no hypothesis that is still consistent with the data will be eliminated. It incorporates Ockham's simplest explanation idea, since the hypotheses with low Kolmogorov complexity are more probable. The inductive reasoning is performed by means of the mathematically sound rule of Bayes.

5.1.5 Hypothesis Identification and Prediction by Compression

Our aim is to demonstrate that data compression is the answer to many questions about how to proceed in inductive reasoning. Given a body of data concerning some phenomenon under investigation, we want to select the most plausible hypothesis from among all appropriate hypotheses or predict future data. It is widely believed that the better a theory compresses the data concerning some phenomenon under investigation, the better we have learned and eneralized, and the better the theory predicts unknown data, following the Occam's razor paradigm

about simplicity. This belief is vindicated in practice but apparently has not been rigorously proved before. Making these ideas rigorous involves the length of the shortest effective description of some object: its Kolmogorov complexity. We treat the relation between data compression and learning and show that compression is almost always the best strategy, both in hypothesis identification using the minimum description length (MDL) principle and in prediction methods in the style of R.J. Solomonoff. The flavor of the argument is that among all hypotheses consistent with the data, the one with least Kolmogorov complexity is the most likely one. *Prediction* in Solomonoff's manner uses a complexity-weighted combination of all hypotheses in the form of the universal prior $\mathbf{M}(\cdot)$ (Section 5.2). Applications of the universal prior to recursion theory are given in Section 5.2.5, and to computer science in Section 5.3. *Hypothesis identification* by minimum description length (Section 5.4) balances the complexity of the model—and its tendency for overfitting—against the preciseness of fitting the data—the error of the hypothesis. Nonprobabilistic statistics in Section 5.5 gives a detailed view of all stochastic properties of data, and, among others, a rigorous foundation and justification of MDL.

5.2 Solomonoff's Theory of Prediction

Let us consider theory formation in science as the process of obtaining a compact description of past observations together with predictions of future ones. Ray Solomonoff argues that the preliminary data of the investigator, the hypotheses he proposes, the experimental setup he designs, the trials he performs, the outcomes he obtains, the new hypotheses he formulates, and so on, can all be encoded as the initial segment of a potentially infinite sequence over a finite alphabet. The investigator obtains increasingly longer initial segments of an infinite sequence ω by performing more and more experiments on some aspect of nature. To describe the underlying regularity of ω, the investigator tries to formulate a theory that governs ω on the basis of the outcome of past experiments. Candidate theories (hypotheses) are identified with computer programs that compute sequences starting with the observed initial segment.

There are many different possible infinite sequences (histories) on which the investigator can embark. The phenomenon he wants to understand or the strategy he uses can be stochastic. In this view each phenomenon can be identified with a measure on the continuous sample space. (For the definition of measure see Section 1.6.)

We express the task of learning a certain concept in terms of sequences over a basic alphabet $\mathcal{B}$. We express what we know as a finite sequence over $\mathcal{B}$; an experiment to acquire more knowledge is encoded as a sequence over $\mathcal{B}$; the outcome is encoded over $\mathcal{B}$; new experiments are encoded over $\mathcal{B}$; and so on. In this way, a concept can be viewed as a

probability distribution (rather, measure) μ over a sample space $S = \mathcal{B}^\infty$ of all one-way infinite sequences. Each such sequence corresponds to one never-ending sequential history of conjectures and refutations and confirmations. The distribution μ can be said to be the concept or phenomenon involved.

Clearly, if we know μ, then we can best predict which element $a \in \mathcal{B}$ is likely to turn up after a finite initial segment x. That is, we want to predict or extrapolate according to the conditional distribution $\mu(a|x)$. Below we show (by Bayes's rule) that $\mu(a|x)$ satisfies Equation 5.2.

Example 5.2.1 Let $\mathcal{B} = \{0, \ldots, 9\}$. The phenomenon consists of those sequences $\omega = \omega_1\omega_2\ldots$ for which ω_{2i+1} is the ith digit ($i = 0, 1, \ldots$) in the decimal expansion of $\pi = 3.1415\ldots$, and $\omega_{2i} = a$, with $a \in \{0, \ldots, 9\}$, with equal probabilities $\frac{1}{10}$. Some example values of the measure μ describing the phenomenon are $\mu(3) = 1$, $\mu(4) = 0$, $\mu(31) = \frac{1}{10}$, $\mu(41) = 0$, $\mu(311) = \frac{1}{10}$, $\mu(314) = 0$, $\mu(3114) = \frac{1}{100}$. Prediction should follow conditional probabilities: $\mu(0|3) = \frac{1}{10}$, $\mu(0|31) = 0$, $\mu(1|31) = 1$, and $\mu(4|311) = \frac{1}{10}$, while $\mu(4|3114) = 1$. But $\mu(3|3114) = 0$ and $\mu(3|311) = \frac{1}{10}$. ◇

Example 5.2.2 Let our basic alphabet be $\mathcal{B} = \{0, \ldots, 9, E, O, H, .\}$. Numbers can be written as decimals such as 5.1 because we have a decimal point in our system. The phenomenon μ that we would like to know about is the value of the gravitational constant g in the formula $h = gt^2/2$, where h is the height from which we drop an object in a vacuum (on Earth at about sea level), and t is the time it takes to hit the ground. An experiment $E10$ means that we drop the object from 10 meters. An outcome $O2$ means that the object takes 2 seconds to hit the ground. A hypothesis $H7.8$ means that we hypothesize $g = 7.8$ meters per second squared. Then, $\mu(O2|H9.8E19.6) = 1$, $\mu(O2|H1E2) \ll 1$, and $\mu(O1|H9.8E4.9) = 1$, because $g = 9.8$. Experimenting, the investigator will eventually discover the law of μ and be able to confidently predict μ after every initial sequence w, by $\mu(Ox|HyEz) = 1$ if $z = yx^2/2$ with $y = 9.8$, and $\ll 1$ otherwise. ◇

5.2.1 Universal Prediction

Following Examples 5.2.1 and 5.2.2, the aim is to *predict* outcomes concerning a phenomenon μ under investigation. In this case we have some prior evidence (prior distribution over the hypotheses, experimental data) and we want to predict future events. This situation can be modeled by considering a sample space S of one-way infinite sequences of basic elements $\mathcal{B}$ defined by $S = \mathcal{B}^\infty$. We assume a prior distribution μ over S with $\mu(x)$ denoting the probability of a sequence starting with x. Here $\mu(\cdot)$ is a *semimeasure* satisfying

$$\mu(\epsilon) \leq 1,$$

$$\mu(x) \geq \sum_{a \in \mathcal{B}} \mu(xa).$$

As before, we depart from the traditional notation $\mu(\Gamma_x)$, where the *cylinder* Γ_x is equal to $\{\omega \in S : \omega \text{ starts with } x\}$, by writing $\mu(x)$ instead. We use $\mu(x)$ for convenience. If equalities hold in the definition then μ is called a *measure*.

Given a previously observed data string x, the inference problem is to predict the next symbol in the output sequence, that is, to extrapolate the sequence x. In terms of the variables in Equation 5.1, H_{xy} is the hypothesis that the sequence starts with initial segment xy. Data D_x consists of the fact that the sequence starts with initial segment x. Then, $\Pr(D_x|H_{xy}) = 1$, that is, the data are forced by the hypothesis; or $\Pr(D_z|H_{xy}) = 0$ for z is not a prefix of xy, that is, the hypothesis contradicts the data. For $P(H_{xy})$ and $\Pr(D_x)$ in Equation 5.1 we substitute $\mu(xy)$ and $\mu(x)$, respectively. For $\Pr(H_{xy}|D_x)$ we substitute $\mu(y|x)$. In this way, the formula is rewritten as

$$\mu(y|x) = \frac{\mu(xy)}{\mu(x)}. \tag{5.2}$$

The final probability $\mu(y|x)$ is the probability of the next symbol string being y, given the initial string x. Obviously we now need only the prior probability μ to evaluate $\mu(y|x)$. The goal of inductive inference in general is to be able either to (i) predict, or extrapolate, the next element after x or (ii) to infer an underlying effective process that generated x, and hence to be able to predict the next symbol. In the most general deterministic case such an effective process is a Turing machine, but it can also be a probabilistic Turing machine or, say, a Markov process (which makes its brief and single appearance here). The central task of inductive inference is to find a universally valid approximation to μ that is good at estimating the conditional probability that a given segment x will be followed by a segment y.

In general this is impossible. But suppose we restrict the class of priors to the *computable* semimeasures as defined in Chapter 4. Under this relatively mild restriction, it turns out that we can use the single universal semimeasure $\mathbf{M}$ (Theorem 4.5.1, page 294) as a universal prior (replacing the real prior) for prediction. The remainder of this section is devoted to the formal expression and proof of this loosely stated fact. Subsequently we demonstrate that ultimate compression and shortest programs almost always lead to optimal prediction.

Definition 5.2.1 Let $\mu : \mathcal{B}^* \to [0,1]$ be a semimeasure and let $x, y \in \mathcal{B}^*$. The conditional semimeasure $\mu(y|x)$ is defined as in Equation 5.2 above.

For each x, the function $\mu(y|x)$ is a semimeasure on the sample space (or the cylinder)

$$\Gamma_x = \{x\omega : \omega \in \mathcal{B}^\infty\}$$

for $x \in \mathcal{B}^*$. If the unconditional μ is a measure, then so is $\mu(\cdot|x)$. This is easily verified from the definitions. We can interpret $\mu(y|x)$ as the μ-probability that a string starting with x continues with y. Hence, if we have observed that a sequence from a μ-distribution starts with x, then $\mu(y|x)$ allows us to predict whether it will continue with y. This formula solves the problem of extrapolation of sequences once we know μ.

However, in many cases μ is unknown or unknowable. We would like a standard procedure to estimate μ. It turns out that $\mathbf{M}(y|x)$ is a good estimator for *all computable* $\mu(y|x)$.

When we predict the continuation of a sequence over $\mathcal{B}^*$ randomly drawn from a distribution μ, can we estimate the error we make using $\mathbf{M}$ for prediction instead of the actually computable distribution μ?

Example 5.2.3 Let μ be a computable semimeasure on $\mathcal{B}^\infty$, and $\omega \in \mathcal{B}^\infty$. If for all n, we have $\mu(\omega_{1:n}) > 0$, then

$$\frac{\mathbf{M}(\omega_{1:n})}{\mu(\omega_{1:n})} = \prod_{i=1}^{n} \frac{\mathbf{M}(\omega_i|\omega_{1:i-1})}{\mu(\omega_i|\omega_{1:i-1})} \geq 2^{-K(\mu)}.$$

This expression is the product of the ratios of the conditional probabilities for the successive symbols of ω. The geometric mean of the factors of this n-fold product is

$$r_n = \left(\prod_{i=1}^{n} \frac{\mathbf{M}(\omega_i|\omega_{1:i-1})}{\mu(\omega_i|\omega_{1:i-1})} \right)^{1/n}. \tag{5.3}$$

By Corollary 4.5.5 on page 320, for each μ-random infinite ω there is a constant c such that

$$\sup_{n \in \mathcal{N}} \left\{ \frac{\mathbf{M}(\omega_{1:n})}{\mu(\omega_{1:n})} \right\} \leq c.$$

Thus, for each element of a subset of $\mathcal{B}^\infty$ of μ-measure one (the μ-random ω's), we can bound r_n from both sides:

$$2^{-K(\mu)/n} \leq r_n \leq c^{1/n}.$$

That is, for μ-random ω's the associated r_n goes to 1 for $n \to \infty$. But, one may ask, would it not be possible that some factors in Equation 5.3

are much greater than 1 while other factors are much less than 1? Is it possible that the geometric mean r_n goes to 1, but the ratio

$$\frac{\mathbf{M}(\omega_{n+1}|\omega_{1:n})}{\mu(\omega_{n+1}|\omega_{1:n})}$$

fluctuates forever? The amplitude of the fluctuations might even increase? This could mean that predictions according to $\mathbf{M}$ could be far off the mark from predictions according to μ infinitely often. However, Theorem 5.2.1 will show that if there are fluctuations of this sort, then they must damp out quickly. ◇

Convergence in Difference

Assume that the one-way infinite sequences in $S = \mathcal{B}^\infty$ are distributed according to a computable measure μ. If we want to predict the next element $a \in \mathcal{B}$ that will appear after an initial sequence $x \in \mathcal{B}^*$, then we can do no better than predicting according to $\mu(a|x)$. If instead we predict according to another distribution ρ, then the prediction error can be measured as the difference of the probabilities involved:

$$|\rho(a|x) - \mu(a|x)|\,.$$

Solomonoff, Exercise 5.2.2 on page 366, looked at the squared difference rather than the absolute difference. However, the squared difference of the square roots,

$$\left(\sqrt{\rho(a|x)} - \sqrt{\mu(a|x)}\right)^2,$$

has more appealing technical properties for us, and its use makes no nontrivial difference with respect to the results. Instead of looking at the error in the nth prediction after a *particular* initial segment of length $n-1$ we consider the μ-average of the nth prediction error over all initial segments of length $n-1$. Of course, we are interested in the prediction error we make if ρ is set to $\mathbf{M}$.

Let us introduce some notation. Suppose that P is a discrete probability measure and Q is a discrete semimeasure, both over a discrete set $\mathcal{B}$; see Definition 4.3.1 on page 265. Define

$$D\,(P \parallel Q) = \sum_{a\in\mathcal{B}} P(a) \ln \frac{P(a)}{Q(a)}, \tag{5.4}$$

where ln is the natural logarithm and we define $0 \ln 0 = 0 \ln 0/0 = 0$. This quantity is called the *Kullback–Leibler divergence* (or distance) of Q with respect to P. Viewed as a function of P and Q it is asymmetric. We have extended the standard definition, where both P and Q are probabilities summing to 1. To see that this makes no difference, consider

proper probabilities P' and Q' over $\mathcal{B} \bigcup \{u\}$, where u is a new element, defined by $P'(a) = P(a)$ and $Q'(a) = Q(a)$ for every $a \in \mathcal{B}$, while $P'(u) = 0$ and $Q'(u) = 1 - \sum_{a \in \mathcal{B}} Q(a)$. Then, $D(P' \parallel Q') = D(P \parallel Q)$. By the concavity of the logarithm, $D(P \parallel Q)$ as in Equation 5.4 is always nonnegative and is 0 only if $Q = P$. It is a measure of the difference between P and Q. The squared *Hellinger distance* between P and Q as above is defined as

$$H(P,Q) = \sum_{a \in \mathcal{B}} \left(\sqrt{P(a)} - \sqrt{Q(a)} \right)^2 .$$

Again, it is always nonnegative and is 0 only if $Q = P$. It is another measure of the difference between P and Q. It is a metric, unlike the Kullback–Leibler distance, but nonetheless it is bounded above by the latter distance.

Lemma 5.2.1 *$H(P,Q) \leq D(P \parallel Q)$ for every discrete probability measure P and discrete semimeasure Q.*

Proof. Let x and y be nonnegative real numbers. Define

$$f(x,y) = x \ln \frac{x}{y} - (\sqrt{x} - \sqrt{y})^2 + y - x.$$

We want to show that $f(x,y) \geq 0$ for all $x, y \geq 0$. Define $\ln 0/0 = 0$. If $x = 0$ then $f(x,y) = 0$ for every $y \geq 0$. Assume $x > 0$. Since $\ln x \leq x - 1$ for every $x \geq 0$, we have $f(x,y)/(2x) = -\ln \sqrt{y/x} + \sqrt{y/x} - 1 \geq 0$ for $\sqrt{y/x} \geq 0$. Altogether, $f(x,y) \geq 0$ for all $x, y \geq 0$, and hence $\sum_{a \in \mathcal{B}} f(P(a), Q(a)) \geq 0$. Consequently,

$$\sum_{a \in \mathcal{B}} P(a) \ln \frac{P(a)}{Q(a)} - \sum_{a \in \mathcal{B}} \left(\sqrt{P(a)} - \sqrt{Q(a)} \right)^2 \geq \sum_{a \in \mathcal{B}} P(a) - \sum_{a \in \mathcal{B}} Q(a) \geq 0.$$

□

Definition 5.2.2 Let $\mathcal{B}$ be a finite alphabet, and let x be a word over $\mathcal{B}$. The *summed expected squared error at the nth prediction* S_n is defined by

$$S_n(a) = \sum_{l(x)=n-1} \mu(x) \left(\sqrt{\mathbf{M}(a|x)} - \sqrt{\mu(a|x)} \right)^2 ,$$

$$S_n = \sum_{a \in \mathcal{B}} S_n(a).$$

Let $K(\mu)$ be the length of the shortest program computing the function μ in a self-delimiting binary programming language.

Theorem 5.2.1 *Let μ be a computable measure. Then, $\sum_n S_n \leq k$ with the constant $k = K(\mu)\ln 2$.*

Proof. Let μ be a computable measure and $\mathbf{M}$ the universal lower semicomputable semimeasure, both over $\mathcal{B}^\infty$. Replacing $P(a)$ and $Q(a)$ in Equation 5.4 by the conditional discrete probabilities $\mu(a|x)$ and $\mathbf{M}(a|x)$, respectively, we define

$$D\left(\mu(\cdot|x) \parallel \mathbf{M}(\cdot|x)\right) = \sum_{a\in\mathcal{B}} \mu(a|x) \ln \frac{\mu(a|x)}{\mathbf{M}(a|x)} ,$$

$$D_n = \sum_{l(x)=n-1} \mu(x) D\left(\mu(\cdot|x) \parallel \mathbf{M}(\cdot|x)\right) .$$

The theorem follows directly from the following two claims.

Claim 5.2.1 *For all n, we have $S_n \leq D_n$.*

Proof. The statement follows directly from Lemma 5.2.1. Substituting $P = \mu(\cdot|x)$ and $Q = \mathbf{M}(\cdot|x)$ we obtain

$$\sum_{a\in\mathcal{B}} \left(\sqrt{\mu(a|x)} - \sqrt{\mathbf{M}(a|x)}\right)^2 \leq \sum_{a\in\mathcal{B}} \mu(a|x) \ln \frac{\mu(a|x)}{\mathbf{M}(a|x)}.$$

Multiplying both sides of the displayed inequality by $\mu(x)$ and summing over all x with $l(x) = n-1$ proves the claim. □

Claim 5.2.2 *$\sum_n D_n \leq K(\mu)\ln 2$, the sum taken from 1 to ∞.*

Proof. In the definition of D_n above, write $x_1 \ldots x_{n-1}$ for x and x_n for a. Then, substitute $\mu(x_n|x_{1:n-1})\mu(x_{1:n-1}) = \mu(x_{1:n})$. This yields, for all $m \geq n$,

$$\begin{aligned} D_n &= \sum_{x_{1:n}} \mu(x_{1:n}) \ln \frac{\mu(x_n|x_{1:n-1})}{\mathbf{M}(x_n|x_{1:n-1})} \\ &= \sum_{x_{1:m}} \mu(x_{1:m}) \ln \frac{\mu(x_n|x_{1:n-1})}{\mathbf{M}(x_n|x_{1:n-1})}, \end{aligned}$$

where the last equality follows since $\sum_{x_{n+1:m}} \mu(x_{1:m}) = \mu(x_{1:n})$, and the expression in the logarithm is independent of $x_{n+1:m}$. Subsequently, we rewrite $\sum_{n=1}^m D_n$ by moving the $\sum_{n=1}^m$ into the logarithmic term as $\prod_{n=1}^m$:

$$\begin{aligned} \sum_{n=1}^m D_n &= \sum_{x_{1:m}} \mu(x_{1:m}) \ln \prod_{n=1}^m \frac{\mu(x_n|x_{1:n-1})}{\mathbf{M}(x_n|x_{1:n-1})} \\ &= \sum_{x_{1:m}} \mu(x_{1:m}) \ln \frac{\mu(x_{1:m})}{\mathbf{M}(x_{1:m})} \leq K(\mu)\ln 2. \end{aligned}$$

The second equality uses the measure property $\mu(x_{1:m}) = \mu(x_1)\mu(x_2|x_1) \cdots \mu(x_m|x_{1:m-1})$, and the semimeasure property $\mathbf{M}(x_{1:m}) = \mathbf{M}(x_1)\mathbf{M}(x_2|x_1)\cdots\mathbf{M}(x_m|x_{1:m-1})$. The last inequality uses the universality of $\mathbf{M}$, Corollary 4.5.1 on page 302, which states that $M(x_{1:m}) \geq 2^{-K(\mu)}\mu(x_{1:m})$. Since the last displayed inequality holds for every m, the finite upper bound on the right-hand side also holds for $m \to \infty$. □ □

Example 5.2.4 Since the series $\sum_n S_n$ converges, we have $\lim_{n\to\infty} S_n = 0$ and in fact, the entire tail sum goes to zero: $\lim_{n\to\infty} \sum_{i=n}^{\infty} S_i = 0$. Compare this with the harmonic series $\sum_n 1/n$, where the individual terms $1/n$ go to 0 as n goes to ∞, but all tail sums diverge: $\lim_{n\to\infty} \sum_{i=n}^{\infty} 1/n = \infty$ for every n. But the expected prediction error S_n in the nth prediction may not go to 0 faster than $1/n$, since S_n may not be monotonic nonincreasing (contrary to that, one expects to predict better after having seen more data). Consider situations such as $S_n = 1/\sqrt{n}$ for $n = i^3$ and $S_n = 1/n^2$ for $n \neq i^3$ ($i \in \mathcal{N}$).

Another, natural, example is that of μ a deterministic measure over $\{0,1\}^\infty$, for instance, $\mu(11\ldots1) = 1$ for all strings consisting of only 1's. Then, $S_n(0) = \mathbf{M}(0|1^{n-1}) = 2^{-K(n)+O(\log K(n))}$, while $S_n(1) = (\sqrt{\mathbf{M}(1|1^{n-1})}-1)^2$, which goes to 0 as $n \to \infty$. Hence for large complex n ($K(n) > l(n)$) we obtain $S_n < S_{n'}$ for larger but regular n' ($K(n') \ll l(n')$) satisfying $n < n' < 2n$. This effect is of course much more pronounced for expected prediction error $S_n(0)$, the error in predicting 0, where we can have $S_n(0) = O((\log n)^{O(1)}/n)$ and $S_{n'}(0) = \Omega(1/\log n)$ with n' satisfying $n < n' < 2n$. ◇

A more precise determination of $S_n(0) = \mathbf{M}(0|1^{n-1}) = 2^{-K(n)+O(1)} = O(1/n)$ is given in [M. Hutter, *Theoret. Comput. Sci.*, 1:384(2007), 33–48], Equation (17) and Appendix A.

Lemma 5.2.2 *Let μ be a measure on $\mathcal{B}^\infty$. For every natural number n, and every function $f_n : \mathcal{B}^n \to \mathcal{R}$, let $F_n = \sum_{l(x)=n} \mu(x) f_n^2(x)$. Then, $\sum_n F_n < \infty$ implies that for some $A \subseteq \mathcal{B}^\infty$ of μ-measure one, for every $\omega = \omega_1\omega_2\ldots$ in A, we have $f_n(\omega_{1:n}) \to 0$ for $n \to \infty$.*

Proof. Let m be a natural number, ϵ a nonnegative rational number, and f_n and F_n as in the statement of the lemma. Let $B_{m,\epsilon} = \{\omega : \text{there is an } n \geq m \text{ such that } f_n(\omega_{1:n}) \geq \sqrt{\epsilon}\}$. Here we use the traditional notation $\mu(A)$ with $A \subseteq \mathcal{B}^\infty$ instead of the simplification $\mu(x)$ for $\mu(\Gamma_x)$. Then,

$$\mu(B_{m,\epsilon}) = \mu\left(\bigcup_{n\geq m} \{\omega : f_n(\omega_{1:n})^2 \geq \epsilon\}\right) \leq \sum_{n=m}^{\infty} \mu\left(\{\omega : f_n(\omega_{1:n})^2 \geq \epsilon\}\right)$$
$$\leq \sum_{n=m}^{\infty} \sum_{\omega_{1:n}} \mu(\omega_{1:n}) \frac{f_n^2(\omega_{1:n})}{\epsilon} = \frac{1}{\epsilon} \sum_{n=m}^{\infty} F_n,$$

where we used $\mu(C \bigcup D) \leq \mu(C) + \mu(D)$ and Markov's inequality of Example 4.3.10 on page 282. By assumption $\sum_{n=1}^{\infty} F_n < \infty$. Therefore,

$$\lim_{m\to\infty} \frac{1}{\epsilon} \sum_{n=m}^{\infty} F_n = 0.$$

Define $B_\epsilon = \lim_{m\to\infty} B_{m,\epsilon}$. The above two displayed equations show that $\mu(B_\epsilon) = 0$. Setting $A = \mathcal{B}^\infty - \bigcup_\epsilon B_\epsilon$, where the union is taken over all nonnegative rational ϵ, of which there are only countably many, the lemma follows. □

Corollary 5.2.1 Let μ be a computable measure. There is a set $A \subseteq \mathcal{B}^\infty$, of μ-measure one, such that for every $\omega \in A$, for every $a \in \mathcal{B}$,

$$\mathbf{M}(a|\omega_{1:n-1}) \to \mu(a|\omega_{1:n-1}),$$

with $\omega = \omega_1\omega_2\ldots$ and $n \to \infty$. (Here $a_n \to b_n$ means $a_n - b_n \to 0$ without implying that $\lim_{n\to\infty} b_n$ exists.)

Proof. Use Theorem 5.2.1 and substitute

$$f^2_{n-1}(\omega_{1:n-1}) = \sum_{a\in\mathcal{B}} \left(\sqrt{\mathbf{M}(a|\omega_{1:n-1})} - \sqrt{\mu(a|\omega_{1:n-1})}\right)^2$$

and $F_{n-1} = S_{n-1}$ in Lemma 5.2.2. □

Example 5.2.5 The proof of Theorem 5.2.1 does not depend on any computability properties, but only on the dominating property of $\mathbf{M}$ over μ. For example, take in Theorem 5.2.1 any (possibly incomputable) measure μ, and a semimeasure ρ over $\mathcal{B}^\infty$ that dominates μ in the sense that there exists a function f such that

$$\rho(x) \geq 2^{-f(n)}\mu(x),$$

for all $x \in \mathcal{B}^*$ of length $l(x) = n$. We let ρ take the role of $\mathbf{M}$, and define $S_i = \sum_{l(x)=i-1} \sum_{a\in\mathcal{B}} \mu(x) \left(\sqrt{\rho(a|x)} - \sqrt{\mu(a|x)}\right)^2$. The same proof now shows that $\sum_{i=1}^{n} S_i \leq f(n) \ln 2$.

We can apply this to Laplace's law of succession (Exercise 1.10.6 on page 65) which states the following. For binary Bernoulli processes $(p, 1-p)$ with unknown p $(0 < p < 1)$, that is, independent flips of a coin with unknown bias p, the best prediction of an outcome 1 following a sequence x of n outcomes containing k outcomes 1 is given by $\rho(1|x) = (k+1)/(n+2)$. To determine the unconditional *Laplace's measure* ρ,

observe that it depends only on the number of 1s and 0s in the argument, but not on their order. This leads to

$$\rho(x) = \frac{k!(n-k)!}{(n+1)!}.$$

Let μ be a measure generated by independent and identically distributed outcomes of a given Bernoulli process $(p, 1-p)$. Then,

$$\begin{aligned}\frac{\mu(x)}{\rho(x)} &= \frac{(n+1)!}{k!(n-k)!}p^k(1-p)^{n-k} \\ &= (n+1)\binom{n}{k}p^k(1-p)^k \leq n+1.\end{aligned}$$

Thus, $\rho(x) \geq 1/(n+1)\mu(x)$. That is, $f(n) = \log(n+1)$. Hence, the summed expected squared error in the first n predictions, Definition 5.2.2 on page 353, using Laplace's measure ρ instead of the real measure μ associated with the given Bernoulli process, is given by

$$\sum_{i=1}^{n} S_i \leq (\ln 2)\log(n+1) = \ln(n+1).$$

◇

Example 5.2.6 We continue Example 5.2.1 with the measure μ defined there, and with $\omega = \omega_1\omega_2\ldots$ being an infinite binary sequence. By Corollary 5.2.1, we find that

$$\lim_{n\to\infty} \mathbf{M}(a|\omega_{1:n}) = \begin{cases} \frac{1}{10} & \text{for } n = 2i \text{ and } a = 0, 1, \ldots, 9, \\ 1 & \text{for } n = 2i+1 \text{ and } a \text{ the } i\text{th digit of } \pi, \\ 0 & \text{for } n = 2i+1 \text{ and } a \text{ not the } i\text{th digit of } \pi \end{cases}$$

That is, the prediction probabilities $\mathbf{M}$ conditioned on growing length initial segments become eventually precisely the conditional probabilities of the true measure μ that by definition give the best possible predictions. ◇

According to [R.J. Solomonoff, *IEEE Trans. Inform. Theory*, 24(1978), 422–432], "Ordinary statistical analysis of a Bernoulli sequence gives an expected squared error for the probability of the nth symbol proportional to $1/n$, and a total squared error [expected] for the first n symbols proportional to $\ln n$." This is clearly much larger than the constant $K(\mu)\ln 2$ we found in Theorem 5.2.1 or the total squared error of $K(\mu)2\ln 2$ we find in Exercise 5.2.2 on page 366. This discrepancy can be understood by noting that the theorem requires μ to be computable. The set of computable measures is countable. But the parameter defining a Bernoulli process can be any real number between 0 and 1. The set of these measures is uncountable. It has been shown that if one restricts the possible hypotheses to countably many, then the statistical

error converges much more rapidly than if one considers uncountably many hypotheses [T.M. Cover, *Ann. Stat.*, 1:5(1973), 862–871]. However, these statements may be misleading for real situations in which the data have definite finite size. Namely, consider prediction of sequences generated by a Bernoulli process with parameter p. Is it true that one will be able to predict better on a limited initial segment, using Solomonoff's method, if one knows that p is rational? No, unless one also knows that it is a *simple* rational, since the error has a multiplicative factor proportional to $K(p)$. Therefore, for a finite segment of length n (which is what really counts), the statistical method taking into account all p's is inferior to the Solomonoff method only if $K(p) < \frac{1}{2} \ln n$. But even if p is very complex or incomputable, then still Solomonoff's procedure is not worse than the standard statistical ones; see [M. Hutter, *Theoret. Comput. Sci.*, 384:1(2007), 33–48]. This topic has spawned an elaborate theory of prediction based on universal distributions to arbitrary loss measures (rather than just the logarithmic loss) using extensions and variations of the proof method; M. Hutter, *Universal Artificial Intelligence: Sequential Decisions Based on Algorithmic Probability*, Springer-Verlag, Berlin, 2005. Assume that a probabilistic theory concerning some phenomenon is expressible as a computable measure μ on $\{0,1\}^\infty$. We can view the universal semimeasure $\mathbf{M}$ as a mixture of hypotheses, including all computable measures, with greater weights for the simpler ones. Then *Solomonoff's inductive formula* $\mathbf{M}(y|x)$, to estimate the actual probabilities $\mu(y|x)$ to predict outcomes y given a sequence of observed outcomes x, can be viewed as a mathematical form of Occam's razor:

> find all rules fitting the data and then predict y according to the universal distribution on them.

Formalization of this principle for probabilistic theories μ encounters difficulties because of a tradeoff between the complexity $K(\mu)$ and the randomness deficiency $\log \mathbf{M}(x)/\mu(x)$. The current approach is accurate for large x and simple μ.

Convergence in Ratio

Theorem 5.2.1 on page 354, and Corollary 5.2.1 on page 356, formulate the fact that the conditional universal measure converges to every conditional computable measure in difference. This leaves open the convergence in ratio, since the difference can become small without the ratio going to one, for instance, when both items in the difference go to zero at a different rate. Apart from this, there is the distinction between the more restricted notion of on-sequence convergence and the general notion of off-sequence convergence, which we illustrate as follows.

Example 5.2.7 Consider two infinite sequences $\omega = \omega_1\omega_2\ldots$ and $\zeta = \zeta_1\zeta_2\ldots$. Then, Theorem 5.2.1 shows that for every $\omega \in \mathcal{B}^\infty$, except for possibly a set of μ-measure zero, and for every $\zeta \in \mathcal{B}^\infty$, possibly constant and not necessarily random, we have

$$\mathbf{M}(\zeta_n|\omega_{1:n-1}) \to \mu(\zeta_n|\omega_{1:n-1})$$

for $n \to \infty$. This is called *off-sequence* convergence. In contrast, we can make no such assertion about the off-sequence convergence of the ratio

$$\frac{\mathbf{M}(\zeta_n|\omega_{1:n-1})}{\mu(\zeta_n|\omega_{1:n-1})}$$

for $n \to \infty$. This is because possibly $\liminf_{n\to\infty} \mu(\zeta_n|\omega_{1:n-1}) = 0$. In Exercise 5.2.3 on page 367 we give a quantitative example for which this ratio diverges. But taking $\zeta = \omega$, we are considering *on-sequence* convergence of the ratio. The distinction between the conditions under which on-sequence and off-sequence convergence of the ratio holds is formulated in Theorem 5.2.2. ◇

Definition 5.2.3 Let μ be a measure over $\mathcal{B}^\infty$. It is called *conditionally bounded away from zero* if there exists a constant $c > 0$ such that $\mu(a|x) > c$ for all $x \in \mathcal{B}^*$ and $a \in \mathcal{B}$.

Theorem 5.2.2 *Let μ be a computable measure over $\mathcal{B}^\infty$.*

(i) *On-sequence convergence: There is a set of μ-measure one in $\mathcal{B}^\infty$ such that for every $\omega = \omega_1\omega_2\ldots$ in the set, if $n \to \infty$ then*

$$\frac{\mathbf{M}(\omega_n|\omega_{1:n-1})}{\mu(\omega_n|\omega_{1:n-1})} \to 1.$$

(ii) *Off-sequence convergence: Let μ be conditionally bounded away from zero. Then, for every fixed m and $x \in \mathcal{B}^m$, there is a set of μ-measure one in $\mathcal{B}^\infty$ such that for every $\omega = \omega_1\omega_2\ldots$ in the set, if $n \to \infty$ then*

$$\frac{\mathbf{M}(x|\omega_{1:n-1})}{\mu(x|\omega_{1:n-1})} \to 1.$$

Clearly, off-sequence convergence implies on-sequence convergence. The strong property of off-sequence convergence of the ratio holds for a subclass of the set of measures for which the weaker property of on-sequence convergence holds.

Proof. (i) Set F_n in Lemma 5.2.2 equal to S_n in Definition 5.2.2 on page 353. Then,

$$f_n(\omega_{1:n}) = \sqrt{\mathbf{M}(\omega_n|\omega_{1:n-1})/\mu(\omega_n|\omega_{1:n-1})} - 1,$$
$$F_n = \sum_{\omega_{1:n-1}} \mu(\omega_{1:n-1}) \sum_{\omega_n} \left(\sqrt{\mathbf{M}(\omega_n|\omega_{1:n-1})} - \sqrt{\mu(\omega_n|\omega_{1:n-1})}\right)^2.$$

By Theorem 5.2.1, we have $\sum_n F_n < \infty$, and applying Lemma 5.2.2 we find that $\lim_{n\to\infty} f_n(\omega_{1:n}) > 0$ for a set of ω's of μ-measure zero. Additionally, $f_n(x)$ is not defined for x with $\mu(x) = 0$. Recall that $\mu(x)$ is shorthand for $\mu(\{\omega : \omega = x\ldots\})$. Define $Z = \{x : \mu(x) = 0\}$ and

$\mathcal{Z} = \{\omega : \omega = x\ldots,\ x \in Z\}$. Since the set of x's such that $\mu(x) = 0$ is countable, we have $\mu(\mathcal{Z}) = 0$. Hence the set of ω's such that $f_n(\omega_{1:n})$ is either not defined for some n, or $\lim_{n\to\infty} f_n(\omega_{1:n}) > 0$, has μ-measure zero. Since f_n is either nonnegative or undefined, the set of ω's such that $\lim_{n\to\infty} f_n(\omega_{1:n}) = 0$ has μ-measure one.

(ii) Let $m \geq 1$ and set $l = n+m-1$. The m-step analogues of the 1-step S_n and D_n in Theorem 5.2.1 are

$$S_{n,m} = \sum_{\omega_{1:l}} \mu(\omega_{1:n-1}) \left(\sqrt{\mathbf{M}(\omega_{n:l}|\omega_{1:n-1})} - \sqrt{\mu(\omega_{n:l}|\omega_{1:n-1})}\right)^2,$$

$$D_{n,m} = \sum_{\omega_{1:l}} \mu(\omega_{1:l}) \ln \frac{\mu(\omega_{n:l}|\omega_{1:n-1})}{\mathbf{M}(\omega_{n:l}|\omega_{1:n-1})}.$$

Thus, $S_{n,1} = S_n$ and $D_{n,1} = D_n$. Note the different scope $\mu(\omega_{1:n-1})$ versus $\mu(\omega_{1:l})$. Similarly to the proof of Claim 5.2.2 on page 354 one shows that $D_{n,m} = \sum_{t=n}^{l} D_t$, and similar to the proof of Claim 5.2.1 on page 354 one can prove $S_{n,m} \leq D_{n,m}$. Therefore,

$$\sum_{n=1}^{\infty} S_{n,m} \leq \sum_{n=1}^{\infty}\sum_{t=n}^{l} D_t \leq m\sum_{n=1}^{\infty} D_n < mK(\mu)\ln 2 < \infty.$$

We have assumed that there is a constant $c > 0$ such that $\mu(a|x) > c$ for all $x \in \mathcal{B}^*$ and $a \in \mathcal{B}$. Therefore, with $l(x) = m$, the measure $\mu(x|\omega_{1:n-1})$ is greater than c^m. We apply Lemma 5.2.2 with m fixed. With $F_n = S_{n,m}$, the corresponding $f_n(\omega_{1:n})$ can be written as

$$\left|\sqrt{\frac{\mathbf{M}(x|\omega_{1:n-1})}{\mu(x|\omega_{1:n-1})}} - 1\right| = \frac{\left|\sqrt{\mathbf{M}(x|\omega_{1:n-1})} - \sqrt{\mu(x|\omega_{1:n-1})}\right|}{\sqrt{\mu(x|\omega_{1:n-1})}}$$
$$< c^{-m/2}\left|\sqrt{\mathbf{M}(x|\omega_{1:n-1})} - \sqrt{\mu(x|\omega_{1:n-1})}\right|.$$

Since $\sum_n F_n < \infty$, the right-hand side goes to 0 as $n \to \infty$ for a set of ω's of μ-measure one. □

Theorem 5.2.2 in this edition differs from the corresponding theorem in the previous editions of this work; see Section 5.6 on page 433.

Theorem 5.2.2 shows that the conditional prior probability $\mathbf{M}(y|x)$ suffices to approximate the conditional probability $\mu(y|x)$, Equation 5.2 on page 350, and convergence is very fast by Theorem 5.2.1. Thus, if we use the fixed distribution $\mathbf{M}$ as prior in Bayes's rule, then this single inferred probability converges very fast to every inferred probability using the actual prior μ, provided the latter is computable. The problem with

Bayes's rule has always been the determination of the prior. Using $\mathbf{M}$ universally gets rid of that problem and is provably perfect.

5.2.4 Prediction by Data Compression

Above it is shown that the universal distribution *itself* is directly suited for prediction. The universal distribution combines a weighted version of the predictions of all lower semicomputable semimeasures, including the prediction of the semimeasure with the shortest program. It is not a priori clear that the shortest program dominates in all cases—and in fact it does not. However, we show that in the overwhelming majority of cases the shortest program dominates sufficiently to validate the approach that uses only shortest programs for prediction.

Given a semimeasure μ on $\mathcal{B}^\infty$ and an initial string x, our goal is to find the most probable extrapolation of x. That is, taking the negative logarithm on both sides of Equation 5.2, we want to determine y with $l(y) = n$ that minimizes

$$\log \frac{1}{\mu(y|x)} = \log \frac{1}{\mu(xy)} - \log \frac{1}{\mu(x)}.$$

Theorem 5.2.3 *Let μ be a semimeasure as in Theorem 5.2.2, Item (ii), on page 359. There is a subset $A \subseteq \{0,1\}^\infty$ of μ-measure one such that for every $\omega \in A$, with x denoting a finite prefix of ω that grows unboundedly, and y a string of fixed length (not necessarily $\omega = xy\ldots$), we have*

$$\lim_{l(x)\to\infty} \log \frac{1}{\mu(y|x)} = Km(xy) - Km(x) + O(1) < \infty.$$

Proof. For $l(x)$ that grows unboundedly with $l(y)$ fixed, we have by Theorem 5.2.2

$$\lim_{l(x)\to\infty} \left(\log \frac{1}{\mathbf{M}(y|x)} - \log \frac{1}{\mu(y|x)} \right) = 0, \tag{5.5}$$

with μ-probability one. Therefore, if x and y satisfy the above conditions, then minimizing $\log 1/\mathbf{M}(y|x)$ over y implies with μ-probability 1 that $\mu(y|x)$ is maximized over y. It is shown in Lemma 4.5.6 on page 308 that $\log 1/\mathbf{M}(x)$ can be slightly smaller than $Km(x)$, the length of the shortest program for x on the reference universal monotonic machine. For binary programs this difference is very small (Equation 4.15 on page 307) but possibly unbounded in the length of x. That is,

$$\begin{aligned} \log \frac{1}{\mathbf{M}(y|x)} &= \log \frac{1}{\mathbf{M}(xy)} - \log \frac{1}{\mathbf{M}(x)} \\ &= (Km(xy) - g(xy)) - (Km(x) - g(x)), \end{aligned}$$

where $g(x)$ is a function that can rise to a value between the inverse of the Ackermann function and $Km(l(x)) \leq \log\log x$—but only in case x

is not μ-random. Therefore, for certain x and xy, optimization using the minimum-length programs may result in incorrect predictions. However, for μ-random x we have that $\log 1/\mathbf{M}(x)$ and $Km(x)$ coincide up to an additional constant independent of x, that is, $g(xy) = O(1)$ and $g(x) = O(1)$, by Lemma 4.5.6. Since both the set of μ-random sequences and the set of sequences satisfying Equation 5.5 have μ-measure one, their intersection, denoted by A, has μ-measure one as well. □

By its definition Km is monotone in the sense that always $Km(xy) - Km(x) \geq 0$. If y makes this difference equal to 0, then the shortest effective monotone program for x is also a shortest effective monotone program for xy and hence predicts y given x. For all large enough μ-random x, predicting by determining y that minimizes the difference of the minimum program lengths for xy and x gives a good prediction. Here y should be preferably large enough to eliminate the influence of the $O(1)$ term.

Corollary 5.2.2 **(Prediction by data compression)** Assume the conditions of Theorem 5.2.3. There is a constant $p > 0$ such that with μ-probability going to one as $l(x)$ grows unboundedly, an extrapolation y from x has probability $\mu(y|x) \geq p/2^c$ if y can be compressed with respect to x in the sense that $Km(xy) - Km(x) \leq c$. That is, it is a good heuristic to choose the extrapolation y that minimizes the length difference between the shortest program that outputs $xy\ldots$ and the shortest program that outputs $x\ldots$.

Example 5.2.8 Assume for convenience that $\mathcal{B} = \{0, 1\}$. If we consider extrapolations of length l, there are 2^l possibilities. Considering large l, we say that y compresses well with respect to x in case $c \leq \log l$. Then, the corollary tells us that $\mu(y|x) \geq p/l$. There can be only l/p strings z of length l with that high a probability. At least $2^l - l/p$ possible extrapolations of length l have lower $\mu(\cdot|x)$-probability. This is a fraction of the total number of l-length extrapolations that goes to 1 as l grows unboundedly. Stated differently, choosing a long extrapolation that compresses well with respect to x causes us to pick an extrapolation that belongs to the vanishing fraction of such extrapolations that have the highest $\mu(\cdot|x)$-probabilities, even though it may not have the very highest probability. ◇

That the compression criterion will not always give us the highest $\mu(\cdot|x)$-probability can be seen as follows. Consider a measure μ with $\mu(0|x) = 2/3$ and choose a reference universal monotone machine with $Km(x0) - Km(x) = 0$ for infinitely many $l(x)$ with probability 1. Minimizing $-\log \mu(y|x)$ gives $y = 0$, and minimizing $Km(xy) - Km(x)$ gives $y = 1$. A detailed study of prediction by compression is [M. Hutter, *J. Comput. Syst. Sci.*, 72(2006), 95–117], where many more such examples are given.

5.2.5 Universal Recursion Induction

In *inductive inference*, a particular abstraction of the induction question associated with the name of E.M. Gold, we are given an effective enumeration of partial recursive functions $f_1, f_2, \ldots$ mapping domain X into range Y. Such an enumeration can be the functions computed by Turing machines, but also the functions computed by finite automata. We want to infer a particular function f. To do so, we are presented with a sequence of examples, $D = e_1, e_2, \ldots, e_n$, containing elements (possibly with repetitions) of the form

$$e = \begin{cases} (x, y, 0) & \text{if } f(x) \neq y, \\ (x, y, 1) & \text{if } f(x) = y, \end{cases}$$

with $x \in X$ and $y \in Y$. For $n \to \infty$ we assume that the resulting D contains all elements $(x, y, l_{x,y})$, $(x, y) \in X \times Y$ and $l_{x,y} = 1$ if $f(x) = y$ and 0 otherwise.

Hypothesis Identification

Let the different hypotheses H_k be '$f = f_k$.' Then, $\sum_j \{\Pr(D|H_j) : D \text{ is consistent with } f_j\}$=1, and $\sum_j \{\Pr(D|H_j) : D \text{ is inconsistent with } f_k\} = 0$. Take any positive prior distribution $P(H_k)$, say $P(H_k) = 1/k(k+1)$, and apply Bayes's rule, Equation 5.1, to obtain

$$\Pr(H_k|D) = \frac{\Pr(D|H_k)P(H_k)}{\sum_j \Pr(D|H_j)P(H_j)}, \tag{5.6}$$

with the summation over those j such that f_j is consistent with D. With increasing n, the numerator is monotonically nonincreasing. Since all examples eventually appear, the numerator and denominator converge to limits. Of course, since we deal with partial computable functions, we cannot effectively decide whether data D is consistent with those functions that are not computable. That question is ignored here.

For each k, the inferred probability of f_k is monotonically nondecreasing with increasing n, until f_k is inconsistent with a new example, in which case it falls to zero and stays there henceforth. Only the f_k's that are consistent with the sequence of presented examples have positive inferred probability. Order the f_k's by decreasing probability. The ordering between the f_k's with positive probability never changes; the only thing that can happen is that f_k's are removed from this set because their probability drops to zero. At each step we infer the f_k with the highest posterior probability. This is always the first element in the current ordering. Assuming that f occurs in the list, at some step all preceding functions before the first occurrence of f have been deleted. From that step onward, f has the highest probability.

Analyzing this process, we find that it is equivalent to starting with the f_k's ordered by decreasing probability according to $P(H_k) = 1/k(k+1)$,

as the list $f_1, f_2, \ldots$. Similar to the classical Gold method, after receiving each new example we eliminate all remaining f_k's that are inconsistent from the beginning onward up to the position of the first consistent function. When we receive a new example e, set $D := D, e$, and repeat this process. Eventually, the remaining first function in the enumeration is a copy of f and it does not change any more. This algorithm is called *learning by enumeration*. One learns more and more about the unknown target function and approximates it until the correct identification has been achieved. This learning model is called *learning in the limit*. The learner eventually learns the concept exactly but never knows when that has happened. This deceptively simple idea has generated a large body of sophisticated literature.

How do we use the universal prior probability in this setting? Choose $P(H_k) = \mathbf{m}(k)$, with $\mathbf{m}(\cdot)$ the universal discrete probability. By Theorem 4.3.1, page 267, we have $\mathbf{m}(x) = 2^{-K(x)}$, with $K(\cdot)$ the prefix complexity. With this prior, at each stage, $\Pr(\cdot|D)$ will be largest for the simplest consistent hypothesis, that is, for the one with the least prefix complexity. In the limit, this will be the case for H_k such that $f_k = f$ with $K(k)$ minimal. This process corresponds to enumeration of the f_k's as $f_{\pi(1)}, f_{\pi(2)}, \ldots$, where π is a permutation such that $K(\pi(i)) \leq K(\pi(i+1))$ for all $i = 1, 2, \ldots$.

In this way, one enumerates the functions by increasing prefix complexity. Assume that the looked-for $f = f_k$ is a simple function, as it in practice always is, with $K(k) \ll \log k$. (If f is a truly random function then it is highly unlikely that anyone will ever conceive of its existence and would want to learn it.) We will find simple f_k much faster using $\mathbf{m}(\cdot)$ as prior than using $1/k(k+1)$; see Exercise 5.2.11 on page 369. In fact, the speed-up can be incomputably fast! But since $\mathbf{m}$ is incomputable, one can use $1/k(k+1)$ as a (trivially computable) approximation.

Mistake Bounds

Consider an effective enumeration $f_1, f_2, \ldots$ of partial recursive functions with values in the set $\{0, 1\}$ only. Each function f in this enumeration that happens to be total defines an infinite binary sequence $\omega = \omega_1\omega_2\ldots$ by $\omega_i = f(i)$, for all i. In this way, we have an enumeration of infinite sequences ω. These sequences form a binary tree with the root labeled ϵ, and each ω is an infinite path starting from the root. We are trying to learn a particular function f, in the form that we predict ω_i from the initial sequence $\omega_1 \ldots \omega_{i-1}$ for all $i \geq 1$. We want to analyze the number of errors, called *mistakes*, we make in this process. If our prediction is wrong (say, we predict a 0 and it should have been a 1), then this counts as one mistake.

Lemma 5.2.3 *Assume the discussion above. We try to infer $f = f_n$. There is an algorithm that makes fewer than $2\log n$ mistakes in all infinitely many predictions.*

Proof. Define, for each f_i with associated infinite sequence ω^i, a measure μ_i by $\mu_i(\omega^i) = 1$. This implies that also $\mu_i(\omega_1^i \dots \omega_n^i) = 1$ for all n. Let ξ be a so-called mixture semimeasure defined by

$$\xi(x) = \sum_i \frac{1}{i(i+1)} \mu_i(x),$$

for each $x \in \{0,1\}^*$. (Note that ξ is a simple computable approximation to $\mathbf{M}$.) The prediction algorithm is very simple:

$$\text{If } \xi(0|x) \geq \tfrac{1}{2}, \text{ then predict 0; otherwise predict 1.}$$

Suppose that the target f is f_n. If there are k mistakes, then the conditional in the algorithm shows that $2^{-k} > \xi(\omega^n)$. (The combined probability of the mistakes is largest if they are concentrated in the first predictions.) By the definition of ξ we have $\xi(\omega^n) \geq 1/(n(n+1))$. Together this shows that $k < 2\log(n+1)$. □

Example 5.2.9 If in the proof we put weight $2^{-K(n)}$ on μ_n (instead of weight $1/(n(n+1))$), then the number of mistakes is at most $k < K(n)$. Recall that always $K(n) \leq \log n + 2\log\log n + O(1)$. But for certain simple n (such as $n = 2^k$) the value $K(n)$ drops to less than $\log\log n + 2\log\log\log n + O(1)$. Of course, the prediction algorithm becomes ineffective because we cannot compute these weights. ◇

Lemma 5.2.4 *If the target function is f and we make k errors in the first m predictions, then $\log\binom{m}{k} + K(m,k) + O(1) \geq K(f(1)\dots f(m))$.*

Proof. Let A be a prediction algorithm. If k is the number of errors, then we can represent the mistakes by the index j in the ensemble of k mistakes out of m, where

$$j \leq \binom{m}{k}.$$

If we are given A, m, k, and j, we can reconstruct $f(1)\dots f(m)$. Therefore, $K(A,m,k,j) \geq K(f(1)\dots f(m))$. Since $K(A) = O(1)$, the lemma is proven. □

Certification

The following lemma sets limits on the number of examples needed to effectively infer a particular function f. In fact, it does more. It sets a

limit to the number of examples we need to *describe* or *certify* a particular function f in any effective way. Let $D = e_1 e_2 \ldots e_n$ be a sequence of examples $e_i = (x_i, y_i, b_i)$ and let $x = x_1 x_2 \ldots x_n$, $y = y_1 y_2 \ldots y_n$, and $b = b_1 b_2 \ldots b_n$. The statement of the lemma must cope with pathological cases such as that x simply spells out f in some programming language.

Lemma 5.2.5 *Let c be a large enough constant. If $K(f|x,y) > K(b|x,y) + c$, then we cannot effectively find f.*

Proof. Assume, by way of contradiction, that we are able to compute f, given D, from a program of length significantly shorter than $K(f|D)$. By Theorem 3.9.1, page 247, with extra conditional x, y in all terms, we have

$$K(f, b|x, y) \leq K(b|x, y) + K(f|b, x, y) + O(1). \tag{5.7}$$

We have assumed that there is an algorithm A that given D, returns f. That is, describing A in $K(A) = O(1)$ bits, we obtain

$$K(f|b, x, y) = K(f|D) + O(1) \leq K(A) + O(1) = O(1).$$

Substituting this in Equation 5.7, we obtain $K(f, b|x, y) \leq K(b|x, y) + O(1)$. Since, trivially, $K(f, b|x, y) = K(f|x, y) + O(1)$, we obtain $K(f|x, y) \leq K(b|x, y) + O(1)$, which contradicts the assumption in the lemma. □

Exercises

5.2.1. [25] Prove Lemma 5.2.1 on page 353 for the cases in which $\mathcal{B}$ is a discrete, possibly nonbinary, alphabet.

Comment. Source: T.M. Cover, J.A. Thomas, *Elements of Information Theory*, Wiley, 1991, pp. 300–301, attributed to I. Csiszár, *Studia Sci. Math. Hungar.*, 2(1967), 299–318, and others.

5.2.2. • [27] Define $S'_n(a) = \sum_{l(x)=n-1} \mu(x)(\mathbf{M}(a|x) - \mu(a|x))^2$, with $a \in \mathcal{B}$, with $\mathcal{B}$ the basic alphabet used in Theorem 5.2.1 on page 354. Let $S'_n = \sum_{a \in \mathcal{B}} S'_n(a)$. This is the summed expected squared difference.

(a) Show that if $\mathcal{B} = \{0, 1\}$ then $\sum_n S'_n(0), S'_n(1) \leq \frac{1}{2} K(\mu) \ln 2$.

(b) Show that for every finite, possibly nonbinary, alphabet $\mathcal{B}$ we have $\sum_n S'_n \leq K(\mu) \ln 2$.

Comments. The definition of S'_n is based on the *Euclidean distance* between two probability distributions P and Q over $\mathcal{B}$, defined as $E(P, Q) = \left[\sum_{a \in \mathcal{B}} (P(a) - Q(a))^2\right]^{1/2}$. Item (a) is Solomonoff's original version of Theorem 5.2.1 using the squared difference rather than the absolute difference. This was used in earlier editions of this book. Source for Item

(a): R.J. Solomonoff, *IEEE Trans. Inform. Theory*, 24(1978), 422–432. Source for Item (b): M. Hutter, *Proc. 12th Europ. Conf. Mach. Learn.*, Lect. Notes Artif. Int., Vol. 2167, Springer-Verlag, Berlin, 2001.

5.2.3. [20] Show the nonconvergence of the ratio in Theorem 5.2.2 for off-sequence prediction. Take the set of basic elements $\mathcal{B} = \{0, 1\}$. Define the measure $\mu(1|\omega_{1:n-1}) = (1 - \mu(0|\omega_{1:n-1})) = \frac{1}{2}n^{-3}$.

(a) Show that $\mu(0^n) \to 0.450\ldots$ for $n \to \infty$, so 0^∞ is μ-random.

(b) Show that $\mathbf{M}(0^{n-1}) = O(1)$, and $\mathbf{M}(0^{n-1}1) = 2^{-K(n)-O(1)}$.

(c) Show that $\mathbf{M}(1|0^{n-1})/\mu(1|0^{n-1}) = \Omega(n)$.

Comments. This illustrates Example 5.2.7 on page 358. Hint for Item (a): $\mu(0^n) = \prod_{i=1}^{n}(1 - \frac{1}{2}i^{-3})$. Hint for Item (c): $\mathbf{M}(1|0^{n-1}) = \Omega(2^{-K(n)}) = \Omega(1/n^2)$. Source: M. Hutter, *Universal Artificial Intelligence*, Springer-Verlag, Berlin, 2004.

5.2.4. [12] Show that for every semimeasure σ and measure μ the function $t(x) = \sigma(x)/\mu(x)$ is a μ-supermartingale.

5.2.5. [19] For every positive measure μ and every μ-supermartingale $t(x)$ as in Exercise 5.2.4, the set of infinite binary sequences ω such that there exists a finite limit of $t(\omega_{1:n})$ has μ-measure one.

Comments. Hint: use Claim 4.5.4 on page 322. Source: An.A. Muchnik (personal communication by N.K. Vereshchagin on January 9, 2004).

5.2.6. [33] (a) Let μ be a positive computable measure, conditionally bounded away from zero as in Definition 5.2.3 on page 359. Show that for every μ-supermartingale $t(x)$ as in Exercise 5.2.4, and for every number k, the set of infinite binary sequences ω such that there exists a number B (a limit) such that for every ϵ there is an n_0 such that for all $n > n_0$ and all x of length less than k we have $|t(\omega_{1:n}x) - B| < \epsilon$ has μ-measure one.

(b) There exists a positive computable measure μ such that for the μ-supermartingale $\mathbf{M}(x)/\mu(x)$ the set of infinite binary sequences defined in Item (a) has μ-measure $1 - \epsilon$ with $\epsilon > 0$.

Comments. Item (a) gives a formulation of the result on off-sequence convergence described in Theorem 5.2.2, Item (ii), on page 359. Namely, $\mathbf{M}(y|x)/\mu(y|x) = t(xy)/t(x)$ tends to $B/B = 1$, with $t(x) = \mathbf{M}(x)/\mu(x)$. Item (b) states that the condition 'there exists a constant c such that $\mu(0|x) > c, \mu(1|x) > c$ for all x' of Definition 5.2.3 is necessary. Therefore, Item (a) is the strongest formulation of Theorem 5.2.2, Item (ii): $\mathbf{M}(y|x)/\mu(y|x) = t(xy)/t(x)$ tends to $B/B = 1$, where $t(x) = \sigma(x)/\mu(x)$.

Source: An.A. Muchnik, *Ibid.*; M. Hutter and An.A. Muchnik, *Theoret. Comput. Sci.*, 382:3(2007), 247–261.

5.2.7. [38] Define $0'$-lower semicomputable as lower semicomputable by a Turing machine equipped with an oracle for the halting problem, Section 1.7.2. Here we use monotone Turing machines, since we want to deal with infinite sequences. Define $0'-\mu$-randomness of infinite binary sequences as previously μ-randomness, but this time using a $0'$-lower semicomputable semimeasure $\mathbf{M}'$ that is universal in the sense that it majorizes every $0'$-lower semicomputable semimeasure. For every positive computable measure μ, every universal semimeasure $\mathbf{M}(x)$ (equivalently, universal reference monotonic Turing machine that generates it), and for every $0'-\mu$-random infinite sequence ω, there exists a finite limit of $t(\omega_{1:n})$, where $t(x) = \mathbf{M}(x)/\mu(x)$,

Comments. This is a counterpart of Exercise 5.2.5; the counterpart of Exercise 5.2.6 is similar. Source: An.A. Muchnik, *Ibid.*; M. Hutter and An.A. Muchnik, *Ibid.*

5.2.8. • [40] (a) Show that there exists an infinite binary sequence ω that is λ-random (with λ the uniform measure), and is $0'$-computable, and there exists a universal semimeasure $\mathbf{M}$ (that is, a universal monotone Turing machine yielding this measure) such that $\mathbf{M}(\omega_{n+1}|\omega_{1:n}) \not\to \lambda(\omega_{n+1}|\omega_{1:n})$ for $n \to \infty$ (here $\lambda(x|l(x) = n) = 1/2^{l(x)}$).

(b) Show that for every universal semimeasure $\mathbf{M}$ there exist computable measures μ and non-μ-random sequences ω such that $\mathbf{M}(\omega_{n+1}|\omega_{1:n})/\lambda(\omega_{n+1}|\omega_{1:n}) \to 1$ for $n \to \infty$.

Comments. Item (a) states that for some reference universal monotonic machine with induced universal measure $\mathbf{M}$ and some random sequence (with respect to the uniform measure) the convergence does not hold. Hence, Theorem 5.2.2 on page 359 does not always hold for y fixed, and xy is an increasing initial segment of a random sequences ω—in contrast to what was claimed, without proof, in earlier editions of this book. In fact, these random sequences for which convergence fails are $0'$-computable (computable by Turing machines with an oracle for the halting problem). An.A. Muchnik supposed, though he did not know a proof, that for some other $\mathbf{M}$ the convergence is true for every random sequence. If so, then strengthening of the convergence in Theorem 5.2.2 on page 359 with μ-probability one to convergence for all individual random sequences is a matter of choice of reference universal monotonic machine. This is a deviation from the usual laws in Kolmogorov complexity that are invariant under change of reference universal machine. Hint for Item (a): Define $\omega = \omega_1\omega_2\ldots$ by $\omega_n = 0$ if $\mathbf{M}(\omega_{1:n-1}0|n) \leq 1/2^n$, and $\omega_n = 1$ otherwise. Prove that ω is λ-random. Construct a lower semicomputable semimeasure ν such that ν dominates $\mathbf{M}$ on ω, but $\nu(\omega_{n+1}|\omega_{1:n}) \not\to \lambda(\omega_{n+1}|\omega_{1:n})$. Then mix ν with $\mathbf{M}$ to make the mixture universal, but with a larger contribution of ν to preserve nonconver-

gence. Item (b) is a converse to Item (a). Source: An.A. Muchnik, *Ibid.*; M. Hutter and An.A. Muchnik, *Ibid.*

5.2.9. [43] Let $\mathcal{B}$ be a finite alphabet. Show that there exists a lower semicomputable semimeasure W such that for every computable positive measure μ and every μ-random infinite sequence ω we have $W(a|\omega_{1:n} \to \mu(a|\omega_{1:n})$ with $n \to \infty$, for every $a \in \mathcal{B}$.

Comments. This W is a universal predictor, but it is not universal, in that it does not multiplicatively dominate every lower semicomputable semimeasure, unlike $\mathbf{M}$. Source: M. Hutter and An.A. Muchnik, *Ibid.*

5.2.10. [14] Recall Equation 5.6 on page 363. Show that as the number m of examples $D = e_1, \ldots, e_m$ grows, the inferred probability $\Pr(H_k|D)$ is either monotonically nondecreasing to a limit or it suddenly falls to 0 and stays there thereafter, for every k. Argue why this process is called 'learning in the limit' and 'learning by enumeration.'

5.2.11. [27] We continue Exercise 5.2.10. Let $f_1, f_2, \ldots$ be the standard enumeration of the partial recursive functions.

(a) Give the implicit effective enumeration of the hypotheses for the prior P defined by $P(H_i) = 1/(i(i+1))$ and give the implicit incomputable enumeration according to the universal prior with the probability of H_i equal to $\mathbf{m}(i)$.

(b) Given an infinite sequence $e_1, e_2, \ldots$ of examples of f_i, the speed of learning a function f_i on that sequence is the minimal number m of examples $e_1, \ldots, e_m$ that contain counterexamples to every function f_j with $j < i$. Show that if $f = f_i$ is the function to be learned with i minimal, then there are sequences of examples with which we learn f about $i/2^{K(i)}$ times faster using prior $\mathbf{m}$ than using prior P. (If $i = 2^r$ then we learn exponentially faster using $\mathbf{m}$, and so on.) Moreover, for every sequence of examples we learn f at least as fast using prior $\mathbf{m}$ as we do using prior P, except possibly if $f = f_i$ is random in the sense that $K(i) \geq \log i$.

Comments. Hint: enumerate all hypotheses in two lists, one list $H_1, H_2, \ldots$ according to decreasing $P(H_i)$, and one list according to decreasing $\mathbf{m}(i)$, and use Gold's learning by enumeration, Section 5.2.6. Roughly speaking, the universal prior puts some simple hypotheses H_i with $K(i)$ very small (sometimes incomputably) much closer to the beginning of the list, but it does not put any hypotheses H_i later in the list than position $2^{K(i)} = O(i \log^2 i)$. Source: M. Li, P.M.B. Vitányi, *J. Comput. System Sci.*, 44:2(1992), 343–384.

5.2.12. [26] We apply Lemma 5.2.4 on page 365 to obtain insight into the relation between the number k of mistakes in the first m predictions

and the complexity of the m predicted target-function values. Define $x = f(1) \ldots f(m)$.

(a) Show that $\log \binom{m}{k} + K(m,k) + O(1) = k \log \frac{m}{k} + m \left(1 - \frac{k}{m}\right) \log \frac{1}{1-k/m} + O(\log m)$.

(b) Show that if k/m is small, then the right-hand side in Item (a) is about $k \log(m/k) + O(\log m)$, and hence $k \geq K(x)/\log(m/K(x))$, approximately. For instance, with $K(x) = \sqrt{m}$ we obtain $k \geq 2\sqrt{m}/\log m$.

(b) Show that if k/m is large, then the right-hand side in Item (a) approximates $mH(k/m)$ (the entropy of a $(k/m, 1 - k/m)$ Bernoulli process). For instance, if $k/m = \frac{1}{3}$, then $nH(\frac{1}{3}) \geq K(x)$.

(c) Show that another approximation than that used in Item (a) with k/m small gives $k \geq mH^{-1}(K(x)/m)$. For instance, if $K(x) = m$, then $k \geq m/2$.

Comments. Hint for Item (a): use Exercise 1.3.3 on page 10. Source: Discussions with P. Gács; see page 434.

5.3 Simple Pac-Learning

The model of pac-learning introduced in 1984 by L.G. Valiant is aimed at describing feasible learning in a rigorous manner to remedy lack of precision and objectivity in the ad hoc and vague descriptions of learning used previously. While the pac model is a first step toward a rigorous mathematical formulation of the notion of learning, it has turned out that its requirements are too strong. That is, many tasks that intuitively are learnable, or are clearly learnable in practice, turn out not to be learnable in the pac model. Moreover, there is a certain triviality in the pac-learning idea in the sense that in many cases, pac-learning algorithms are just algorithms that may return any hypothesis that is consistent with the examples.

It turns out that using the notion of universal distribution, one can both refine the pac learning concept and make it more sensible, while also the algorithms required to achieve this new *simple pac-learning* must do something cleverer than just be consistent.

5.3.1 Pac-Learning

We start with the basic theory of pac-learning. According to commonly accepted views in the theory of computation, 'feasibility' means that the learning algorithm should run in polynomial time and use a polynomial number of examples. This requirement implies that not all examples can turn up. Hence, it is impossible to infer a concept precisely. This means that one can only hope to learn the concept approximately. Moreover, one could be presented with unrepresentative examples for the concept to be inferred. It seems reasonable to assume that the examples are drawn randomly from a sample space according to a probability distribution.

The approximation between the target concept and the learned concept can be expressed in the probability of the set of examples on which the two concepts disagree.

The sample space S can be discrete ($\mathcal{N}$) or continuous ($\mathcal{R}$). The elements of S are called *examples*. A *concept* c is a subset of S. Abusing notation, we use c and the *characteristic function* $f : S \to \{0, 1\}$ of c interchangeably for the concept $c \subseteq S$ and for the syntactic representation of c. A *concept class* $\mathcal{C}$ is a set of concepts.

Consider a concept class $\mathcal{C}$. For each concept $f \in \mathcal{C}$ and example $v \in S$,

$$f(v) = \begin{cases} 1 & \text{if } v \text{ is a positive example of } f, \\ 0 & \text{if } v \text{ is a negative example of } f. \end{cases}$$

The learning algorithm draws examples from the sample space S according to a fixed but unknown probability distribution P. Each example in the sample comes with a label Positive or Negative.

Definition 5.3.1 A concept class $\mathcal{C}$ is *pac-learnable* (probably approximately correct learnable) if there exists a (possibly randomized) learning algorithm A such that for each $f \in \mathcal{C}$ and ϵ $(0 < \epsilon < 1)$, algorithm A halts in a finite number of steps and examples, and outputs a concept $h \in \mathcal{C}$ that satisfies the following: With probability at least $1 - \epsilon$,

$$\sum_{f(v) \neq h(v)} P(v) < \epsilon.$$

A concept class is *polynomially pac-learnable* if it is pac-learnable and the learning algorithm always halts within time and number of examples $p(l(f), 1/\epsilon)$, for some polynomial p.

Examples of concept classes are the set of finite automata, the set of Boolean formulas and the set of k-DNF formulas (Boolean formulas in disjunctive normal form such that each term has at most k literals).

5.3.2 Occam's Razor Formalized

Given a set of examples, Occam's razor tells us to choose the simplest concept consistent with the data. However, the simplest concept may be incomputable (in the sense of shortest effective program) or infeasible to find (in the sense of shortest description in some fixed syntax). For instance, computing a shortest k-DNF formula consistent with a given set of examples is NP-hard. It turns out that each polynomial-time reasonable approximation to a shortest rule can be used to achieve polynomial pac-learning.

Definition 5.3.2 Let $0 \leq \alpha < 1$ and $\beta \geq 1$ be constants, m the number of examples, and s the length (in number of bits) of the smallest concept in $\mathcal{C}$ consistent

with the examples. An *Occam algorithm* is a polynomial-time algorithm that finds a hypothesis $h \in \mathcal{C}$ consistent with the examples and satisfying $C(h) \leq s^\beta m^\alpha$.

Intuitively, for a sufficiently large (but still polynomial) set of examples, generated according to the unknown probability distribution P, any consistent hypothesis that is significantly smaller than the examples will be consistent with most of the unseen examples as well. The commonly used form of an Occam algorithm results from replacing $C(h)$ by the size of h. Our definition yields a stronger version (which is sometimes required) of following *Occam's razor theorem*.

Theorem 5.3.1 *A concept class $\mathcal{C}$ is polynomially pac-learnable if there is an Occam algorithm for it.*

Proof. Fix an error tolerance ϵ $(0 < \epsilon < 1)$. Choose m such that

$$m \geq \max\left\{\left(\frac{2\ s^\beta}{\epsilon}\right)^{1/(1-\alpha)},\ \frac{2}{\epsilon}\ \log\frac{1}{\epsilon}\right\}. \tag{5.8}$$

This is polynomial in s and $1/\epsilon$.

Claim 5.3.1 Let m be as in Equation 5.8. Let $\mathcal{C}$ be a set of r concepts, and let f be one of them. The probability that some concept $h \in \mathcal{C}$ both satisfies $P(f \neq h) \geq \epsilon$ and is consistent with m independent examples of f is at most $(1-\epsilon)^m r$.

Proof. If $P(h \neq f) \geq \epsilon$, then h is a *bad hypothesis*. That is, h and f disagree with probability at least ϵ on a random example. The set of bad hypotheses is denoted by B. Let E_h be the event that some fixed bad hypothesis h agrees with all m examples of f. Since the m examples of f are independent,

$$P(E_h) \leq (1-\epsilon)^m.$$

Since there are at most r bad hypotheses,

$$P\left(\bigcup_{h\in B} E_h\right) \leq (1-\epsilon)^m r.$$

□

The postulated Occam algorithm finds a hypothesis of Kolmogorov complexity at most $s^\beta m^\alpha$. The number r of hypotheses of this complexity satisfies

$$\log r \leq s^\beta m^\alpha.$$

By assumption on m,

$$r \leq (1-\epsilon)^{-m/2}.$$

(Use $\epsilon < -\log(1-\epsilon) < \epsilon/(1-\epsilon)$ for $0 < \epsilon < 1$.) Using the claim, the probability of producing a hypothesis with error larger than ϵ is at most

$$(1-\epsilon)^m r \leq (1-\epsilon)^{m/2}.$$

Substituting m, we find that the right-hand side is at most ϵ. □

Corollary 5.3.1 According to Definition 5.3.2 with $\alpha = 0$, the theorem says that if a learning algorithm compresses the data (of m examples where m satisfies Equation 5.8) to a representation of length s^β, that is, length polynomial in the length of the target concept, and the algorithm runs in time polynomial in the length of the target concept, then that algorithm is a polynomially pac-learning algorithm.

Informally, Occam's razor theorem says that given a set of positive and negative data, any consistent concept of size 'reasonably' shorter than the size of data is an 'approximately' correct concept with high probability. That is, if one finds a shorter representation of data, then one learns. The shorter the conjecture is, the more efficiently it explains the data, hence the more precise the future prediction.

5.3.3 Making Pac-Learning Simple

In the pac-learning model we require that the algorithm learn under all distributions. Therefore, pac-learning is also called *distribution-free learning*. It has turned out that many problems are intractable (NP-hard) in this model. Maybe it is feasible to learn only under *some* distributions, such as the computable ones. And maybe it is too much to ask to be able to learn all finite automata fast (humans cannot either), but surely we ought to be able to learn a sufficiently *simple* finite automaton fast (as humans can).

Pac-learning is not really distribution-free, since all examples are drawn independent and identically distributed (i.i.d.) from the same, arbitrary but fixed, distribution. This is much more restricted than the general Solomonoff setting we saw before, where every next example may depend on, and be differently distributed from, the previous ones. On the other hand, in the Solomonoff setting we require recursivity of the distribution, while in pac-learning we do not.

Certain concepts that are not known to be polynomially pac-learnable in the distribution-free model can be polynomially pac-learned under the uniform distribution by a specialized learning algorithm. However, learning under one distribution may be too restrictive to be useful. It may be useful to investigate polynomial pac-learnability under a certain class of distributions. One would like this class to be wide enough

to be interesting, yet restrictive enough to make more concept classes polynomially learnable. An outline of this section is as follows:

Discrete sample set. Let Q be a distribution on a discrete sample set. A concept class is polynomially pac-learnable under all distributions that are multiplicatively dominated by Q, *provided you draw your sample according to Q in the learning phase* iff the concept class is polynomially pac-learnable under Q. The caveat is that while pac-learning for distribution P, we need to draw the sample according to Q. We shall develop this theory with $Q = \mathbf{m}$, the universal lower semicomputable distribution. The remarkable property of this distribution is that in a polynomial sample, with overwhelming probability all examples of logarithmic complexity (the simple examples) will be represented. Hence, a learning algorithm just needs to reconstruct a concept approximately when all simple examples are given. This leads to a new type of learning algorithm.

Continuous sample set. Let μ be a semimeasure on a continuous sample set. A concept class is polynomially pac-learnable under all semimeasures that are multiplicatively dominated by μ iff the concept class is pac-learnable under μ. Here we have dropped the polynomial-time requirement and also replaced the requirement of drawing according to μ in the learning phase. We will take for μ the universal lower semicomputable semimeasure $\mathbf{M}$ and show that there is a class that is pac-learnable under $\mathbf{M}$ but not pac-learnable under all distributions.

5.3.4 Discrete Sample Space

Consider the discrete sample space $S = \mathcal{N}$. A distribution P is *simple* if it is (multiplicatively) dominated by the universal distribution $\mathbf{m}$; see Section 4.3.1. That is, there exists a constant c_P such that for all x,

$$c_P \, \mathbf{m}(x) \geq P(x).$$

The first question is how large the class of simple distributions is. It includes the lower semicomputable (and a fortiori the computable) distributions such as the uniform distribution, normal distribution, geometric distribution, and Poisson distribution (with computable parameters). It can be shown that there is a distribution that is simple but not lower semicomputable and that there is a distribution that is not simple. See Exercise 5.3.1 on page 381.

In the discrete case we needed to modify the standard pac-learning model by the requirement that the sample in the learning phase be drawn according to $\mathbf{m}$. A possible justification (which we give for what it is worth) is that in real life the examples are sometimes provided by mechanical

or artificial means or good-willed teachers, rather than provided according to the underlying distribution. Naturally, the simpler examples are provided first—that is, more or less according to $\mathbf{m}(x) = 2^{-K(x)}$. We prove the following completeness result:

Theorem 5.3.2 *A concept class $\mathcal{C}$ is polynomially learnable under the universal distribution $\mathbf{m}$ iff it is polynomially learnable under each simple distribution P, provided that in the learning phase the set of examples is drawn according to $\mathbf{m}$.*

Proof. Since P is simple, there is a constant $c_P > 0$ such that for all x,

$$c_P \ \mathbf{m}(x) \geq P(x).$$

Assume that $\mathcal{C}$ is polynomially pac-learnable under distribution $\mathbf{m}$. Then one can run the learning algorithms with error parameter ϵ/c_P in polynomial time t. Let *err* be the set of strings that are misclassified by the learned concept. So with probability at least $1 - \epsilon$,

$$\sum_{x \in err} \mathbf{m}(x) \leq \frac{\epsilon}{c_P}.$$

Then,

$$\sum_{x \in err} P(x) \leq c_P \sum_{x \in err} \mathbf{m}(x) \leq \epsilon.$$

If the underlying distribution is $P(\cdot)$ rather than $\mathbf{m}(\cdot)$, then we are guaranteed to pac-learn $\mathcal{C}$ (in time t) if sampling according to $\mathbf{m}(\cdot)$. □

There are two undesirable aspects of Theorem 5.3.2 in this easy textbook version. The algorithm must know $c_P = 2^{K(P)}$ depending on the unknown distribution P in order to determine error parameter ϵ/c_P. Secondly, in general we are not dealing with the sample space $\mathcal{N}$ but with a subset such as $D = \{0,1\}^n$. Hence we actually deal with $\mathbf{m}(\cdot|D)$. Both problems are technically resolved (essentially by polynomial oversampling) at the cost of somewhat complicating the statement of Theorem 5.3.2 and its proof in [M. Li and P.M.B. Vitányi, *SIAM J. Comput.*, 20:5 (1991), 911–935].

Since $\mathbf{m}(\cdot)$ assigns higher probabilities to simpler strings, one could suspect that after polynomially many examples, *all* simple strings are sampled and the probability that is concentrated on the unsampled strings is very low (inverse polynomial). However, this is not the case. Let E be any set of n^c examples. Then it can be shown (left to the reader as Exercise 5.3.2 on page 381) that

$$\sum_{x \notin E} \mathbf{m}(x) = \Omega\left(\frac{1}{\log^2 n}\right). \tag{5.9}$$

Previous approaches considered only syntactically described classes of concepts. Kolmogorov complexity allows us to introduce the idea of restricting a syntactically described class of concepts to the concepts that are simple in the sense of having short effective descriptions.

Example 5.3.1 Learning log-DNF. We consider a class not known to be polynomial pac-learnable (under all distributions). We show that this class is polynomially pac-learnable under simple distributions.

Definition 5.3.3 Consider concepts that are Boolean formulas over variables $x_1, \ldots, x_n$. A *literal* is either a variable x or its negation $\neg x$. Denote logical disjunction $\vee$ by $+$ and logical conjunction $\wedge$ by concatenation. A *DNF formula* is a Boolean formula f in disjunctive normal form $f = m_1 + \cdots + m_s$. The terms m_i are called *monomials*. Each m_i is a product of literals. A *CNF formula* is a Boolean formula g in conjunctive normal form $g = c_1 \cdots c_s$. The terms c_i are called *clauses*. Each c_i is a sum of literals.

The sample space is $S = \{0,1\}^n$. An example vector $v \in S$ represents a truth assignment to the n variables: if the ith element of v is 1, then $x_i = \textbf{true}$; otherwise $x_i = \textbf{false}$. An example v is a positive example for f if its truth assignment makes the whole formula f true. Denote this by $f(v) = 1$. Otherwise v is a negative example for f, denoted by $f(v) = 0$.

Assume that the positive examples are distributed according to some distribution P^+, and the negative examples according to some distribution P^-. The learning algorithm can choose to press a button to obtain an example from P^+, and it can press a button to obtain an example from P^-. (This is easily shown to be equivalent to the standard pac model with one distribution of positive and negative examples.) A k-DNF formula has at most k literals per monomial. Without loss of generality, let us assume that all k-DNF (over n variables) formulas have length at least n. The class k-DNF is known to be polynomially pac-learnable for the case that k is a fixed constant independent of n; for other k the status is unknown.

Let us generalize this a little. Write log-DNF to denote DNF formulas over n variables where each monomial term m has complexity $K(m) = O(\log n)$ and the length of the formula does not exceed a polynomial in n. This is a nontrivial superset of k-DNF. It is not known whether log-DNF is polynomially pac-learnable; we will show that it is polynomially pac-learnable under $\mathbf{m}$ (and hence simple-distribution-free polynomially pac-learnable in the sense of Theorem 5.3.2).

Lemma 5.3.1 *The class* log-DNF *is polynomially learnable under* $\mathbf{m}$.

Proof. Let $f(x_1, \ldots, x_n)$ be a log-DNF with each term of prefix complexity at most $c \log n$. If m is a monomial term in f, we write $m \in f$.

Sample n^d examples, where d is large enough to satisfy the argument below.

Claim 5.3.2 With probability greater than $1 - n^c/e^n$, all examples of the following form will be drawn:

For each monomial term $m \in f$, the example vectors 0_m, defined as the vectors that satisfy m and have 0 entries for all variables not in m; the example vectors 1_m, defined as the vectors that satisfy m and have 1 entries for all variables not in m.

Proof. Each monomial $m \in f$ satisfies $K(m) \leq c \log n$, and therefore $K(0_m) \leq c \log n + O(1)$. Therefore,

$$\mathbf{m}(0_m) \geq 2^{-c \log n - O(1)} \geq n^{-c-1},$$

for large enough n. This is the probability that 0_m will be sampled in one draw. Let E be the event that 0_m does not occur in n^d draws. Then

$$\Pr(E) < (1 - n^{-c-1})^{n^d} \leq \frac{1}{2} e^{-n},$$

for large enough d. The same estimate holds for the probability that the example 1_m is not sampled in n^d draws. There are only n^c possible monomials m such that $K(m) \leq c \log n$. Hence, the probability such that all vectors 0_m and 1_m associated with such monomials m are sampled is at least $1 - n^c/e^n$. □

Now we approximate f by the following learning procedure:

Step 0. Sample n^d examples according to $\mathbf{m}$. Set POS (NEG) to the set of positive (negative) examples sampled.

Step 1. For each pair of examples in POS, construct a monomial that contains x_i if both vectors have 1 in position i, contains $\neg x_i$ if both vectors have 0 in position i, and does not contain variable x_i otherwise ($1 \leq i \leq n$).

Step 2. Among the monomials constructed in Step 1, delete the ones that imply examples in NEG. **{**The remainder forms a set M**}**

Step 3. Set $A_m = \{v : m(v) = 1\}$. **{**That is, A_m is the set of positive examples implied by monomial m.**}** Use a greedy set-cover algorithm to find a small set, say C, of monomials $m \in M$ such that $\bigcup_{m \in C} A_m$ covers all positive examples in POS.

We need to prove the correctness of the algorithm.

Claim 5.3.3 With probability greater than $1 - n^c/e^n$, the set of monomials $\{m : m \in f\}$ is a subset of M.

Proof. By Claim 5.3.2, with probability at least $1 - n^c/e^n$, we draw all vectors 1_m and 0_m such that the monomial m has prefix complexity at most $c \log n$. In Step 1 of the algorithm we form the monomial m from examples 1_m and 0_m. Thus, with probability at least $1 - n^c/e^n$, all monomials of f belong to M. □

Of course, many other monomials consistent with the examples may also be in M. Finding all of the original monomials of f precisely is NP-hard. For the purpose of learning it is sufficient to approximate f. We use the following well-known approximation of the standard set-cover result.

Claim 5.3.4 Let $A_1, \ldots, A_n$ be sets such that $\bigcup_{i=1}^n A_i = A = \{1, \ldots, q\}$. If there exist k sets $A_{i_1}, \ldots, A_{i_k}$ such that $A = \bigcup_{j=1}^k A_{i_j}$, then it is possible to find in polynomial time $l = O(k \log q)$ sets $A_{h_1}, \ldots, A_{h_l}$ such that $A = \bigcup_{j=1}^l A_{h_j}$.

Let f have k monomials. The k monomials cover the positive examples in the sense that POS $\subseteq \bigcup_{m \in f} A_m$. There are 2^n examples. By Claim 5.3.4, we can find $O(kn)$ monomials to approximate f and cover the examples in POS in polynomial time. Occam's razor theorem, Theorem 5.3.1, implies that our algorithm polynomially learns log-DNF. □

Step 3 of the above algorithm is needed to compress the data in order to apply Occam's razor theorem. One may wonder whether one can encode each monomial as binary vectors efficiently, and hence sample all binary vectors of prefix complexity $c \log n$; then decode these into monomials to get all monomials of prefix complexity $c \log n$; then run the set-cover algorithm to choose a small set of monomials to achieve learning. But this does not work, since there are 2^n 0-1 vectors and 3^n monomials of n variables. It can be shown that there is no effective encoding scheme that selectively codes only 2^n monomials including all monomials of prefix complexity $c \log n$. ◇

If $\mathbf{m}$ is given as a table, then one can randomly sample according to $\mathbf{m}$ as explained in Example 4.4.3 on page 293. Unfortunately, $\mathbf{m}$ is not computable. This problem is eliminated by using a time-bounded version of $\mathbf{m}$. Then the entire theory still holds in a scaled-down version as discussed in Example 7.6.2 on page 585.

5.3.5 Continuous Sample Space

Consider a continuous sample space $S = \{0,1\}^\infty$, that is, the set of one-way infinite binary sequences or, equivalently, the set of real numbers in the unit interval $[0,1]$ (Section 1.6). For example, the uniform distribution now is defined as $\lambda(\Gamma_x) = 2^{-2l(x)-1}$, where Γ_x denotes the set of all one-way infinite binary strings starting with x. This is the uniform measure on the interval $[0,1]$. We will use $\mathbf{M}$, the continuous universal lower semicomputable measure of Theorem 4.3.1, page 267.

Definition 5.3.4 A measure μ over S is *simple* if it is dominated multiplicatively by $\mathbf{M}$, in the sense of $\mu(x) = O(\mathbf{M}(x))$.

Similar to the discrete case one can show that there are simple measures that are not lower semicomputable, and there are measures that are not simple.

A concept class is a subset $\mathcal{C} \subseteq 2^S$ of concepts, each of which is a measurable set. If c is a concept to be learned, then $\omega \in S$ is a positive example if $\omega \in c$, and it is a negative example if $\omega \in S - c$. The remaining definitions of pac-learning can now be rephrased in the continuous setting in the obvious way. If c, c' are two sets, then $c \Delta c'$ is the symmetric set difference $(c \bigcup c') - (c \bigcap c')$.

While for discrete sample spaces all concept classes are pac-learnable (although not all are polynomially pac-learnable), this is not the case for continuous sample spaces. Here we show that all continuous concept classes are pac-learnable over each simple measure μ iff they are pac-learnable under the universal measure $\mathbf{M}$. In contrast with polynomial pac-learning of discrete concepts, we do not need to require (but do allow) that the learning algorithm sample according to the universal measure.

Theorem 5.3.3 *A concept class $\mathcal{C}$ of concepts in S is pac-learnable under $\mathbf{M}$ iff it is learnable under each simple measure.*

Proof. (If) This holds vacuously.

(Only if) We say that $\mathcal{C}_\epsilon \subseteq 2^S$ is an *ϵ-cover* of $\mathcal{C}$, with respect to distribution μ, if for every $c \in \mathcal{C}$ there is a $\hat{c} \in \mathcal{C}_\epsilon$ that is ϵ-close to c ($\mu(c \Delta \hat{c}) < \epsilon$). A concept class $\mathcal{C}$ is *finitely coverable* if for every $\epsilon > 0$ there is a finite ϵ-cover $\mathcal{C}_\epsilon$ of $\mathcal{C}$, everything with respect to a given measure μ. A finite ϵ-cover $\mathcal{C}_\epsilon$ has finitely many concepts c, and each c is in the closure of the set of cylinders under finite union and complement (and finite intersection).

Claim 5.3.5 A concept class $\mathcal{C}$ is finitely coverable with respect to μ iff $\mathcal{C}$ is learnable with respect to μ.

Proof. The proof does not use Kolmogorov complexity, so there is no reason to give it here. It is due to G. Benedek and A. Itai, *Theoret. Comput. Sci.*, 86:2(1991), 377–390; also reproduced in [M. Li and P.M.B. Vitányi, *SIAM J. Comput.*, 20(1991), 911–935]. □

Using Claim 5.3.5, if $\mathcal{C}$ can be learned under $\mathbf{M}$, then $\mathcal{C}$ can be finitely covered with respect to $\mathbf{M}$. Let μ be a simple distribution and $d > 0$ such that $\mathbf{M}(x) \geq d\mu(x)$ for all x. Then every finite ϵd-cover of $\mathcal{C}$ with respect to $\mathbf{M}$ is also a finite ϵ-cover with respect to μ. Using Claim 5.3.5 again, it follows that $\mathcal{C}$ is learnable with respect to μ. This finishes the proof of the theorem. □

Note that this is a strong statement, since we are saying that if one can learn under $\mathbf{M}$, then one can also learn under every simple measure μ while sampling according to μ. In the polynomial learning of discrete concepts we had to require sampling according to $\mathbf{m}$. This improvement is made possible by relaxing the polynomial-time learning requirement to just learning.

Example 5.3.2 If all continuous concept classes that can be simple pac-learned could also be pac-learned, then Theorem 5.3.3 would be vacuously true. However, there exist continuous concept classes that can be simple pac-learned but cannot be pac-learned under all distributions. Let the witness class $\mathcal{C}$ consist of all concepts of the form $c = \bigcup\{\Gamma_x : x \in I\}$, where I satisfies the condition that if $x, y \in I$ and $x \neq y$, then either $\mathbf{M}(x) \geq 2\mathbf{M}(y)$ or vice versa. (As usual, the cylinder Γ_x is the set of all infinite binary sequences starting with x, and $\mathbf{M}(x)$ is the $\mathbf{M}$-measure of Γ_x.)

Given a continuous concept class $\mathcal{C}$ (as defined in Section 5.3.3) and a finite set $E \subseteq S = \{0,1\}^\infty$, if $\{E \bigcap c : c \in \mathcal{C}\} = 2^E$, then we say that E is *shattered* by $\mathcal{C}$. The *Vapnik–Chervonenkis (VC) dimension* of $\mathcal{C}$ is the smallest integer d such that no $E \subseteq S$ of cardinality $d+1$ is shattered by $\mathcal{C}$; if no such d exists, then the dimension of $\mathcal{C}$ is infinite. It is known that a class $\mathcal{C}$ has finite VC-dimension iff it is pac-learnable.

This was proven by A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth *J. ACM*, 35:4(1989), 929–965. The fundamental Occam's razor theorem also appears there.

It can be easily shown, and it is left to the reader as Exercise 5.3.9 on page 382, that $\mathcal{C}$ has an infinite VC-dimension and therefore is not pac-learnable under all measures. On the other hand, $\mathcal{C}$ is finitely coverable under $\mathbf{M}$, and therefore pac-learnable with respect to $\mathbf{M}$ (and hence under all simple measures). ◇

Exercises

5.3.1. [27] Consider discrete distributions. Show that there is a simple distribution that is not lower semicomputable, and there is a distribution that is not simple.

Comments. Therefore, the simple distributions properly contain the lower semicomputable distributions but do not include all distributions. Source: this and the next three exercises are from [M. Li and P.M.B. Vitányi, *SIAM J. Comput.*, 20:5(1991), 911–935].

5.3.2. [24] Prove Equation 5.9. Can you improve this bound?

5.3.3. [30] Consider DNF over n variables. A DNF formula f is *simple* if for each term m of f, there is a vector $v_m \in \{0,1\}^n$ that satisfies m but does not satisfy any other monomials of f even by changing one bit and $K(v_m) = O(\log n)$. Simple DNFs can contain many high-prefix-complexity terms as opposed to $O(\log n)$-DNF. For example, take $y \in \{0,1\}^n$ with $K(y) \geq n - O(1)$. Then the number of 1's in y is about $\frac{1}{2}n$. Construct a term m containing x_i if $y_i = 1$, and neither x_i nor $\neg x_i$ otherwise. Then $K(m) \geq n - O(1)$ but the vector 1_m consisting of all 1's satisfies m and has $K(1_m) = O(\log n)$. The class of simple DNF is pretty general. Show that it is polynomially learnable under **m**.

5.3.4. [25] Show that log-DNF of Example 5.3.1 does not contain and is not contained in simple DNF of Exercise 5.3.3.

5.3.5. [33] A *k-decision list* over n variables is a list of pairs $L = (m_1, b_1), \ldots, (m_s, b_s)$, where m_i is a monomial of at most k variables and $b_i \in \{0,1\}$, for $1 \leq i \leq s$, except that always $m_s = 1$. A decision list L represents a Boolean function f_L defined as follows: For each example $v \in \{0,1\}^n$, let $f_L(v) = b_i$, where i is the least index such that v satisfies m_i. Since there are at most $(2n)^{k+1}$ monomials of k literals over n variables, we have $s \leq (2n)^{k+1}$.

(a) Using Occam's razor theorem, Theorem 5.3.1, we show that k-decision lists are polynomially pac-learnable.

(b) Let us define a *log-decision* list to be a decision list with each term having prefix complexity $O(\log n)$. Show that log-decision list is polynomially learnable under **m**.

Comments. Source: Item (a) [R. Rivest, *Machine Learning*, 2:3(1987), 229–246]. Item (b) was stated as an open problem in the first edition of this book and was solved by J. Castro and J.L. Balcázar [pp. 239–248 in: *Proc. 6th Int. Workshop on Algorithmic Learning Theory, Lect. Notes Artific. Intell.*, Vol. 997, Springer-Verlag, Berlin, 1995]. They also show that simple decision lists, a generalization of the simple DNF of Exercise 5.3.3, are polynomial learnable under **m**.

5.3.6. [O35] Are any of log-DNF, simple DNF, log-decision list polynomially pac-learnable?

5.3.7. [35] A Boolean formula is *monotone* if no literal in it is negated. A *k-term DNF* is a DNF consisting of at most k monomials.

(a) Show that pac-learning monotone k-term DNF requires more than polynomial time unless RP=NP.

(b) Show that the class of monotone k-term DNF is polynomially pac-learnable under **m** (even with the algorithm learning the exact concept with high probability).

Comments. This shows that more concepts are simple pac-learnable than are pac-learnable unless RP=NP. Source: for Item (a) L. Pitt and L.G. Valiant, *J. ACM*, 35(1989), 965–984; for Item (b) M. Li and P.M.B. Vitányi, *SIAM J. Comput.*, 20:5(1991), 911–935.

5.3.8. [32] Show that the class of deterministic finite-state automata (DFA) whose canonical representations have logarithmic Kolmogorov complexity is polynomially pac-learnable under **m**.

Comments. It is known that both exact and approximate (in the pac sense) identification of DFA is NP-hard. Source: R. Parekh, V. Honavar, *Machine Learning*, 44:1-2(2001), 9–35.

5.3.9. [M36] Show that the continuous concept class $\mathcal{C}$ defined in Example 5.3.2 is pac-learnable under all simple measures but not pac-learnable (that is, under all measures).

Comments. Source: M. Li and P.M.B. Vitányi, *SIAM J. Comput.*, 20:5 (1991), 911–935.

5.4 Hypothesis Identification by MDL

We can formulate scientific theories in two steps. First, we formulate a set of possible alternative hypotheses, based on scientific observations or other data. Second, we select one hypothesis as the most likely one. Statistics is the mathematics of how to do this. A relatively recent paradigm in statistical inference was developed by J.J. Rissanen and by C.S. Wallace and his coauthors. The method can be viewed as a computable approximation to the incomputable approach in Section 5.2 and was inspired by it. In accordance with Occam's dictum, it tells us to go for the explanation that compresses the data the most. The minimum description length principle tells us:

> **MDL principle.** Given a sample of data and an effective enumeration of the appropriate alternative theories to explain the data, the best theory is the one that minimizes the sum of
>
> - the length, in bits, of the description of the theory;

- the length, in bits, of the data when encoded with the help of the theory.

The idea of a two-part code for a body of data D is natural from the perspective of Kolmogorov complexity. If D does not contain any regularities at all, then it consists of purely random data, and there is no hypothesis to identify. Assume that the body of data D contains regularities. With help of a description of those regularities (a model) we can describe the data compactly. Assuming that the regularities can be represented in an effective manner (that is, by a Turing machine), we encode the data as a program for that machine. Squeezing all effective regularity out of the data, we end up with a Turing machine representing the meaningful regular information in the data together with a program for that Turing machine representing the remaining meaningless randomness of the data, as in Section 2.1.1. MDL is based on striking a balance between regularity and randomness in the data. All we will ever see are the data at hand (if we know more, then in fact we possess more data, which should be used as well). The best model or explanatory theory is taken to be the one that optimally uses regularity in the data to compress. 'Optimally' is used in the sense of maximal compression of the data in a two-part code, that is, a model and a description from which the data can be reproduced with help of the model. If the data are truly random, no compression is possible and the optimum is reached for the empty model. The empty model has description length 0. If the data are regular, then compression is possible, and using the MDL principle identifies the optimal model.

We call such a model or theory, a 'hypothesis.' With a more complex description of the hypothesis H, it may fit the data better and therefore decrease the misclassified data. If H describes all the data, then it does not allow for measuring errors. A simpler description of H may be penalized by an increase in misclassified data. If H is a trivial hypothesis that contains nothing, then all data are described literally and there is no generalization. The rationale of the method is that a balance between these extremes seems to be required.

5.4.1 Ideal MDL

The analysis of both hypothesis identification by ideal MDL below, and of prediction in Section 5.2.4, shows that maximally compressed descriptions give good results on data samples that are random or typical with respect to the contemplated models. Let us assume that H and D are expressed as natural numbers or finite binary strings.

Definition 5.4.1 The *ideal MDL* principle selects the hypothesis H that minimizes

$$K(H) + K(D|H), \tag{5.10}$$

which is the code-independent, recursively invariant, absolute form of the MDL principle.

Example 5.4.1 **(Alternative MDL principle)** In Theorem 3.9.1 on page 247 it is shown that symmetry of information holds for individual objects in the following sense:

$$\begin{aligned} K(H,D) &= K(H) + K(D|H, K(H)) + O(1) \\ &= K(D) + K(H|D, K(D)) + O(1). \end{aligned}$$

Substitution gives $K(H|D) = K(H,D) - K(D)$, up to an $O(\log K(H,D))$ additive term. The term $K(D)$ is fixed and doesn't change for different H's. Minimizing the left-hand term, $K(H|D)$ can then be interpreted as

> Given a hypothesis space **H**, we want to select the hypothesis H such that the length of the shortest encoding of D together with hypothesis H is minimal.

Since it is not more difficult to describe some object if we get more conditional information, we have $K(D|H, K(H)) \leq K(D|H) + O(1)$. Using the last displayed equations, the quantity in Equation 5.10 satisfies

$$K(H) + K(D|H) \geq K(H,D) + O(1) \geq K(D) + O(1),$$

with equalities for the trivial hypothesis $H_0 = D$ or $H_0 = \varnothing$. At first glance this would mean that the hypothesis $H_{\rm mdl}$ that minimizes the sum of Equation 5.10 could be set to D or $\varnothing$, which is absurd in general. It is exactly the solution how to prevent such trivialities that gives us the key to the very meaning of ideal MDL, and by extension some insight into applied versions. ◇

5.4.2 Logarithmic Version of Bayes's Rule

Bayes's rule, Equation 5.13 on page 345, maps input $(P(H), D)$ to output $\Pr(H|D)$—the *posterior* probability. For many model classes (such as Bernoulli processes and Markov chains), if the number n of data generated by a true model in the class increases, then the total inferred probability can be expected to concentrate on the true hypothesis (with probability one for $n \to \infty$). That is, as n increases, the weight of the factor $\Pr(D|H)/\Pr(D)$ dominates the influence of the prior $P(\cdot)$ for typical data—by the law of large numbers. The importance of Bayes's rule is that the inferred probability gives us as much information as possible about the possible hypotheses from only a small number of (typical) data and the prior probability.

We are concerned only with finding the H that maximizes $\Pr(H|D)$ with D and P fixed. Taking the logarithms of both sides of the equation, this

is equivalent to *minimizing* the expression $\log 1/\Pr(H|D)$ over H:

$$\log \frac{1}{\Pr(H|D)} = \log \frac{1}{\Pr(D|H)} + \log \frac{1}{P(H)} - \log \frac{1}{\Pr(D)}.$$

Since the probability $\Pr(D)$ is constant under varying H, this means that we want to find a hypothesis H that minimizes

$$\log \frac{1}{\Pr(D|H)} + \log \frac{1}{P(H)}. \tag{5.11}$$

5.4.3 Discrepancy Between Probability and Complexity

To obtain ideal MDL from Equation 5.11, we need to replace the log-inverse probability terms by prefix complexities. There is a prefix-code E_Q with $l(E_Q(x)) = \log 1/Q(x)$ by Example 1.11.2 on page 68. This is the Shannon–Fano code for an alphabet of source words distributed according to probability Q. This code realizes the least expected code-word length among all prefix-codes up to one bit. Therefore, the H that minimizes Equation 5.11, rewritten as

$$l(E_{\Pr(\cdot|H)}(D)) + l(E_P(H)),$$

minimizes the sum of two prefix-codes in *expectation*.

However, in ideal MDL we want to minimize the sum of the effective description lengths of the *individual* elements H, D involved, as in Equation 5.10. Thus, we cannot simply replace the logarithmic terms in Equation 5.11 by corresponding $K(\cdot)$ terms for each individual case. Note that we can do so for the expectations, since the prefix-code with code-word length $K(x)$ for object x also has expected shortest code-word length in the sense of Theorem 8.1.1 on page 604.

To satisfy Equation 5.10 we are free to make the new *assumption* that the prior probability P in Bayes's rule, Equation 5.1, is fixed as $\mathbf{m}$, as in Exercise 5.4.4 on page 401.

However, we *cannot assume* that the probability $\Pr(\cdot|H)$ equals $\mathbf{m}(\cdot|H)$. Namely, as explained at length in Section 5.1.3, probability $\Pr(\cdot|H)$ may be totally determined by the hypothesis H. Depending on H therefore, $l(E_{\Pr(\cdot|H)}(D))$ may be *very* different from $K(D|H)$. This holds especially for simple data D that have low probability under the assumption of hypothesis H.

Example 5.4.2 Let us look at a simple example evidencing this discrepancy. Suppose we flip a coin of unknown bias n times. Let hypothesis H and data D be defined by:

$$H := \text{ Probability of 'heads' is } \tfrac{1}{2},$$
$$D := \underbrace{hh\ldots h}_{n \text{ times } h(\text{eads})}$$

Then we have $\Pr(D|H) = 1/2^n$ and

$$l(E_{\Pr(\cdot|H)}(D)) = \log \frac{1}{\Pr(D|H)} = n.$$

In contrast, $K(D|H) \leq \log n + 2 \log\log n + O(1)$. ◇

5.4.4 Resolving the Discrepancy

To obtain the ideal MDL principle, Equation 5.10 on page 383, from Bayes's rule we need to replace the logarithmic inverse probabilities involved in Equation 5.11 on page 385 by the corresponding prefix complexities. The theory dealing with randomness of individual objects states that under certain conditions the logarithm of the inverse probability and the prefix complexity are close. This amounts to approximately replacing the probabilities involved by the corresponding universal probabilities $\mathbf{m}(\cdot)$ of Theorem 4.3.1 on page 267 and applying the coding theorem, Theorem 4.3.3 on page 273.

Lemma 5.4.1 (i) *If $P = \mathbf{m}$, then $\log 1/P(H) = K(H)$ up to an $O(1)$ additive term.*

(ii) *If P is computable and H is P-random, then $\log 1/P(H) = K(H)$ up to an additive term $K(P)$.*

(iii) *If $\Pr(\cdot|H)$ is computable and D is $\Pr(\cdot|H)$-random, then $\log 1/\Pr(D|H) = K(D|H)$ up to an additive term $K(\Pr(\cdot|H)) + O(1) \leq K(H) + O(1)$.*

Proof. The functions Pr and P are computable if Pr can be computed to any required precision for each argument D and conditional H, while P can be computed to any required precision for each argument H.

(i) If the prior probability $P(H)$ is $\mathbf{m}(H)$, then $\log \mathbf{m}(H)/P(H) = 0$, that is, every object is $\mathbf{m}$-random, Example 4.3.11 on page 283. Therefore, by the coding theorem, Theorem 4.3.3, we have

$$\left| K(H) - \log \frac{1}{P(H)} \right| = O(1).$$

(ii) If $P(\cdot)$ is computable and H is P-random, then

$$\mathbf{m}(H) \geq 2^{-K(P)} P(H),$$
$$\log \frac{\mathbf{m}(H)}{P(H)} \leq K(P).$$

The first inequality holds since $\mathbf{m}$ majorizes P this way; the second inequality expresses the assumption of randomness of H, where $K(P)$ is the length of the shortest self-delimiting program for the reference universal prefix machine to simulate the Turing machine computing the

probability mass function $P : \mathcal{N} \to [0,1]$. That is, it is the shortest effective self-delimiting description of P. Then,

$$\left| K(H) - \log \frac{1}{P(H)} \right| \leq K(P). \tag{5.12}$$

(iii) Firstly, by Example 4.3.3 on page 269 following Theorem 4.3.1 on page 267,

$$\mathbf{m}(D|H) \geq 2^{-K(\Pr(\cdot|H))} \Pr(D|H).$$

Therefore,

$$\log \frac{\mathbf{m}(D|H)}{\Pr(D|H)} \geq -K(\Pr(\cdot|H)). \tag{5.13}$$

Secondly, we call the data sample D *sufficiently random* with respect to the computable probability mass function $\Pr(\cdot|H)$ (with respect to H therefore) in the sense of Martin-Löf if

$$\log \frac{\mathbf{m}(D|H)}{\Pr(D|H)} \leq K(\Pr(\cdot|H)) + O(1). \tag{5.14}$$

Here $\kappa_0((D|H)|\Pr(\cdot|H)) = \log(\mathbf{m}(D|H)/\Pr(D|H))$ is a universal sum P-test as in Theorem 4.3.5, page 279. In Theorem 4.3.5, page 279, a finite object x is called P-random if $\kappa_0(x|P) = \log \mathbf{m}(x)/P(x) \leq 0$. However, as remarked in Chapter 4, randomness of finite objects is but a matter of degree, and the different randomness tests treated vary slightly in strength. Here we take the test according to Equation 5.14. It has the advantage of being symmetric in the sense that the outcome for random objects is in the interval $[-K(P), K(P)]$, up to additive constants, and also guarantees that objects are P-random with overwhelming P-probability.

With overwhelming Pr-probability, see Exercise 5.4.1 on page 400, the D's are random in this sense because for each H we have

$$\sum_D \Pr(D|H) 2^{\kappa_0((D|H)|\Pr(\cdot|H))} = \sum_D \mathbf{m}(D|H) \leq 1,$$

since $\mathbf{m}(\cdot|H)$ is a discrete semimeasure. For D's that are random in the appropriate sense, Equations 5.13 and 5.14 mean by the coding theorem, Theorem 4.3.3, that

$$\left| K(D|H) - \log \frac{1}{\Pr(D|H)} \right| \leq K(\Pr(\cdot|H)) + O(1).$$

Note that $K(\Pr(\cdot|H)) \leq K(H) + O(1)$, since from H we can compute $\Pr(\cdot|H)$ by the assumption of the computability of $\Pr(\cdot|\cdot)$. □

Example 5.4.3 Let us look at an example of a distribution for which some hypotheses are random, and some other hypotheses are nonrandom. Let the possible hypotheses correspond to the binary strings of length n, while P is the uniform distribution that assigns probability $P(H) = 1/2^n$ to each hypothesis $H \in \{0,1\}^n$. Then $H = 00\ldots0$ has low complexity $K(H) \leq \log n + 2\log\log n$. However, $\log 1/P(H) = n$. Therefore, by Equation 5.12, H is not P-random. If we obtain H by n flips of a fair coin, then with overwhelming probability we will have that $K(H) = n + O(\log n)$, and therefore $\log 1/P(H) \approx K(H)$ and H is P-random. ◇

Example 5.4.4 The notion that the data D is $\Pr(\cdot|H)$-random means that the data are typical for the probability mass function $\Pr(\cdot|H)$ induced by the hypothesis H. If, for example, $H = (\mu, \sigma)$ induces the Gaussian distribution $\Pr(\cdot|(\mu,\sigma)) = N(\mu,\sigma)$ and the data D are concentrated in a tail of this distribution, such as $D = 00\ldots0$, then D is atypical with respect to $\Pr(\cdot|H)$ in the sense of being nonrandom because it violates the $\Pr(\cdot|H)$-randomness test of Equation 5.14. ◇

Example 5.4.5 Lemma 5.4.1 does not hold if either D is not $\Pr(\cdot|H)$-random and therefore by Equation 5.14 we have $\log 1/\Pr(D|H) > K(D|H)$ which means that the data is not typical for H, or H is not P-random and therefore $\log 1/P(H) > K(H)$ which means that the hypothesis is not typical for the prior distribution. We give an example of the first case:

We sample a polynomial $H_2 = ax^2 + bx + c$ at n arguments chosen uniformly at random from the interval $[0,1]$. The sampling process introduces Gaussian errors in the function values obtained. The set of possible hypotheses is the set of polynomials. Assume that all numbers involved are of fixed bounded accuracy.

Because of the Gaussian error in the measuring process, with overwhelming probability the only polynomials H' that fit the sample precisely are of degree $n-1$. Denote the data sample by D. Such a hypothesis H' is likely to minimize $K(D|H)$, since we just have to describe the n lengths of the intervals between the sample points along the graph of H'. However, data sample D is certainly not $\Pr(\cdot|H')$-random, since it is extremely unlikely, and hence atypical, that D arises in sampling H' with Gaussian error. Therefore, Equation 5.14 is violated, which means that $\log 1/\Pr(D|H') > K(D|H')$, which contradicts Lemma 5.4.1, Item (iii).

In contrast, with overwhelming likelihood H_2 will show that the data sample D is random to it. That being the case, Lemma 5.4.1, Item (iii), holds. Now what happens if H' is the true hypothesis and the data sample D by chance is as above? In that case Lemma 5.4.1, Item (iii), does not hold, and Bayes's rule and MDL may each select very different hypotheses as the most likely ones. ◇

Definition 5.4.2 Given data sample D and prior probability P, hypotheses H are called *admissible* if both data D is $\Pr(\cdot|H)$-random and H is P-random.

Theorem 5.4.1 *Let the data sample be D and let the corresponding set of admissible hypotheses be $\mathcal{H}_D \subseteq \mathcal{H}$. Then the maximum a posteriori probability hypothesis $H_{\rm bayes} \in \mathcal{H}_D$ in Bayes's rule and the hypothesis $H_{\rm mdl} \in \mathcal{H}_D$ selected by ideal MDL are roughly equal:*

$$|\log \frac{\Pr(H_{\rm mdl}|D)}{\Pr(H_{\rm bayes}|D)}| \leq 2\alpha(P,H), \tag{5.15}$$
$$|K(D|H_{\rm mdl}) + K(H_{\rm mdl}) - K(D|H_{\rm bayes}) - K(H_{\rm bayes})| \leq 2\alpha(P,H),$$

with $\alpha(P,H) = K(\Pr(\cdot|H)) + K(P) + O(1)$. If $P = \mathbf{m}$ then we set $K(P) = 0$.

Proof. Assume that Lemma 5.4.1 holds. By definition, $H = H_{\rm mdl}$ minimizes $K(H|D) + K(H)$. Denote the minimal value by A. Then in Theorem 5.4.1 the H' that minimizes $\log 1/\Pr(D|H) + \log 1/P(H)$ yields a value B with $|A - B| \leq \alpha(P,H)$. This is easy to see, as follows. If $A - B > \alpha(P,H)$ then $K(H'|D) + K(H') < A$ by Theorem 5.4.1, contradicting that A is the minimum for the sum of the complexities. If $B - A > \alpha(P,H)$ then $\log 1/\Pr(D|H_{\rm mdl}) + \log 1/P(H_{\rm mdl}) < B$, contradicting that B is the minimum for the sum of the log inverse probabilities. Now $H = H_{\rm bayes}$ maximizes $\Pr(H|D) = \Pr(D|H)P(H)/\Pr(D)$ with $\Pr(D)$ constant and therefore $H_{\rm bayes}$ is an H' that minimizes $\log 1/\Pr(D|H) + \log 1/P(H)$. Denote $\log 1/\Pr(D|H_{\rm mdl}) + \log 1/P(H_{\rm mdl})$ by B'. Then by Theorem 5.4.1 we have $|A - B'| \leq \alpha(P,H)$ and therefore

$$|B - B'| \leq 2\alpha(P,H).$$

By Bayes's rule, $B + \log \Pr(D) = \log 1/\Pr(H_{\rm bayes}|D)$ and $B' + \log \Pr(D) = \log 1/\Pr(H_{\rm mdl}|D)$, where $\log \Pr(D)$ is constant. Substitution in the displayed equation yields the first inequality of the theorem. The second inequality follows by the same argument with the roles of complexities and log inverse probabilities interchanged. □

If $\alpha(P,H)$ is small then this means that the prior distribution P is simple or $\mathbf{m}$ and that the probability mass function $\Pr(\cdot|H)$ over the data samples induced by hypothesis H is simple. The theorem states that the hypothesis selected by MDL and by Bayesianism are close both in the sense of a posteriori probability (the Bayesian criterion) and in sum of the complexities of the hypothesis and the data encoded with the help of the hypothesis (the MDL criterion). In contrast, if $\alpha(P,H)$ is large, which means that either of the mentioned distributions is not simple, for example when $K(\Pr(\cdot|H)) = K(H)$ for complex H, then the discrepancy can widen for both criteria.

As a consequence, if $\alpha(P, H)$ is small enough, and Bayes's rule selects an admissible hypothesis, and so does ideal MDL, then both criteria are (approximately) optimized by both selected hypotheses. In Exercise 5.4.4 on page 401 it is shown that this is, in a certain sense, a likely situation.

5.4.5 Applying Minimum Description Length

Unfortunately, the function K is not computable (Section 3.4). For practical applications one must settle for easily computable approximations. One way to do this is as follows: First encode both H and $D|H$ by a simply computable bijection as a natural number in $\mathcal{N}$. Assume that we have some standard procedure to do this. Then consider a simple self-delimiting description of x. For example, x is encoded by $x' = 1^{l(x)}0l(x)x$. This makes $l(x') = \log x + 2\log\log x + 1$, which is a simple upper approximation of $K(x)$; see Section 3.2. Since the length of code-word sets of prefix-codes corresponds to a probability distribution by Kraft's inequality, Theorem 1.11.1, this encoding corresponds to assigning probability $2^{-l(x')}$ to x. In the MDL approach, this is the specific usable approximation to the universal prior. In the literature we find a more precise approximation, which, however, has no practical meaning. For convenience, we smooth our encoding as follows.

Definition 5.4.3 Let $x \in \mathcal{N}$. The *universal MDL prior* over the natural numbers is $M(x) = 2^{-\log x - 2\log\log x}$.

In the Bayesian interpretation the prior distribution expresses one's prior knowledge about the true value of the parameter. This interpretation may be questionable, since the used prior is usually not generated by repeated random experiments. In Rissanen's view, the parameter is generated by the selection of the class of hypotheses and it has no inherent meaning. It is just one means to describe the properties of the data. The selection of H that minimizes $K(H) + K(D|H)$ (or Rissanen's approximation thereof) allows one to make statements about the data. Since the complexity of the models plays an important part, the parameters must be encoded. To do so, we truncate them to a finite precision and encode them with the prefix-code above. Such a code happens to be equivalent to a distribution on the parameters. This may be called the universal MDL prior, but its genesis shows that it expresses no prior knowledge about the true value of the parameter. See [J.J. Rissanen, *Stochastic Complexity and Statistical Inquiry*, World Scientific, 1989]. Above we have given a validation of MDL from Bayes's rule, which holds irrespective of the assumed prior, provided it is computable and the hypotheses and data are random.

Example 5.4.6 **(Statistics)** In statistical applications, H is some statistical distribution (or model) $H = P(\theta)$ with a list of parameters $\theta = (\theta_1, \ldots, \theta_k)$, where the number k may vary and influence the (descriptional) complexity of θ. (For example, H can be a normal distribution $N(\mu, \sigma)$ described by $\theta = (\mu, \sigma)$.) Each parameter θ_i is truncated to finite precision and encoded with the prefix-code above.

The data sample consists of n outcomes $\mathbf{y} = (y_1, \ldots, y_n)$ of n trials $\mathbf{x} = (x_1, \ldots, x_n)$ for distribution $P(\theta)$. The data sample D in the above formulas is given as $D = (\mathbf{x}, \mathbf{y})$. By expansion of conditional probabilities we have therefore

$$\Pr(D|H) = \Pr(\mathbf{x}, \mathbf{y}|H) = \Pr(\mathbf{x}|H) \cdot \Pr(\mathbf{y}|H, \mathbf{x}).$$

In the argument above we take the negative logarithm of $\Pr(D|H)$, that is,

$$\log \frac{1}{\Pr(D|H)} = \log \frac{1}{\Pr(\mathbf{x}|H)} + \log \frac{1}{\Pr(\mathbf{y}|H, \mathbf{x})}.$$

Taking the negative logarithm in Bayes's rule, and the analysis of the previous section now yields that MDL selects the hypothesis with highest inferred probability satisfying $\mathbf{x}$ is $\Pr(\cdot|H)$-random and $\mathbf{y}$ is $\Pr(\cdot|H, \mathbf{x})$-random. Thus, Bayesian reasoning selects the same hypothesis as MDL does, provided the hypothesis with maximal inferred probability causes $\mathbf{x}, \mathbf{y}$ to satisfy these randomness requirements.

Under certain general conditions, J.J. Rissanen has shown that with k parameters and n data (for large n), Equation 5.11 is minimized for hypotheses H with θ encoded by

$$\log \frac{1}{P(H)} = \frac{k}{2} \log n$$

bits. This is called the *optimum model cost*, since it represents the cost of the hypothesis description at the minimum description length of the total.

As an example, consider a Bernoulli process $(p, 1-p)$ with p close to $\frac{1}{2}$. For such processes, $k = 1$. Let the outcome be $x = x_1 x_2 \ldots x_n$. Set $f_x = \sum_{i=1}^{n} x_i$. For outcome x with $C(x) \geq n - \delta(n)$, the number of 1's will be (by Lemma 2.3 on page 167)

$$f_x = \frac{1}{2} n \pm \sqrt{\frac{3}{2} \frac{\delta(n) + c) n}{\log e}}.$$

With $\delta(n) = \log n$, the fraction of such x's in $\{0,1\}^n$ is at least $1 - 1/n$ and goes to 1 as n rises unboundedly. Hence, for the overwhelming number of x's, the frequency of 1's will be within $1/\sqrt{n}$ of the value $\frac{1}{2}$. That is, to express an estimate of parameter p with high probability it suffices to use a precision of $\frac{1}{2} \log n$ bits. It is easy to generalize this example to arbitrary p. ◇

Example 5.4.7 **(Inferring polynomials)** In biological modeling, we often wish to fit a polynomial f of unknown degree to a set of data points

$$D = (x_1, y_1), \ldots, (x_n, y_n),$$

such that it can predict future data y given x. Even if the data did come from a polynomial curve of degree, say, two, because of measurement errors and noise, we still cannot find a polynomial of degree two fitting all n points exactly. In general, the higher the degree of fitting polynomial, the greater the precision of the fit. For n data points, a polynomial of degree $n-1$ can be made to fit exactly, but probably has no predictive value. Applying ideal MDL we look for $H_{\text{mdl}} := \text{minarg}_H\{K(\mathbf{x}, \mathbf{y}|H) + K(H)\}$.

Let us apply the ideal MDL principle whereby we describe all $(k-1)$-degree polynomials by a vector of k entries, each entry with a precision of d bits. Then the entire polynomial is described by

$$kd + O(\log kd) \text{ bits.} \tag{5.16}$$

(We have to describe k, d, and account for self-delimiting encoding of the separate items.) For example, $ax^2 + bx + c$ is described by (a, b, c) and can be encoded by about $3d$ bits. Each data point (x_i, y_i) that needs to be encoded separately with precision of d bits per coordinate costs about $2d$ bits.

For simplicity assume that probability $\Pr(\mathbf{x}|H)$ equals 1 (because $\mathbf{x}$ is prescribed). To apply the ideal MDL principle we must trade the cost of hypothesis H (Equation 5.16) against the cost of describing $\mathbf{y}$ with the help of H and $\mathbf{x}$. As a trivial example, suppose that $n-1$ out of n data points fit a polynomial of degree 2 exactly, but only 2 points lie on any polynomial of degree 1 (a straight line). Of course, there is a polynomial of degree $n-1$ that fits the data precisely (up to precision). Then the ideal MDL cost is $3d + 2d$ for the second degree polynomial, $2d + (n-2)d$ for the first degree polynomial, and nd for the $(n-1)$th-degree polynomial. Given the choice among those three options, we select the second degree polynomial for all $n > 5$.

A more sophisticated approach, accounting for the average encoding cost of exceptions, assumes that the data are Gaussian distributed. Consider polynomials f of degree at most $n-1$ that minimize the error

$$\text{error}(f) = \sum_{i=1}^{n} (f(x_i) - y_i)^2.$$

In this way, we obtain an optimal set of polynomials for each $k = 1, 2, \ldots, n$. To apply the MDL principle we must balance the cost of hypothesis H (Equation 5.16) against the cost of describing $D|H$.

To describe measuring errors (noise) in data it is common practice to use the normal distribution. In our case this means that each y_i is the outcome of an independent random variable distributed according to the normal distribution with mean $f(x)$ and variance, say, constant. For

each of them we have that the probability of obtaining a measurement y_i, given that $f(x)$ is the true value, is of order $\exp(-(f(x) - y_i)^2)$. Considering this as a value of the universal MDL probability, this is encoded in $s(f(x)-y_i)^2$ bits, where s is a (computable) scaling constant. For all experiments together, we find that the total encoding of $D|f, \mathbf{x}$ takes $s \cdot \text{error}(f)$ bits. The MDL principle thus tells us to choose a k-degree function f_k, $k \in \{0, \ldots, n-1\}$, that minimizes (ignoring the vanishing $O(\log kd)$ term) $kd + s \cdot \text{error}(f_k)$. ◇

Example 5.4.8 **(Inferring Decision Trees)** We apply the MDL principle to infer decision trees. We are given a set of data, possibly with noise, representing a collection of examples. Each example is represented by a data item in the data set, which consists of a tuple of *attributes* followed by a binary *Class* value indicating whether the example with these attributes is a positive or negative example.

The table in Figure 5.3 gives a small sample set. The columns in the table, apart from the last one, describe attributes that are weather conditions. The rows represent weather conditions in relation to a particular unspecified occurrence. The last column classifies the examples as

No.	Attributes				Class
	Outlook	*Temperature*	*Humidity*	*Windy*	
1	overcast	hot	high	not	N
2	overcast	hot	high	very	N
3	overcast	hot	high	medium	N
4	sunny	hot	high	not	P
5	sunny	hot	high	medium	P
6	rain	mild	high	not	N
7	rain	mild	high	medium	N
8	rain	hot	normal	not	P
9	rain	cool	normal	medium	N
10	rain	hot	normal	very	N
11	sunny	cool	normal	very	P
12	sunny	cool	normal	medium	P
13	overcast	mild	high	not	N
14	overcast	mild	high	medium	N
15	overcast	cool	normal	not	P
16	overcast	cool	normal	medium	P
17	rain	mild	normal	not	N
18	rain	mild	normal	medium	N
19	overcast	mild	normal	medium	P
20	overcast	mild	normal	very	P
21	sunny	mild	high	very	P
22	sunny	mild	high	medium	P
23	sunny	hot	normal	not	P
24	rain	mild	high	very	N

FIGURE 5.3. Table of sample data set

positive or negative, where P means that the combination of weather conditions in the row happened and N means that it did not happen. We would like to obtain good predictions for such occurrences by compressing the data. Our task can now be explained as a communication problem between Alice, who has observed the data in Figure 5.3, and Bob. Alice and Bob both know the four parameters (outlook, temperature, humidity, windy) and their attributes. Alice wishes to send Bob the information in the table, using as few bits as possible. Alice and Bob have to agree in advance on an encoding technique to be used.

Alice and Bob do not know in advance which table they have to transmit. The simplest strategy for Alice is to transmit the complete table in Figure 5.3 to Bob literally. There are 24 rows. Each row has four attributes and one Class value. Three attributes have three alternative values each; the other attribute and Class have two alternative values each. Then this requires $24(3\log_2 3 + 2) = 24(3 \times 1.585 + 2) \approx 162.12$ bits. Or Alice can agree with Bob beforehand about a fixed order of enumerating all $3 \times 3 \times 2 \times 3 = 54$ combinations of attributes, and then just send the last column of 54 bits, supplying arbitrary Class values for the 30 rows missing from the table in Figure 5.3. These methods use no data compression.

If Alice is clever enough to find some regularity in the data, such as 'the Class value is N iff it rains,' then Alice needs only a few bits to transmit this sentence to Bob, and Bob can use this rule to reconstruct the complete table with all the combinations of attributes with $3 \times 3 \times 2 \times 3 = 54$ rows.

Let us say that Alice and Bob have agreed to use a decision tree. A decision tree that is consistent with the data can be viewed as a classification procedure. The internal nodes in the tree are *decision nodes*. Each such node specifies a test of a particular attribute; the possible answers are labeled on the arcs leaving the decision node. A leaf of the tree specifies a class to which the object that passed the attribute tests and arrived at this leaf belongs.

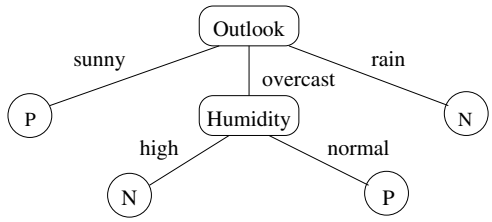


FIGURE 5.4. Imperfect decision tree

Given data as in Figure 5.3, we can construct many different decision trees of various sizes that agree with the data. Two such trees are given in Figures 5.4 and 5.5. The tree in Figure 5.4 is imperfect, since it makes an error in row 8; the tree in Figure 5.5 classifies all of the sample set correctly. The tree in Figure 5.5 is the smallest perfect decision tree for the data in Figure 5.3.

Some data, for example noise, may not obey the predicting rule defined by the decision tree. One usually has a choice between using a small *imperfect* tree that classifies some data falsely or a big *perfect* tree that correctly classifies all given data. Alice can use a smaller imperfect tree or the bigger perfect tree. The tree in Figure 5.5 grows much bigger just because of a single (perhaps noisy) example (row 8), and Alice may find that it is more economical to code it separately, as an *exception*.

The goal is often to construct a decision tree that has the smallest error rate for classifying unknown future data. Is the *smallest perfect decision tree* really a good predictor? It turns out that in *practice* this is not the case. Due to the presence of noise or inadequacy of the given attributes, selecting a perfect decision tree overfits the data and gives generally poor predictions. Many ad hoc rules have been suggested and used for overcoming this problem.

The MDL principle appears to provide a solution and generally works well in practice. Essentially, given the data sample *without* the class values, we look for a reasonably small tree such that most data are correctly classified by the tree. We encode the inconsistent data as exceptions. We minimize the sum of

- the number of bits to encode a (not necessarily perfect) decision tree T, that is, the model that gives the $\log 1/P(H)$ term; and

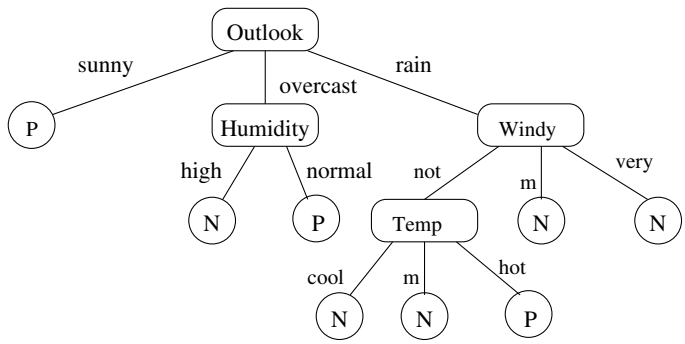


FIGURE 5.5. Perfect decision tree

- the number of bits to encode (describe) the examples (x_i, y_i) in the sample $(\mathbf{x}, \mathbf{y})$ that are inconsistent with T, given the entire data sample $\mathbf{x}$ without the class values $\mathbf{y}$. This gives the $\log 1/\Pr(\mathbf{y}|H, \mathbf{x})$ term.

We have to provide a coding method. This is important in applications, since it determines where the optimum is. If the encoding of trees is not efficient, then we may end up with a very small tree (with relatively large depth), and too many examples become exceptions. An inefficient encoding of the exceptions would result in overly large trees. In both cases, the prediction of unseen data is affected. The reader should realize that the choice of cost measure and encoding technique cannot be objective. One can encode a tree by making a depth-first traversal. At each internal node, write down the attribute name in some self-delimiting form followed by its edge label. At a leaf write down the class value. If the tree is not perfect, then the data that do not fit in the tree are encoded separately as exceptions (in some economical way using the provided total data sample without the class values).

Coding the Tree. It is desirable that the smaller trees be represented by shorter encodings. Alice can make a depth-first traversal of the tree in Figure 5.4, and accordingly she writes down

1 Outlook 0 P 1 Humidity 0 N 0 P 0 N.

For the tree in Figure 5.5, she writes down

1 Outlook 0 P 1 Humidity 0 N 0 P 1 Windy 0 N 0 N
1 Temperature 0 N 0 N 1 P.

Alice uses a 1 to indicate that the next node in the depth-first search is an internal node and then writes the corresponding attribute; Alice writes a 0 followed by N (P) if she meets a leaf with value N (P). Representing N or P requires only one bit. Representing attribute 'Outlook' at the root level requires 2 bits, since there are 4 possibilities. Encoding the next-level attributes requires $\log 3$ bits, since there are only 3 choices left ('Outlook' is already used). She can use just one bit for 'Windy' and one bit for 'Temperature' (in fact, this one bit is not needed). Thus, the smaller (but imperfect) tree requires 13.585 bits; the bigger (but perfect) tree requires 25.585 bits.

Coding the Exceptions. Since the decision tree in Figure 5.4 is not perfect, we need to indicate where the exceptions are. In this case there is a single exception. The most straightforward way is to indicate its *position* among all 54 possible combinations of attributes. This costs $\log 54 \approx 5.75$ extra bits.

Thus, the encoding using the decision tree in Figure 5.4 uses 19.335 bits; the encoding using the decision tree in Figure 5.5 uses 25.585 bits. The MDL principle tells us to use the former method, which is also much shorter than the 54-bit plain encoding.

Procedures for computing decision trees have been implemented by J.R. Quinlan and R. Rivest [*Inform. Computation*, 80(1989), 227–248]. Computing the absolute minimum decision tree is NP-hard, as shown by T. Hancock, T. Jiang, M. Li, and J.T. Tromp, *Inform. Comput.*, 126:2(1996), 114–122. They have shown that approximating minimum decision trees is also NP-hard, even approximation to within some polynomial factor. Consequently, approximation heuristics have to be used. See also K. Yamanishi, A Randomized Approximation of the MDL for Stochastic Models with Hidden Variables, Proc. 9th ACM Comput. Learning Conference, ACM Press, 1996; and V. Vovk, *J. Comput. System Sci.*, 55:1(1997), 96-104.

◇

Example 5.4.9 **(Exception-based MDL)** In the above interpretation of MDL we essentially look for a hypothesis H minimizing $K(D|H) + K(H)$. This always satisfies

$$K(D|H) + K(H) \geq K(D).$$

Define $E := D - D_H$, where D_H is the data set classified according to H. In our new approach, called *E-MDL*, we want to minimize

$$K(H, E) \approx K(H) + K(E|H)$$

over H. That is, E denotes the subset of the data sample D that are *exceptions* to H in the sense of being not covered by H. We want to find H such that the sum of the descriptions of H and the *exception list* E are minimized. Note that in this case, we always have

$$\min_H\{K(H) + K(E|H)\} \leq K(\varnothing) + K(D) = K(D),$$

in contrast to the standard interpretation of MDL above. This incarnation of MDL is not straightforwardly derived by our approach above. We may interpret it that we look for the shortest description of an accepting program for the data consisting of a classification rule H and an exception list E. While this principle often gives good results, application may lead to absurdity, as the following shows:

In many problems the data sample consists of positive examples only, for example, in learning (a grammar for) the English language, given the *Oxford English Dictionary*. According to E-MDL, the best hypothesis is the trivial grammar H_0 generating *all* sentences over the alphabet. This

grammar gives $K(H_0) = O(1)$ independent of D and also $E_0 := \varnothing$. This choice leads to

$$K(H_0) + K(E_0|H_0) = K(H_0) + O(1) = O(1),$$

which is true but absurd. The E-MDL principle is vindicated and reduces to the standard one in the context of interpreting $D = (\mathbf{x}, \mathbf{y})$ as in Example 5.4.6 on page 390, with $\mathbf{x}$ fixed as in supervised learning. This is a correct application as in Example 5.4.8 on page 393. Now for constant $K(\mathbf{x}|H)$, the hypothesis $H_{\mathrm{e-mdl}}$ of minimal complexity among the hypotheses that minimize

$$K(H) + K(\mathbf{y}|H, \mathbf{x}) + K(\mathbf{x}|H)$$

is the same as the hypothesis H_{mdl} of minimal complexity among the hypotheses that minimize

$$K(H) + K(\mathbf{y}|H, \mathbf{x}).$$

Ignoring the constant $\mathbf{x}$ in the conditional, $K(\mathbf{y}|H, \mathbf{x})$ corresponds to $K(E|H)$. ◇

Example 5.4.10 **(Maximum likelihood)** The *maximum likelihood* (ML) principle says that for given data D, one should select the hypothesis H that maximizes $\Pr(D|H)$ or equivalently, minimizes $\log 1/\Pr(D|H)$. In case of finitely many hypotheses, this is a special case of the Bayes's rule with the hypotheses distributed uniformly (all have equal probability). It is also a special case of MDL. The principle has many admirers, is supposedly objective, and is due to R.A. Fisher. ◇

Example 5.4.11 **(Maximum entropy)** In statistics there are a number of important applications for which the ML principle fails but the maximum entropy principle has been successful, and conversely.

In order to apply Bayes's rule, we need to decide what the prior probabilities $p_i = P(H_i)$ are, subject to the constraint $\sum_i p_i = 1$ and certain other constraints provided by empirical data or considerations of symmetry, probabilistic laws, and so on. Usually these constraints are not sufficient to determine the p_i's uniquely. E.T. Jaynes proposed to select the prior by the *maximum entropy* (ME) principle.

The ME principle selects the estimated values $\hat{p}_i$ that maximize the entropy function

$$H(p_1, \ldots, p_k) = \sum_{i=1}^{k} p_i \ln \frac{1}{p_i}, \tag{5.17}$$

subject to

$$\sum_{i=1}^{k} p_i = 1 \tag{5.18}$$

and some other constraints. For example, consider a loaded die, $k = 6$. If we do not have any information about the die, then using the principle of indifference, we may assume that $p_i = \frac{1}{6}$ for $i = 1, \ldots, 6$. This actually coincides with the ME principle, since $H(p_1, \ldots, p_6) = \sum_{i=1}^{6} p_i \ln 1/p_i$, constrained by Equation 5.18, achieves its maximum $\ln 6 = 1.7917595$ for $p_i = \frac{1}{6}$ for all i.

Now suppose it has been experimentally observed that the die is biased and the average throw gives 4.5, that is,

$$\sum_{i=1}^{6} i p_i = 4.5. \tag{5.19}$$

Maximizing the expression in Equation 5.17, subject to the constraints in Equations 5.18 and 5.19, gives the estimates

$$\hat{p}_i = e^{-\lambda i}(\sum_j e^{-\lambda j})^{-1}, \quad i = 1, \ldots, 6,$$

where $\lambda = -0.37105$. Hence,

$$(\hat{p}_1, \ldots, \hat{p}_6) = (0.0543, 0.0788, 0, 1142, 0.1654, 0.2398, 0.3475).$$

The maximized entropy $H(\hat{p}_1, \ldots, \hat{p}_6)$ equals 1.61358. How dependable is the ME principle? Jaynes has proven an entropy concentration theorem that, for example, implies the following: In an experiment of $n = 1000$ trials, 99.99% of all 6^{1000} possible outcomes satisfying the constraints of Equations 5.19 and 5.18 have entropy

$$1.602 \leq H\left(\frac{n_1}{n}, \ldots, \frac{n_6}{n}\right) \leq 1.614,$$

where n_i is the number of times the value i occurs in the experiment. We show that the maximum entropy principle can be considered as a special case of the MDL principle, as follows:

Consider the same type of problem. Let $\theta = (p_1, \ldots, p_k)$ be the prior probability distribution of a random variable. We perform a sequence of n independent trials. Shannon has observed that the real substance of Formula 5.17 is that we need approximately $nH(\theta)$ bits to record a sequence of n outcomes. Namely, it suffices to state that each outcome appeared $n_1, \ldots, n_k$ times, respectively, and afterward give the index of

which one of the possible sequences D of n outcomes actually took place. The number of possible sequences is given by the multinomial coefficient

$$\binom{n}{n_1, \ldots, n_k} = \frac{n!}{n_1! \cdots n_k!},$$

as in Exercise 1.3.4 on page 10. For this, no more than

$$k \log n + \log \binom{n}{n_1, \ldots, n_k} + O(\log \log n) \tag{5.20}$$

bits are needed. The first term corresponds to $-\log P(\theta)$, the second term corresponds to $-\log \Pr(D|\theta)$, and the third term represents the cost of encoding separators between the individual items. Using Stirling's approximation of $n! \sim \sqrt{2\pi n}(n/e)^n$ to evaluate the displayed multinomial coefficient, we find that for large n, Equation 5.20 is approximately

$$n \sum_{i=1}^{k} \frac{n_i}{n} \log \frac{n}{n_i} = nH\left(\frac{n_1}{n}, \ldots, \frac{n_k}{n}\right).$$

Since k and n are fixed, the least upper bound on the minimum description length, for an arbitrary sequence of n outcomes under constraints such as those in Equations 5.18 and 5.19, is found by maximizing the displayed multinomial coefficient subject to said constraints. This is equivalent to maximizing the entropy function in Equation 5.17 under the constraints.

Viewed differently, let S_θ be the set of outcomes with values $(n_1, \ldots, n_k)$, with $n_i = np_i$, corresponding to a distribution $\theta = (p_1, \ldots, p_k)$. Then due to the small number of values (k) in θ relative to the size of the sets, we have

$$\log \sum_{\theta} d(S_\theta) \approx \max_{\theta} \{\log d(S_\theta)\}.$$

The left-hand side is the minimum description length; the right-hand side is the maximum entropy. ◇

Exercises

5.4.1. [20] Let $R(n)$ denote the fraction of binary strings of length n that are sufficiently $\Pr(\cdot|H)$-random as in Equation 5.14. Show that $R(n) = 1 - O(1/2^{K(H,n)})$ and goes to 1 for $n \to \infty$. Moreover, $\limsup_{n\to\infty} R(n) = 1 - O(1/n)$.

Comments. Source for this and the next three exercises: P.M.B. Vitányi and M. Li, *IEEE Trans. Inform. Theory*, 46:2(2000), 446–464.

5.4.2. [27] Show that the probability that for data of binary length n, the hypotheses of binary length m that are selected by the Bayesian maximum a posteriori and minimum-description-length principles, respectively, are close in the sense of satisfying the relations of Theorem 5.4.1. Show that this probabilty goes to one as m and n grow unboundedly. Moreover, the lim sup of that probability exceeds $1 - O(1/\min\{m, n\})$.

5.4.3. [23] Show that if $\log 1/\Pr(D|H) + \log 1/P(H) = K(D|H) + K(H) + O(1)$, then H is P-random up to $K(\Pr(\cdot|H)) - K(P) + O(1)$, and D is $\Pr(\cdot|H)$-random up to $K(P) - K(\Pr(\cdot|H)) + O(1)$. (Negative randomness deficiencies correspond to 0.)

5.4.4. [26] Let $\alpha(P, H)$ in Theorem 5.4.1 be small (for example, a constant) and prior $P := \mathbf{m}$. Show that Theorem 5.4.1 is satisfied iff the data sample D is $\Pr(\cdot|H_{\rm mdl})$-random. Show also that this has probability going to one for the binary length n of the data increasing unboundedly (and the lim sup of the probability exceeds $1 - O(1/n)$).

Comments. The choice of $\mathbf{m}$ as prior satisfies the condition of Theorem 5.4.1. This prior is an objective and recursively invariant form of Occam's razor: a simple hypothesis H (with $K(H) \ll l(H)$) has high $\mathbf{m}$-probability, and a complex or random hypothesis H (with $K(H) \approx l(H)$) has low $\mathbf{m}$-probability $2^{-l(H)}$. Note that all hypotheses are random with respect to the distribution $\mathbf{m}$. The exercise shows that ideal MDL corresponds to Bayes's rule with the universal prior distribution $\mathbf{m}(\cdot)$ and contemplated hypotheses H for which the data D are $\Pr(\cdot|H)$-random. Thus, ideal MDL selects the simplest hypothesis H that balances $K(D|H)$ and $K(H)$ and such that the data sample D is random to it. Hint: use the coding theorem, Theorem 4.3.3 on page 273, to show that $\log 1/\Pr(D|H) = K(D|H) + O(1)$, and so determine the admissible hypotheses for the data D. In particular, the $H := H_{\rm mdl}$ minimizing $K(D|H) + K(H)$ is admissible iff D is $\Pr(\cdot|H)$-random. This happens with probability going to one by Exercise 5.4.1.

5.5 Nonprobabilistic Statistics

As one of the last scientific acts in a long and brilliant career, Kolmogorov in 1974 suggested a reformulation of statistical science independent of probabilistic assumptions. This leads to model selection whereby the performance is related to the individual data sample and the individual model selected. Moreover, it gives a nonprobabilistic foundation for model selection and prediction, founded on the firm and uncontentious ground of finite combinatorics and effective computation. As in Section 1.11.5 on probabilistic statistics, we can choose a model from a set of contemplated models only (which may or may not contain the best possible model). Intuitively, our selection criteria are that (i) the data should be a typical element of the model selected, and (ii) the selected

model should have a simple description. We need to make the meaning of 'typical' and 'simple' rigorous and balance the requirements (i) and (ii). The resulting theory presents a new viewpoint on the foundations of maximum likelihood and minimum description length.

5.5.1 Algorithmic Sufficient Statistic

It is very difficult, if not impossible, to formalize the goodness of fit of an individual model for individual data in the classic probabilistic statistics setting. It is as hard to express the practically important issues in induction in those terms, which is no doubt one of the reasons why contention is rampant in that area. In the algorithmic setting a statistic is defined as a data-containing finite set.

Definition 5.5.1 Let $S = \{x_1, \ldots, x_n\}$ be a finite set of strings. Let U be the reference universal prefix machine U, and let $U(p) = \langle x_1, \langle x_2, \ldots, \langle x_{n-1}, x_n\rangle \ldots\rangle\rangle$, that is, U with binary program p computes the lexicographic listing of the elements of S and then halts. The *prefix complexity of the finite set* S, denoted by $K(S)$, is the length of the shortest such program p. We define $S^* = p$, or, if there is more than one such shortest program, then the first one that halts in a standard dovetailed running of all programs. Similarly, $K(x|S)$ denotes the length of the shortest self-delimiting binary program that computes x from the lexicographic listing of all the elements of S.

Definition 5.5.2 A finite set S containing x is an *algorithmic statistic* for x.

One can describe every string x, the *data*, by a *two part description*: the *model description*, or algorithmic statistic for x, in the form of a finite set S of strings containing x, and the *data-to-model code* describing x given S.

Definition 5.5.3 A string x is a *typical* or *random* element of a finite set S, and S is a *fitting model* for x, if $x \in S$ and the randomness deficiency $\delta(x|S)$ defined by $\log d(S) - K(x|S)$ is $O(1)$. Here $O(1)$ is a constant (or possibly a small value) independent of x and S.

Example 5.5.1 **(Randomness deficiency and model fitness)** (i) Randomness deficiency is almost nonnegative, that is, $\delta(x|S) \geq c$ for some, possibly negative, constant c and every $x \in S$. This is easy to see, since every $x \in S$ can be described by its $\log d(S)$-bit index in S once we are given S. Thus $K(x|S) \leq \log d(S) + O(1)$.
(ii) For every S, the randomness deficiency of almost all elements of S is small: The number of $x \in S$ with $\delta(x|S) > \beta$ is fewer than $d(S)2^{-\beta}$. Indeed, $\delta(x|S) > \beta$ implies $K(x|S) < \log d(S) - \beta$. Since there are at most $2^{\log d(S)-\beta}$ programs of fewer than $\log d(S) - \beta$ bits, the number of x's satisfying the inequality cannot be larger.

(iii) Every element with small randomness deficiency in S possesses every property possessed by a majority of elements in S (we identify a property of elements of S with a subset of S consisting of all elements having the property). More specifically, assume that A is a subset of S with $d(A) \geq (1 - 2^{-\beta})d(S)$ and $K(A|S) \leq \gamma$. Then the randomness deficiency of all $x \notin A$ in S satisfies $\delta(x|S) > \beta - \gamma - O(\log \log d(A))$, which is large if β is large and γ is small.

The randomness deficiency measures our disbelief that x can be obtained by random sampling in S (where all elements of S are equiprobable). By property (ii), the randomness deficiency of an element randomly chosen in S is small, with high probability. On the other hand, if $\delta(x|S)$ is small, then there is no way to refute the hypothesis that x was obtained by random sampling in S: every such refutation is based on a simply described property possessed by a majority of S but not by x. Here it is important that we consider only simply described properties, since otherwise we can refute the hypothesis by exhibiting the property $A = S - \{x\}$.

Consider the set S_1 of binary strings of length n whose every odd position is 0. Let x be an element of this set in which the subsequence of bits in even positions is an incompressible string. Then, x is a typical element of S_1. As another example, the string x is a typical element of the set $S_2 = \{x\}$. ◇

Definition 5.5.4 A finite set S containing x is called an *algorithmic sufficient statistic* for x if $K(S) + \log d(S) = K(x) + O(1)$. Here $O(1)$ is a constant (or indeed a small value) independent of x and S.

Intuitively, a statistic expresses the essence of the data. It is sufficient if the two-part code describing the data consisting of the statistic and the data-to-model code is as concise as the best one-part description. Compare the algorithmic sufficient statistic of Definition 5.5.4 with the probabilistic sufficient statistic of Definition 1.11.6 on page 83.

Lemma 5.5.1 *If S is an algorithmic sufficient statistic for x, then x is a random, typical element of S.*

Proof.

$$\begin{aligned} K(x) \leq K(x, S) &\leq K(S) + K(x|S) + O(1) \\ &\leq K(S) + \log d(S) + O(1) \leq K(x) + O(1), \end{aligned}$$

where only the substitution of $K(x|S)$ by $\log d(S) + O(1)$ uses the fact that x is an element of S. □

Example 5.5.2 Sets for which x is typical form a much wider class than algorithmic sufficient statistics for x: If x is a simple string of length n, say $K(x) = \frac{1}{4}n$, then x is still typical for $S_3 = \{x, y\}$ irrespective of y. But with $K(y) \gg K(x)$, the set S_3 will be too complex to be sufficient for x. For a perhaps less artificial example, consider a random string y of length n. Let S_y be the set of strings of length n that have 0's exactly where y has, and can have 0's or 1's where y has 1's. Let x be a random element of S_y. Then x has about one-quarter 1's, so its complexity is much less than n. The set S_y is typical with respect to x, but is too complex to be sufficient, since its complexity is about n. ◇

5.5.2 Structure Functions

The first parameter we consider in model selection is the *simplicity* $K(S)$ of the model S explaining the data x. The second parameter is *how typical* the data is with respect to S, expressed by the randomness deficiency $\delta(x|S) = \log d(S) - K(x|S)$. The third parameter is how *short the two-part code* $\Lambda(S) = K(S) + \log d(S)$ of the data sample x, using model S, is. These parameters induce a partial order on the contemplated set of models. We write $S_0 \leq S_1$ if S_0 scores equal or less than S_1 in all three parameters. If this is the case, then we may say that S_0 is at least as good as S_1 as an explanation for x. The converse need not necessarily hold, since it is possible that S_0 is at least as good a model for x as S_1 without scoring better than S_1 in all three parameters simultaneously.

The algorithmic statistical properties of a data sample x are fully represented by the set

$$A_x = \{(K(S), \delta(x|S), \Lambda(S))\}, \tag{5.21}$$

such that $S \ni x$, together with a componentwise order relation on the elements of those triples. The complete characterization of this set is given in Example 5.5.11 on page 413.

To do the analysis, for every fixed i, we consider selection of a model S for x with $K(S) \leq i$ in three ways, characterized by three different functions. In information theory this i is called the 'rate.'

Best-Fit Estimator

With the first method we select a model S containing x, with $K(S) \leq i$, for which the data x has least randomness deficiency $\delta(x|S)$.

Definition 5.5.5 The *minimal randomness deficiency* function, also called the *best-fit estimator*, is defined by (setting $\min \varnothing = \infty$)

$$\beta_x(i) = \min_S \{\delta(x|S) : S \ni x,\ K(S) \leq i\}.$$

This implies that a witness model of $\beta_x(i)$ is a model of best fit for x among the models of complexity at most i, Example 5.5.1 on page 402.

This gives us a proper model for x at every model complexity. Unfortunately, it will turn out that the method is too difficult to apply. But we shall show that we can obtain our goal in a round about manner.

Example 5.5.3 **(Lossy compression)** The function $\beta_x(i)$ is relevant to lossy compression, used, for instance, to compress images. Assume we need to compress x to i bits where $i \ll K(x)$. Of course, this implies some loss of information present in x. One way to select redundant information to discard is as follows: Instead of x use a set $S \ni x$ with $K(S) \leq i$ and with small $\delta(x|S)$ in the form of a compressed version S' ($l(S') \leq i$) of S. To reconstruct an x', a decompresser uncompresses S' to S and selects at random an element x' of S. Since with high probability the randomness deficiency of x' in S is small, x' serves the purpose of the message x as well as x itself does. Let us look at an example. To transmit a picture of 'rain' through a channel with limited capacity i, one can transmit the indication that this is a picture of the rain, and the particular drops may be chosen by the receiver at random. In this interpretation, $\beta_x(i)$ indicates how random or typical x is with respect to the best model at complexity level i, and hence how indistinguishable from the original x the randomly reconstructed x' can be expected to be. ◇

Maximum-Likelihood Estimator

A.N. Kolmogorov, at a conference in Tallinn, Estonia, in 1973 (no written version), and in a talk at the Moscow Mathematical Society in the next year, proposed model selection guided by the following function. We select a model S containing data x, with $K(S) \leq i$, that minimizes the maximal data-to-model code length $\log d(S)$.

Definition 5.5.6 The Kolmogorov *structure function*, also called the *maximum likelihood (ML) estimator*, h_x of given data x is defined by

$$h_x(i) = \min_S \{\log d(S) : S \ni x,\ K(S) \leq i\},$$

where $S \ni x$ is a contemplated model for x, and i is a nonnegative integer value bounding the complexity of the contemplated S's.

Every finite set $S \ni x$ is equivalent to a probability mass function P defined by $P(y) = 1/d(S)$ for $y \in S$, and $P(y) = 0$ otherwise. Let $\mathcal{P}$ be the family of these probability mass functions. We can rewrite $h_x(i) = \min_P \{\log 1/P(x) : P(x) > 0,\ K(P) \leq i\}$, where the minimum is taken over the elements of $\mathcal{P}$. The minimum of $\log 1/P(x)$ is reached for the $P \in \mathcal{P}$ that maximizes $P(x)$, whence the name 'ML-estimator' for $h_x(i)$. See Exercise 5.5.19 on page 428 for the generalization of the family $\mathcal{P}$ to the model class of computable probability mass functions (probability models).

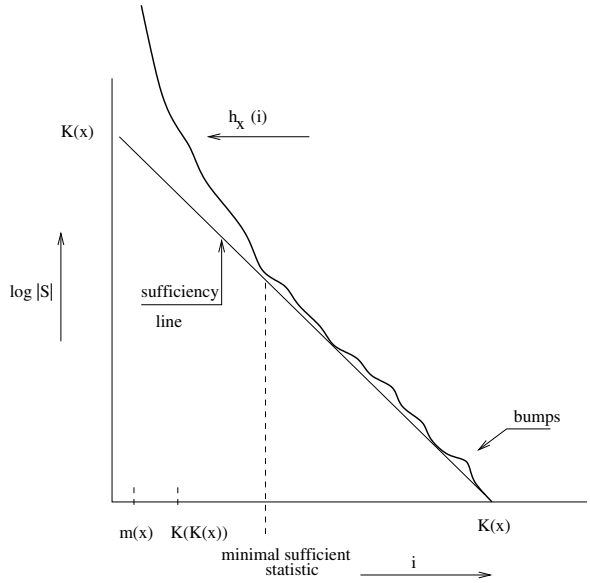


FIGURE 5.6. Kolmogorov's structure function (ML estimator)

Example 5.5.4 **(Sufficiency line)** Clearly, the function $h_x(i)$ is nonincreasing and reaches $\log d(\{x\}) = 0$ for $i = K(x) + c_1$, where c_1 is the number of bits required to change x into $\{x\}$; see Figure 5.6. For every $S \ni x$ we can describe x by giving S and the index of x in S, which shows that $K(x) \leq K(S) + \log d(S) + O(1)$. Therefore,

$$K(x) \leq i + h_x(i) + O(1),$$

that is, the function $h_x(i)$ never decreases more than a fixed independent constant below the diagonal *sufficiency line* L defined by $L_x(i) = K(x) - i$. The sufficiency line is therefore a lower bound on $h_x(i)$ and is approached to within a constant distance by the graph of h_x for certain i's (for instance, for $i = K(x) + c_1$). For these i's we have $i + h_x(i) = K(x) + O(1)$; and a model corresponding to such an i (witness for $h_x(i)$) is a sufficient statistic as in Definition 5.5.4 on page 403. It is *minimal* for the least such i; see Example 5.5.5 and Section 5.5.9. ◇

Example 5.5.5 **(Minimal sufficient statistic)** The *minimal algorithmic sufficient statistic* for data x is a finite set S_0 witnessing the least i for which $\beta_x(i) = 0$ (equivalently, $h_x(i) = K(x) - i$). Every nonminimal sufficient statistic, say associated with witness set S, with $K(S) > K(S_0)$, in addition

models $K(S) - K(S_0)$ bits of randomness in the data. Thus, the higher the complexity of the sufficient statistic, the more additional noise in the data it models. This is called overfitting. At lower complexity levels $i < K(S_0)$, the structure function has not yet descended to the sufficiency line $L_x(i) = K(x) - i$. We know from Exercise 5.5.1 on page 420 that the complexity of a minimal sufficient statistic is at least $K(K(x))$, up to a fixed additive constant, for every x. Hence, for smaller arguments the structure function definitively rises above the sufficiency line. Continued in Example 5.5.10 on page 413. ◇

MDL Estimator

We select a model S such that the total two-part description length, consisting of one part describing S containing x and the second part describing the maximal data-to-model code of a string in S, is minimized.

Definition 5.5.7 The length of the minimal two-part code for x using a model S containing x, with $K(S) \leq i$, consisting of the model cost $K(S)$ and the maximal length $\log d(S)$ of the index of an element of S, is given by the *MDL function*, also called the *minimum description length* (*MDL*) *estimator*,

$$\lambda_x(i) = \min_S \{\Lambda(S) : S \ni x,\ K(S) \leq i\},$$

where $\Lambda(S) = \log d(S) + K(S) \geq K(x) - O(1)$ is the total length of two-part code of x with help of model S.

Relations Between the Structure Functions

First we show that properties of λ_x translate directly into properties of h_x, since $h_x(i)$ is always close to $\lambda_x(i) - i$. The functions $h_x(i) + i$ (the ML code length plus the model complexity) and $\lambda_x(i)$ (the MDL code length) are essentially the same function. This is not trivial, since in the first case we minimize the first term of the sum of two terms, and in the second case we minimize the sum of two terms.

Definition 5.5.8 Let $f, g, \Delta : \mathcal{N} \to \mathcal{N}$ be three integer functions. Then $f(i)$ follows the *shape* of $g(i)$ up to error $\Delta(i)$ if the graph of $f(i)$ is situated in a strip of width $2\Delta(i) + O(1)$ centered on the graph of $g(i)$, as in Figure 5.8 on page 411.

Lemma 5.5.2 *The MDL estimator $\lambda_x(i)$ follows the shape of $h_x(i) + i$ up to error $K(i)$. Formally, for every x and i we have*

$$\lambda_x(i) \leq h_x(i) + i \leq \lambda_x(i) + K(i) + O(1).$$

Proof. The inequality $\lambda_x(i) \leq h_x(i) + i$ is immediate. So it suffices to prove that $h_x(i) + i \leq \lambda_x(i) + K(i) + O(1)$. The proof of this inequality

uses an explicit construction of a set $S \ni x$ witnessing $h_x(i)$ that also witnesses $\lambda_x(i)$ up to an additive $K(i)$ term. It is left to the reader as Exercise 5.5.3 on page 421. □

Example 5.5.6 The proof of the lemma implies that the *same model* witnessing $h_x(i)$ also witnesses $\lambda_x(i)$ up to an additive term of $K(i)$. The converse is true only for the smallest cardinality set witnessing $\lambda_x(i)$. Without this restriction, a counterexample is the following: For x of length n with $K(x) \geq n$, the set $S = \{0,1\}^n$ witnesses $\lambda_x(\frac{1}{2}n) = n + O(K(n))$ but does not witness $h_x(\frac{1}{2}n) = \frac{1}{2}n + O(K(n))$. (If $\lambda_x(i) = K(x)$, then every set of complexity $i' \leq i$ witnessing $\lambda_x(i') = K(x)$ also witnesses $\lambda_x(i) = K(x)$.) ◇

The central result of this section tells us that a model achieving the MDL code length $\lambda_x(i)$, or the ML code length $h_x(i)$, also achieves the best possible fit $\beta_x(i)$, up to ignorable error:

Theorem 5.5.1 *The graph of $\lambda_x(i)$ follows the shape of the graph of $\beta_x + K(x)$ up to error $\Delta = O(\log n)$. Formally, for every i, with $0 \leq i \leq K(x) - O(\log n)$, there is a $\Delta = O(\log n)$ such that*

$$\beta_x(i) + K(x) - \lambda_x(i) \leq O(1),$$
$$0 \leq \beta_x(i) + K(x) - \lambda_x(i + \Delta).$$

Proof. Firstly, note that both functions $\lambda_x(i)$ and $\beta_x(i)$ are by definition nonincreasing in i. Therefore, the theorem follows from the fact that the graph of λ_x tracks the graph of $\beta_x + K(x)$, in the following sense: for every x of length n, and complexity $K(x)$, we have

$$\beta_x(i) + K(x) \leq \lambda_x(i) + O(1), \tag{5.22}$$
$$\lambda_x(i + O(\log n)) \leq \beta_x(i) + K(x) + O(\log n), \tag{5.23}$$

with $0 \leq i \leq K(x)$ in Equation 5.22, and $0 \leq i \leq K(x)$ in Equation 5.23. To prove these inequalities, it is convenient to first rewrite the formula for the randomness deficiency $\delta(x|A)$ using the symmetry of information, Theorem 3.9.2 on page 249, as follows:

$$\begin{aligned} \delta(x|A) &\leq \log d(A) + K(A) - K(A|x^*) - K(x) + O(1) \\ &= \Lambda(A) - K(A|x^*) - K(x) + O(1). \end{aligned} \tag{5.24}$$

Proof of Equation 5.22. This is easy, since for every set $S \ni x$ witnessing $\lambda_x(i)$ we have by Equation 5.24 that $\delta(x|S) \leq \Lambda(S) - K(x) + O(1) = \lambda_x(i) - K(x) + O(1)$. Since $\beta_x(i) \leq \delta(x|S)$ by definition, we are done.

Proof of Equation 5.23. This is more difficult. By Equation 5.24, and the obvious $K(A|x^*) \leq K(A|x) + O(1)$, it suffices to prove that for every $A \ni x$ there is an $S \ni x$ with

$$K(S) \leq K(A) + O(\log \Lambda(A)),$$
$$\log d(S) \leq \log d(A) - K(A|x) + O(\log \Lambda(A)).$$

To see this, note that for every set A witnessing $\beta_x(i)$, the set S will witness

$$\lambda_x(i + O(\log n)) \leq \beta_x(i) + K(x) + O(\log n),$$

since $\Lambda(A) = \log d(A) + K(A) = K(x|A^*) + \beta_x(i) + K(A) \leq 3n + O(\log n)$, provided $i \geq K(n) + O(1)$. For $i < K(n) + O(1)$ we use Equation 5.23 with $i = i' + 2\log n$. The proof obligation is implied by the stronger assertion of Exercise 5.5.4 on page 421, which is left to the reader. □

Example 5.5.7 **(Witness sets)** The bare statement of Theorem 5.5.1 does not preclude that $\beta_x(i)$, $\lambda_x(i)$, and $h_x(i)$ are witnessed by three different sets. But in the proof of the theorem we see that for every finite set $S \ni x$, of complexity at most $i + O(\log n)$ and minimizing $\Lambda(S)$, we have $\delta(x|S) \leq \beta_x(i) + O(\log n)$. Ignoring $O(\log n)$ terms, this means that the same set that witnesses $\lambda_x(i)$ also witnesses $\beta_x(i)$. By Example 5.5.6 on page 408, we know that every set that witnesses $h_x(i)$ also witnesses $\lambda_x(i)$. Hence,

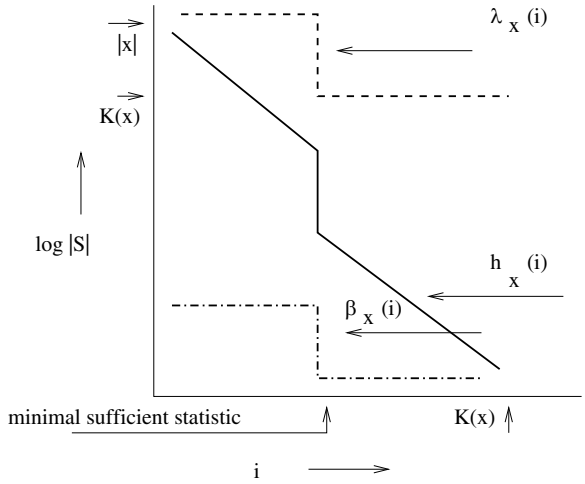


FIGURE 5.7. Relations among the structure functions

at every complexity level i, every best model with respect to $h_x(i)$ is also a best one with respect to $\lambda_x(i)$ and also a best one with respect to $\beta_x(i)$, that is, we have typicality. This explains why it is worthwhile to find shortest two-part descriptions for given data x: This is the single known way to find in an effective manner an $S \ni x$ $(K(S) \leq i)$ with respect to which x is as typical as possible. Since by Exercise 5.5.16 on page 426 the set

$$\{(x, S, \beta) : x \in S,\ \delta(x|S) < \beta\}$$

is not recursively enumerable, we are not able to generate such best-fit S's directly. ◇

Corollary 5.5.1 A model achieving the MDL code length $\lambda_x(i)$, or the ML code length $h_x(i)$, essentially achieves the best possible fit $\beta_x(i)$.

Example 5.5.8 **(Witness sets, continued)** Witness models for the best-fit estimator may not be witness models for the ML estimator or the MDL estimator. That is, not every finite set witnessing $\beta_x(i)$ also witnesses $\lambda_x(i)$ or $h_x(i)$. For example, let x be a string of length n with $K(x) \geq n$. Let $S_1 = \{0,1\}^n \cup \{y\}$, where y is a string of length $\frac{1}{2}n$ such that $K(x, y) \geq \frac{3}{2}n$, and let $S_2 = \{0,1\}^n$. Then both S_1, S_2 witness $\beta_x(\frac{1}{2}n + O(\log n)) = O(1)$, but

$$\Lambda(S_1) = \frac{3}{2}n + O(\log n) \gg \lambda_x\left(\frac{1}{2}n + O(\log n)\right) = n + O(\log n),$$

while

$$\log d(S_2) = n \gg h_x\left(\frac{1}{2}n + O(\log n)\right) = \frac{1}{2}n + O(\log n).$$

However, for every i such that $\lambda_x(i)$ decreases when $i \to \alpha$ with $i \leq \alpha$, a witness set for $\beta_x(\alpha)$ is also a witness set for $\lambda_x(\alpha)$ and $h_x(\alpha)$. Similarly, witness models for the MDL estimator may not be witness models for the ML estimator. ◇

Shapes

The structure function graphs for data x $(l(x) = n)$ have integer coordinates and are situated in an $n \times n$ area. For the h_x function it is easy to see that the obvious constraints are

$$h_x(i) + i \geq K(x) + O(1),\ h_x(K(x) + O(1)) = 0,$$
$$h_x(K(n) + O(1)) \leq n,\ h_x(i) + i \leq h_x(j) + j + O(\log n),$$

for every $i \geq j$. In Example 5.5.9 on page 412 we see that the structure function h_x for every x of length n and complexity $K(x) \geq n - \log n$ approximately coincides with the diagonal $L(i) = n - i$. But there are fewer

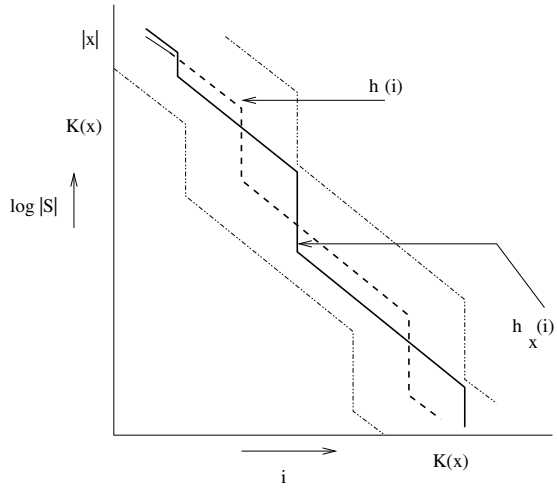


FIGURE 5.8. Graph of $h_x(i)$ in strip around $h(i)$

than $2^n/n$ strings x of length n that have complexity $K(x) < n - \log n$. Exercise 5.5.2 on page 421 shows that the number of distinct h functions available for them is far greater. Therefore, not every one of these possibilities can constitute an h_x function for some x. But Theorem 5.5.2 tells us that if we take a coarser view of the shape of a function, then every such shape can be realized by some x.

Recall Definition 5.5.8 on page 407. The next theorem states the possible shapes of the graphs of the λ-function in a precise form. By $K(i, n, \lambda)$ we mean the minimum length of a program that outputs n, i, and computes $\lambda(j)$ given any j in the domain of λ. We first analyze the possible shapes of the structure functions.

Theorem 5.5.2 (i) *For every n and every string x of length n and complexity $K(x)$, there is a nonincreasing integer-valued function λ defined on $[0, K(x)]$ such that $\lambda(0) \leq n$, $\lambda(K(x)) = K(x)$ and λ_x follows the shape of λ up to error $K(n)$.*

(ii) *Conversely, for every n, k, and nonincreasing integer-valued function λ whose domain includes $[0, k]$ and such that $\lambda(0) \leq n$ and $\lambda(k) = k$, there is a string x of length n and complexity $k \pm (K(k, n, \lambda) + O(1))$ such that $\lambda_x(i)$ follows the shape of $\lambda(i)$ up to error $K(i, n, \lambda)$.*

Proof. We defer the proof to Exercise 5.5.9 on page 424. □

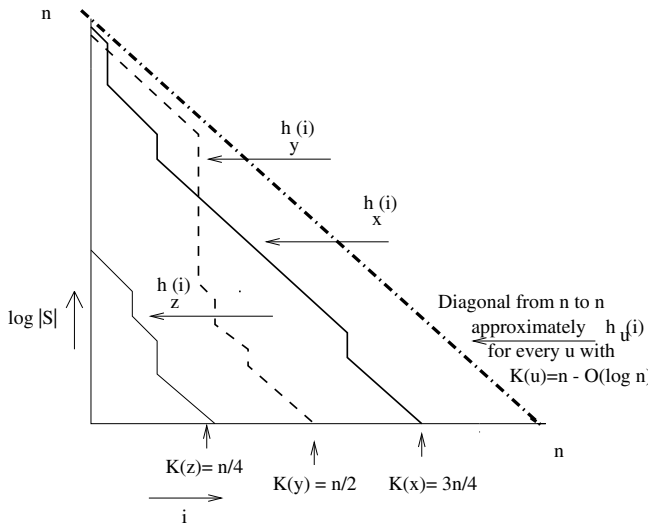


FIGURE 5.9. Some shapes of the structure function

Corollary 5.5.2 By Theorem 5.5.2, the MDL code length λ_x, and by Lemma 5.5.2 the original structure function h_x can assume essentially every possible shape as a function of the contemplated maximal model complexity. The theorem implies that for every function $h(i)$ defined on $[0, k]$ such that the function $\lambda(i) = h(i) + i$ satisfies the conditions of item (ii) there is an x such that the graph of $h_x(i)$ is situated in a strip of width $2K(h) + O(\log n)$ centered on $h(x)$ (absorbing the $K(i, n)$ term in the $O(\log n)$ term); see Figure 5.8.

Corollary 5.5.3 For every x of length n and complexity $k \leq n$ there is a nonincreasing function β such that $\beta(0) \leq n - k$, $\beta(k) = 0$, and β_x follows the shape of β up to error $O(\log n)$. Conversely, for every nonincreasing function β such that $\beta(0) \leq n - k$, $\beta(k) = 0$, there is x of length n and complexity $k \pm \delta$ such that β_x follows the shape of β up to error $O(\log n) + K(\beta)$.

Example 5.5.9 **(Variety of shapes)** Consider strings of n bits. For every curve monotonically nonincreasing at a slope of at most -1, in the triangle below the diagonal in Figure 5.9 there is some string that realizes it as its structure function, within logarithmic precision, Corollary 5.5.2. We depict possible structure functions for some string x of complexity $K(x) = \frac{3}{4}n$, some string y of complexity $K(y) = \frac{1}{2}n$, and some string z of complexity $K(z) = \frac{1}{4}n$. Note that strings of these complexities can have other structure functions. There are at least $(1-1/n)2^n$ data strings u of complexity $K(u) \geq n - \log n$, all of which have individual structure functions that

approximately coincide with the diagonal. They dominate to the extent that the pointwise expectation of the total collection of structure function curves coincides with the diagonal up to the stated precision. For another example, see Figure 5.10 on page 438. ◇

Example 5.5.10 **(Minimal sufficient statistic, continued)** We continue Example 5.5.5 on page 406. For every n there exist so-called *nonstochastic* objects x of length n that have a sufficient statistic of high complexity only. By Theorem 5.5.2, there are strings x of length n of complexity $K(x) = n - O(\log n)$ such that every sufficient statistic S for x has complexity $K(S) \geq n - O(\log n)$. See also Exercise 5.5.10 on page 424 and the historical comments at Figure 5.10 on page 438. Indeed, by Exercise 5.5.11 on page 425, there are even strings x of length n and complexity $K(x|n) = n - O(1)$ such that every sufficient statistic S of x has complexity either $K(S|n) > n - O(1)$ or $K(x|S) < \log d(S) - O(1)$. By Lemma 5.5.1 we have $K(x|S) \geq \log d(S) - O(1)$, so $K(S|n) > n - O(1)$. Clearly, we also have $K(S|n) + \log d(S) \leq K(x|n) + O(1) \leq n + O(1)$, which shows that $\log d(S) = O(1)$. Therefore, $d(S)$ is bounded by a fixed universal constant. For such x every sufficient statistic S consists of not much more than the singleton set consisting of the string itself. ◇

Example 5.5.11 **(Essence of model selection)** We have argued that the algorithmic statistical properties of a data sample x are fully represented by the set A_x of triples of Equation 5.21 on page 404. Up to additive logarithmic precision, Theorems 5.5.1, 5.5.2 describe the possible shapes of the closely related set B_x consisting of all triples (i, δ, λ) such that there is a set $S \ni x$ with $K(S) \leq i$, $\delta(x|S) \leq \delta$, $\Lambda(S) \leq \lambda$. That is, $A_x \subseteq B_x$, and A_x and B_x have the same minimal triples. Hence, we can informally say that our results describe completely possible shapes of the set of triples $(K(S), \delta(x|S), \Lambda(S))$ for nonimprovable models S explaining x. That is, up to $O(\log n)$ accuracy, and writing $k = K(x)$ and $n = l(x)$:

- For every minimal triple (i, δ, λ) in B_x we have $0 \leq i \leq k$, $0 \leq \delta$, and $\delta + k = \lambda \leq n$.
- There is a triple of the form $(i_0, 0, k)$ in B_x (the minimal such i_0 is the complexity of the minimal sufficient statistic for x). This property allows us to recover the complexity k of x from B_x.
- There is a triple of the form $(0, \lambda_0 - k, \lambda_0)$ in B_x with $\lambda_0 \leq n$.

Conversely, for every set B of triples of integers satisfying these three properties, there is a string x of length n and complexity close to k such that for every triple in B_x there is a triple in B such that the corresponding coordinates are within $O(\log n)$ of one another, and vice versa. ◇

Example 5.5.12 **(Computability)** The functions $h_x(i), \lambda_x(i), \beta_x(i)$ have finite domain for given x and hence can be given as a table—so formally speaking they are computable. But this evades the issue: there is no algorithm that computes these functions for every given x and i. Considering them as two-argument functions, we can show the following:

- The functions $h_x(i)$ and $\lambda_x(i)$, and their witness sets, are upper semicomputable but they are not computable up to any reasonable precision, Exercise 5.5.14 on page 426.
- Moreover, there is no algorithm that given x, $K(x)$, and i finds $h_x(i)$ or $\lambda_x(i)$ or their witness sets, Exercise 5.5.14, Item (b), on page 426.
- The function $\beta_x(i)$ is not upper or lower semicomputable, not even to any reasonable precision, Exercise 5.5.16 on page 426 and Exercise 5.5.17 on page 427. It is easy to see that one can compute $\beta_x(i)$ given an oracle for the halting problem. These statements hold also for the corresponding witness sets.
- There is no algorithm that given x and $K(x)$ finds a minimal sufficient statistic for x up to any reasonable precision, Exercise 5.5.15 on page 426.

◇

5.5.8 Foundations of MDL

In Theorem 5.4.1 in Section 5.4 we have shown that ideal MDL selects about the same model as maximum a posteriori Bayesian reasoning with the universal distribution as prior, provided we select models only from those for which the data are random. In Theorem 5.5.1 it turns out that for finite-set models, the contemplated model of least cardinality containing the data is a model for which the data are random and thus satisfies Theorem 5.4.1. The procedure is as follows.

Given x, the data to explain, and i, the maximum allowed complexity of an explanation, we search for programs p of length at most i that print a finite set $S \ni x$. Such pairs (p, S) are possible explanations. The best explanation is the (p, S) for which $\delta(x|S)$ is minimal. Since $K(x|S)$, and therefore the function $\delta(x|S) = \log d(S) - K(x|S)$, are not computable, we cannot find the best explanation directly. To overcome this problem we minimize the randomness deficiency by minimizing the MDL code length maximizing the fitness of the model for this data sample, by Theorem 5.5.1.

Definition 5.5.9 Let x be a data sample. Let A be an algorithm that, given x and some $i \leq l(x)+O(\log l(x))$, produces a finite sequence of pairs $(p_1, S_1), (p_2, S_2), \ldots,$

(p_f, S_f), where each p_t is a binary program of length at most i that prints a finite set $S_t \subseteq \{0,1\}^n$ with $x \in S_t$ $(1 \leq t \leq f)$. If $l(p_t) + \log d(S_t) < l(p_{t-1}) + \log d(S_{t-1})$ for all $1 < t \leq f$, then A is an *MDL algorithm with parameter* i.

Note that according to this definition an MDL algorithm may consider only a proper subset of all binary programs of length at most i. In particular, the final $l(p_f) + \log d(S_f)$ may be greater than the optimal MDL code of length $\min\{K(S) + \log d(S) : x \in S,\ K(S) \leq i\}$. This happens when the program p printing S with $x \in S$ and $l(p) = K(S) \leq i$ is not in the subset of binary programs considered by the algorithm. The next theorem gives an MDL algorithm that always finds the optimal MDL code and, moreover, the model concerned is shown to be an approximately best-fitting model for the data.

Lemma 5.5.3 *There exists an MDL algorithm with parameter i with the property that $\lim_{t\to\infty}(p_t, S_t) = (\hat{p}, \hat{S})$, such that $\delta(x|\hat{S}) \leq \beta_x(i - O(\log n)) + O(\log n)$.*

Proof. We exhibit such an MDL algorithms with parameter i:

Algorithm MDL(i):

> **Step 1.** Let x be a string of length n. Run all binary programs $p_1, p_2, \ldots$ of length at most i in lexicographic length-increasing order dovetailed fashion. The computation proceeds by stages $1, 2, \ldots$, and in each stage j the overall computation executes step $j - k$ of the particular subcomputation of p_k, for each k such that $j - k > 0$.
>
> **Step 2.** At every computation step t, consider all pairs (p, S) such that program p has printed the set $S \subseteq \{0,1\}^n$ containing x by time t. We assume that there is a first elementary computation step t_0 such that there is such a pair. Let a *best explanation* (p_t, S_t) at computation step $t \geq t_0$ be a pair that minimizes the sum $l(p) + \log d(S)$ among all the pairs (p, S).
>
> **Step 3.** We change the best explanation (p_{t-1}, S_{t-1}) of computation step $t - 1$ to (p_t, S_t) at computation step t only if $l(p_t) + \log d(S_t) < l(p_{t-1}) + \log d(S_{t-1})$.

In this MDL algorithm with parameter i, the best explanation (p_t, S_t) changes from time to time due to the appearance of a strictly better explanation. Since no pair (p, S) can be selected as best explanation twice, and there are only finitely many pairs, from some moment onward the explanation (p_t, S_t) that is declared best no longer changes. Therefore the limit $(\hat{p}, \hat{S})$ exists. The model $\hat{S}$ is a witness set of $\lambda_x(i)$. The lemma follows by Theorem 5.5.1 and Example 5.5.7. □

Thus, if we continue to approximate the two-part MDL code contemplating every relevant model, then we will eventually reach the optimal code that is approximately the best explanation. That is the good news. The bad news is that we do not know when we have reached this optimal solution. The functions h_x and λ_x, and their witness sets, cannot be computed within any reasonable accuracy, Exercise 5.5.14 on page 426. Hence, there does not exist a criterion that we could use to terminate the approximation somewhere close to the optimum. In the practice of real-world MDL, in the process of finding the two-part MDL code, we often have to be satisfied with running times t that are much less than the time to stabilization of the best explanation $(\hat{p}, \hat{S})$. For such small t, the model S_t has a weak guarantee of goodness, since we know that

$$\delta(x|S_t) + K(x) \leq l(p_t) + \log d(S_t) + O(1),$$

because $K(x) - K(x|S_t) \leq K(S_t) + O(1) \leq l(p_t) + O(1)$. That is, the sum of the randomness deficiency of x in S_t and $K(x)$ is less than a known value. Lemma 5.5.3 implies that Algorithm MDL gives not only *some* guarantee of goodness during the approximation process, but also that in the limit, that guarantee approaches the value of its lower bound, that is, $\delta(x|\hat{S}) + K(x)$. Thus, in the limit, Algorithm MDL will yield an explanation that is only a little worse than the best explanation.

Example 5.5.13 **(Direct method)** Use the same dovetailing process as in Algorithm MDL, with the following addition. At every elementary computation step t, select a (p, S) for which $\log d(S) - K^t(x|S)$ is minimal among all programs p that up to this time have printed a set S containing x. Here $K^t(x|S)$ is the approximation of $K(x|S)$ defined by $K^t(x|S) = \min\{l(q) : \text{the reference universal prefix machine } U \text{ outputs } x \text{ on input } (q, S) \text{ in at most } t \text{ steps}\}$. Let (q_t, S_t) denote the best explanation after t steps. We change the best explanation at computation step t only if $\log d(S_t) - K^t(x|S_t) < \log d(S_{t-1}) - K^{t-1}(x|S_{t-1})$.

This time, the same explanation can be chosen as the best one twice. However, from some time t onward, the best explanation (q_t, S_t) no longer changes. In the approximation process, the model S_t has no guarantee of goodness at all: since $\beta_x(i)$ is not semicomputable to any significant precision by Exercise 5.5.16 on page 426, we cannot know an upper bound, either for $\delta(x|S_t)$ or for $\delta(x|S_t) + K(x)$. Hence, we must prefer the indirect method of Algorithm MDL, approximating a witness set for $\lambda_x(i)$, instead of the direct one of approximating a witness set for $\beta_x(i)$. ◇

In practice we often must terminate an MDL algorithm as in Definition 5.5.9 prematurely. It is seductive to assume that the longer we approximate the optimal two-part MDL code, the better the resulting

model explains the data, that is, that every next shorter two-part MDL code also yields a better model. This is incorrect. To give an example that shows where things go wrong, it is easiest first to give the conditions under which premature search termination is all right. Suppose we replace the currently best explanation (p_1, S_1) in an MDL algorithm with explanation (p_2, S_2) only if $l(p_2) + \log d(S_2)$ is not just less than $l(p_1) + \log d(S_1)$, but less by slightly more than the excess of $l(p_1)$ over $K(S_1)$. Then, it turns out that every time we change the explanation, we improve its goodness unless the change is just caused by the fact that we have not yet found the minimum-length program for the current model.

Lemma 5.5.4 *Let x be a string of length n, let A be an MDL algorithm, and let (p_1, S_1) and (p_2, S_2) be sequential (not necessary consecutive) candidate best explanations produced by A. If*

$$l(p_2) + \log d(S_2) \leq l(p_1) + \log d(S_1) - (l(p_1) - K(S_1)) - 10 \log n,$$

then $\delta(x|S_2) \leq \delta(x|S_1) - 5 \log n$. (These conditions are satisfied by Algorithm MDL.)

Proof. For every pair of sets $S_1, S2 \ni x$ we have

$$\delta(x|S_2) - \delta(x|S_1) = \Lambda(S_2) - \Lambda(S_1) + \Delta,$$

with

$$\begin{aligned} \Delta = & -K(S_2) - K(x|S_2) + K(S_1) + K(x|S_1) \\ \leq & -K(S_2, x) + K(S_1, x) + K(S_1^*|S_1) + O(1) \\ \leq & K(S_1, x|S_2, x) + K(S_1^*|S_1) + O(1). \end{aligned}$$

Ignoring $O(1)$ terms, we rewrite according to the symmetry of information theorem, Theorem 3.9.1 on page 247, noting that $-K(S_2, x) \geq -K(S_2) - K(x|S_2)$ and $K(S_1, x) + K(S_1^*|S_1) \geq K(S_1) + K(x|S_1)$, and finally use that $K(a|b) \geq K(a) - K(b)$. Setting $\Lambda = \Lambda(S_2) - \Lambda(S_1)$, we obtain, by the assumption in the theorem,

$$\begin{aligned} \Lambda & \leq l(p_2) + \log d(S_2) - \Lambda(S_1) \\ & = l(p_2) + \log d(S_2) - (l(p_1) + \log d(S_1)) + (l(p_1) - K(S_1)) \\ & \leq -10 \log n. \end{aligned}$$

Since by assumption the difference in MDL codes $\Lambda(S_2) - \Lambda(S_1) > 0$, it suffices to show that $K(S_1, x|S_2, x) + K(S_1^*|S_1) \leq 5 \log n$ to prove the theorem. To identify S_1 we need to know only the MDL algorithm concerned, the maximal complexity i of the contemplated models, (p_2, S_2), the number j of candidate best explanations between (p_1, S_1)

and (p_2, S_2), and x. To identify S_1^* from S_1 we require only $K(S_1)$ bits. The program p_2 can be found from S_2 and length of p_2 as the first program computing S_2 of length $l(p_2) \leq i$ in the process of running all programs of length at most i dovetailed style. In an MDL algorithm, $j \leq l(p_1) + \log d(S_1) \leq i + n$ and $K(S_1) \leq i$. Therefore,

$$\begin{aligned} K(S_1, x|S_2, x) + K(S_1^*|S_1) &\leq \log l(p_2) + \log i + \log K(S_i) + \log j + b \\ &\leq 3 \log i + \log(i + n) + b, \end{aligned}$$

where b is the number of bits we need to encode the description of the MDL algorithm concerned, the delimiters to separate the descriptions of the constituent parts given above (apart from x), and the description of a program to reconstruct S_1 and S_1^* from these data plus x. Since $i \leq n + O(\log n)$, we find that $K(S_1, x|S_2, x) + K(S_1^*|S_1) \leq 4 \log n + O(\log \log n)$, which finishes the proof. □

Example 5.5.14 In the sequence $(p_1, S_1), (p_2, S_2), \ldots$ of candidate best explanations produced by an MDL algorithm according to Definition 5.5.9, $(p_{t'}, S_{t'})$ is actually better than (p_t, S_t) $(t < t')$ if improvement in two-part MDL code length is in excess of the unknown, and in general incomputable, $l(p_t) - K(S_t)$. On the one hand, if $l(p_t) = K(S_t) + O(1)$ and

$$l(p_{t'}) + \log d(S_{t'}) \leq l(p_t) + \log d(S_t) - 10 \log n,$$

then $S_{t'}$ is a better explanation for data x than S_t in the sense that

$$\delta(x|S_{t'}) \leq \delta(x|S_t) - 5 \log n.$$

On the other hand, if $l(p_t) - K(S_t)$ is large, then $S_{t'}$ may be a much worse explanation than S_t. Then, it is possible that we improve the two-part MDL code length by giving a worse model $S_{t'}$ using, however, a $p_{t'}$ such that $l(p_{t'}) + \log d(S_{t'}) < l(p_t) + \log d(S_t)$ while $\delta(x|S_{t'}) > \delta(x|S_t)$. ◇

Example 5.5.15 **(True model not in model class)** A question arising in MDL or maximum likelihood (ML) estimation is its performance if the true model is not part of the contemplated model class. Given certain data, why would we assume that they are generated by probabilistic or deterministic processes? They have arisen by natural processes most likely not conforming to mathematical idealization. Even if we can assume that the data arose from a process that can be mathematically formulated, we may restrict modeling of data arising from a complex source (conventional analogue being data arising from $2k$-parameter sources) to simple models (conventional analogue being k-parameter models). Theorem 5.5.1 shows that within the class of models of maximal complexity i, we still select a

model for which the data is maximally typical, even if the minimal sufficient statistic has complexity $\gg i$, that is, $h_x(i) + i \gg K(x) + O(1)$. This situation is potentially common by Theorem 5.5.2, in particular if the data is nonstochastic, Exercise 5.5.10 on page 424. ◇

5.5.9 Explanation and Interpretation

Let $l(x) = n$. Theorem 5.5.1 states that within negligible additive logarithmic (in n) terms, in argument and value,

$$K(x) + \beta_x(i) = \lambda_x(i) = h_x(i) + i. \tag{5.25}$$

This relation among the structure functions is depicted in Figure 5.7. Thus, up to logarithmic additive precision,

$$h_x(i) - \beta_x(i) = L_x(i),$$

where $L_x(i) = K(x) - i$ is the sufficiency line. See also Exercise 5.5.8 on page 423.

Trivial Case: $h_x(i) + i = K(x) + O(1)$. Then, the witness set S of log-cardinality $h_x(i)$ is a sufficient statistic of x, and by Lemma 5.5.1 we have that x is a typical element of S. This means that the randomness deficiency $\delta(x|S)$ is $O(1)$, that is, $\beta_x(i) = O(1)$. Then, Theorem 5.5.1 and Equation 5.25 hold trivially. For every x, with minimal sufficient statistic S_0, the trivial case occurs for every $i \in [K(S_0), K(x) + O(1)]$, and for those arguments $h_x(i) = L_x(i)$ up to an additive constant.

Consider the case that x is random in the sense of having high prefix complexity, $K(x|n) = O(1)$. That is, x is a typical outcome of n fair coin flips. Then, $S = \{0,1\}^n$ is a sufficient statistic for x, implying that the minimal sufficient statistic S_0 for x has complexity $K(S_0) \leq K(S) = K(n) + O(1) = \log n + O(\log\log n)$. So for these high-complexity x's the trivial case occurs for virtually all arguments i, that is, at least for all $i \in [K(n) + O(1), K(x) + O(1)]$.

Nontrivial Case: $h_x(i) + i > K(x) + O(1)$. Such strings x have precisely randomness deficiency $\beta_x(i) = h_x(i) + i - K(x)$ in the witness set S of $h_x(i)$, ignoring logarithmic additive terms in argument and value. This is the best fit, up to the stated precision, that we can achieve for models of complexity at most i. The importance of Theorem 5.5.1 and Equation 5.25 lies in the fact that for every string x containing significant effective regularities, the minimal sufficient statistic S_0 has complexity $K(S_0)$ significantly greater than $K(n)$. Then, $h_x(i) + i > K(x) + O(1)$ for every $i \in [0, K(S_0)]$, and the bound $K(S_0)$ can take all values up to $K(x)$ by Theorem 5.5.2.

Since $K(S_0)$ is incomputable, Exercise 5.5.15 on page 426, and can be quite large, it is important that if we restrict our search to a model

of complexity i, possibly (but unknown to us) smaller than $K(S_0)$, we still know the following: Algorithm MDL will upper semicompute approximately the MDL estimator $\lambda_x(i)$ and a witness set (a variant of Algorithm MDL will upper semicompute approximately the ML estimator $h_x(i)$ and a witness set). Thus, these algorithms in the limit return a model for x of almost best fit among all models satisfying the complexity restriction i, Lemma 5.5.3.

Summary: For all data x, both the maximum likelihood estimator and the minimum description length estimator select a model that optimizes the best-fit estimator, in the class of complexity-restricted contemplated models, *surely* and not only with high probability. In particular, when the true model that generated the data is not in the complexity-restricted model class considered, then the ML or MDL estimator still gives a model that best fits the data. While the best-fit quantity $\beta_x(i)$ cannot be computationally monotonically approximated up to any significant precision, Exercise 5.5.16 on page 426, we can monotonically minimize the two-part MDL code (find a model witnessing $\lambda_x(i)$ Lemma 5.5.3), or the model cardinality (find a model witnessing $h_x(i)$), and thus monotonically approximate *implicitly* the almost best-fitting model. But this should be sufficient: we want a good model rather than a number that measures its goodness. These results usher in a completely new era of statistical inference that is *always* best rather than *expected*.

The generality of the results is at the same time a restriction. In classical statistics one is commonly interested in model classes that are partially poorer and partially richer than the ones we consider. For example, the classes of Bernoulli processes or k-state Markov chains, with computable parameters, are poorer than the class of computable probability mass functions of moderate maximal Kolmogorov complexity i, in that the latter may contain functions that require far more complex computations than the rigid syntax of the former classes allows. Indeed, the class of computable probability mass functions of even moderate complexity allows implementation of a function mimicking a universal Turing machine computation. On the other hand, even the lowly Bernoulli process can be equipped with an incomputable real bias in $(0, 1)$. This incomparability between the algorithmic model classes and the statistical model classes means that the current results cannot be directly transplanted to the traditional setting. Indeed, they should be regarded as pristine truths that hold in a Platonic world that can be used as a guideline to develop analogues in model classes that are of more traditional concern.

Exercises

5.5.1. [25] Let x be a finite binary string and let S be a finite set containing x. Define S to be *strongly typical* for x if $\log d(S) - K(x|S^*) = O(1)$, where S^* is the first shortest program to print S in lexicographic

length-increasing order. (The standard Definition 5.5.3 on page 402 is slightly weaker since it requires $\log d(S) - K(x|S)$ to be $O(1)$.) Recall that $I(x;S) = K(x) + K(S) - K(x,S) + O(1)$ is the truly symmetric variant of algorithmic mutual information.

(a) Show that if S is strongly typical for x, then $I(x;S) = K(x) - \log d(S) + O(1)$.

(b) Show that if S is strongly typical for x, then $I(x;S) \geq K(K(x)) - K(I(x;S)) + O(1)$ and $\log d(S) \leq K(x) - K(K(x)) + K(I(x;S)) + O(1)$.

(c) Show that S is a sufficient statistic for x iff it is strongly typical and $K(S|x^*) = O(1)$.

(d) Show that if S is a sufficient statistic for x, then $I(x;S) = K(S) + O(1) \geq K(K(x)) + O(1)$ and $\log d(S) \leq K(x) - K(K(x)) + O(1)$.

Comments. Item (d) tells us that every sufficient statistic for x has complexity at least $K(K(x))$. Source: P. Gács, J.T. Tromp, and P.M.B. Vitányi, *IEEE Trans. Inform. Theory*, 47:6(2001), 2443–2463; Correction, 48:8(2002), 2427.

5.5.2. [21] (a) Compute the number of different integer functions h defined on $0, 1, \ldots k$ for some $k \leq n - \log n$ satisfying $h(0) \leq n$ and $h(i) + i$ is nonincreasing.

(b) Conclude that the number in Item (a) is far greater than the number of x's of length n and complexity $k \leq n - \log n$.

Comments. Hint for Item (a): The function $h(i)$ starts for $k = 0$ at some nonnegative value $\leq n$. Here $\delta_0 = n - h(0)$ can be viewed as the first decrease. At every increment of i, the function $h(i)$ can decrease one or more units $\delta_i \geq 1$. There are k increments. The sum total is n. Hence we consider $n-k$ units that can be partitioned into $k+1$ nonnegative integer summands. For every k there are $\binom{n}{k}$ such partitions. Hence there are $\sum_{k=0}^{n-\log n} \binom{n}{k} \geq 2^{n-1}$ ways to partition as asked in Item (a). All strings x of length n and complexity $k \geq n - \log n$ have structure functions h_x roughly following the diagonal $L(i) = n - i$. The number of x's of length n with complexity $k \leq n - \log n$ is at most $2^{n-K(n)-\log n+O(1)}$ by Theorem 3.3.1. This is far less than the number of functions h available for them (potential structure functions h_x) as computed in Item (a). Source: N.K. Vereshchagin and P.M.B. Vitányi, Unpublished, 2007.

5.5.3. [30] Prove the difficult side of Lemma 5.5.2 on page 407.

Comments. Source: N.K. Vereshchagin and P.M.B. Vitányi, *IEEE Trans. Inform. Theory*, 50:12(2004), 3265- 3290.

5.5.4. [39] (a) Show that for every set $A \ni x$ there is a set $S \ni x$ with $K(S) \leq K(A) + O(\log \Lambda(A))$ and $\lceil \log d(S) \rceil = \lceil \log d(A) \rceil - K(A|x) + O(\log \Lambda(A))$.

(b) Show that for every set $A \ni x$ there is a set $S \ni x$ with $K(S) \leq K(A) - K(A|x) + O(\log \Lambda(A))$ and $\lceil \log d(S) \rceil = \lceil \log d(A) \rceil$. Recall that $\Lambda(A) = K(A) + \log d(A)$.

Comments. Item (b) implies Item (a), which in turn proves Equation 5.23 on page 408, and hence finishes the proof of Theorem 5.5.1. Source: N.K. Vereshchagin and P.M.B. Vitányi, *Ibid.*

5.5.5. • [25] (a) Show that if we recode data x by its shortest program x^*, then this can change the structure functions.

(b) Let f be a recursive permutation of the set of finite binary strings (one-to-one, total, and onto). Show that the graph of $h_{f(x)}$ follows the shape of h_x with error at most $K(f) + O(1)$.

Comments. If the structure functions could change under common recodings of the data, clearly our claim that the structure functions represent the stochastic properties of the data would be false: One doesn't believe that those properties change under common recoding. Hint for Item (a): since x^* is incompressible, it is a typical element of the set of all strings of length $l(x^*) = K(x)$, and hence $h_{x^*}(i)$ drops to the sufficiency line $L_x(i) = K(x) - i$ already for some $i \leq K(K(x))$, so almost immediately (and it stays within logarithmic distance of that line henceforth). That is, ignoring logarithmic additive terms, $h_{x^*}(i) + i = K(x)$ irrespective of the (possibly quite different) shape of h_x. Since the Kolmogorov complexity function $K(x) = l(x^*)$ is not recursive, the recoding function $f(x) = x^*$ is also not recursive. Moreover, while f is one-to-one and total, it is not onto. However, the structure function is invariant under 'proper' recoding of the data as in Item (b). Source: N.K. Vereshchagin and P.M.B. Vitányi, *Ibid.* Item (a) was first observed by O. Watanabe.

5.5.6. [35] Let x be a finite binary string. A *prediction strategy* P is a mapping from the set of strings of length less than $l(x)$ into the set of rational numbers in the segment $[0, 1]$. The value $P(x_1 \ldots x_i)$ $(i < n)$ is regarded as our belief (or probability) that $x_{i+1} = 1$ after we have observed $x_1, \ldots, x_i$. If the actual bit x_{i+1} is 1, the strategy suffers the loss $\log 1/p$, otherwise $\log(1/(1-p))$, for $p = P(x_1 \ldots x_i)$. The strategy is a finite object and $K(P)$ may by defined as the complexity of this object, or as the minimum size of a program that identifies n and, given y, finds $P(y)$. The *total loss* $\mathrm{Loss}_P(x)$ of P on x is the sum of all n losses, $\mathrm{Loss}_P(x) = \sum_{i=0}^{n-1} \log 1/|P(x_1 \ldots x_i) - 1 + x_{i+1}|$. The *snooping curve* s_x is defined by $s_x(i) = \min_{P:K(P)\leq i} \mathrm{Loss}_P(x)$. The snooping curve $s_x(i)$ gives the minimal loss suffered on all of x by a prediction strategy as a function of the complexity of the prediction strategies. Show that $s_x(i + O(\log l(x))) \leq h_x(i)$ and $h_x(i + O(\log l(x))) \leq s_x(i)$ for every x and i.

Comments. Hence, if $S \ni x$ is a smallest set such that $K(S) \leq i$, then S can be converted into a best strategy of complexity at most i to predict the successive bits of x given the preceding ones, and vice versa. The snooping curve was proposed by V.V. Vyugin *Theoret. Comput. Sci.*, 276:1-2(2002), 407–415, who obtained partial results. Because of Item (a), Lemma 5.5.2 and Theorem 5.5.2 describe the coarse shape of all possible snooping curves. From Item (b) it follows that the witness set of $h_x(i)$ not only is an almost best-fitting model for x, but also can be converted into an almost best predictor for the successive bits of x. The notion of the snooping curve $s_x(i)$ of a finite string x quantifies the quality of prediction expressed in terms of the errors in predicting the probabilities of the consecutive elements of x. In the prediction situation in Section 5.2.7 we wanted to predict the actual consecutive elements of an infinite sequence, and the quality of the prediction algorithm is given by the number of mistakes we make. Source: N.K. Vereshchagin and P.M.B. Vitányi, *Ibid.*

5.5.7. [31] (a) Show that there is a string x of length n and complexity about $\frac{1}{2}n$ for which $\beta_x(O(\log n) = \frac{1}{4}n + O(\log n)$.

(b) Show that for the set $A_0 = \{0,1\}^n$ we have $K(A_0) = O(\log n)$ and $K(x|A_0) = \frac{1}{2}n + O(\log n)$, and therefore $K(A_0) + K(x|A_0) = \frac{1}{2}n + O(\log n)$ is minimal up to a term $O(\log n)$.

(c) Show that the randomness deficiency of x in A_0 is about $\frac{1}{2}n$, which is much bigger than the minimum $\beta_x(O(\log(n)) \approx \frac{1}{4}n$.

(d) Show that for the model A_1 witnessing $\beta_x(O(\log(n)) \approx \frac{1}{4}n$ we also have $K(A_1) = O(\log n)$ and $K(x|A_1) = \frac{1}{2}n + O(\log n)$, but $\log d(A_1) = \frac{3}{4}n + O(\log n)$, which causes the smaller randomness deficiency.

Comments. Ultimate compression of the two-part code in ideal MDL, Section 5.4, means minimizing $K(A) + K(x|A)$ over all models A in the model class. In Theorem 5.5.1 we have essentially shown that the worst-case data-to-model code is the approach that guarantees the best-fitting model. In contrast, the ultimate compression approach can yield models that are far from best fit. It is easy to see that this happens only if the data are not typical for the contemplated model. Hint for Item (a): use Corollary 5.5.3 on page 412. Source: N.K. Vereshchagin and P.M.B. Vitányi, *Ibid.*

5.5.8. [23] Use the terminology of Section 5.5.1. Apart from parameters i, β, γ, there a fourth important parameter, $K(S|x^*)$, reflecting the *determinacy of model* S by the data x. Prove the equality $\log d(S) + K(S) - K(x) = K(S|x^*) + \delta(x|S) + O(\log n\Lambda(S))$. Conclude that the central result Equation 5.25 on page 419 establishes that $K(S|x^*)$ is logarithmic in $l(x)$ for *every* set S witnessing $h_x(i)$. This also shows that there are at most polynomially many (in $l(x)$) such sets.

Comments. Source: N.K. Vereshchagin and P.M.B. Vitányi, *Ibid.*

5.5.9. [34] Prove Theorem 5.5.2.

Comments. A less precise result of the same nature is given in Theorem 8.1.6 on page 619, and its proof deferred to Exercise 8.1.8 on page 626, for arbitrary distortion measures. In the terminology used there, the Kolmogorov structure function is the distortion-rate function of list distortion. Source: N.K. Vereshchagin and P.M.B. Vitányi, *Ibid.*

5.5.10. [34] Let x be string and α, β natural numbers. Kolmogorov called a string (α, β)-*stochastic* if there is a finite set $A \subseteq \mathcal{N}$ and $x \in A$ such that

$$x \in A, \quad K(A) \leq \alpha, \quad K(x|A) \geq \log d(A) - \beta.$$

The first inequality (with α not too large) means that A is sufficiently simple. The second inequality (with β not too large) means that x is an undistinguished (typical) element of A. If x had properties defining a very small subset B of A, then these could be used to obtain a simple description of x by determining its ordinal number in B. This would require $\log d(B)$ bits, which would be significantly less than $\log d(A)$.

(a) Show that (α, β)-stochasticity of x is equivalent to $\beta_x(\alpha) \leq \beta$.

(b) Show that the overwhelming majority of strings of length n with an equal number of 0's and 1's is $(O(\log n), O(1))$-stochastic.

(c) Show that the overwhelming majority of strings in $\{0,1\}^n$ is $(O(\log n)$,-$O(1))$-stochastic.

(d) Show that for some constants c, C, for every n and every α, β with $\alpha \geq \log n + C$ and $\alpha + \beta \geq n + c \log n + C$, every string of length n is (α, β)-stochastic.

(e) Show that for some constants c, C, for all n and all α, β with $\alpha + \beta < n - c \log n - C$, there is a string x of length n that is not (α, β)-stochastic.

Comments. Hint for Item (b): the set A has cardinality $\binom{n}{n/2} = \Theta(2^n/\sqrt{n})$ and $K(A) = K(n) + O(1) = O(\log n)$. The majority of elements $x \in A$ have complexity $K(x|A) = \log d(A) + O(1)$. Hint for Item (c): although the overwhelming majority of strings in $\{0,1\}^n$ do not belong to A, they are nonetheless also $(O(\log n), O(1))$-stochastic, this time with respect to the set $\{0,1\}^n$. Source for Item (d): A.K. Shen, *Soviet Math. Dokl.*, 28:1(1983), 295–299. Item (e): in [A.K. Shen, *Ibid.*] it is proven that for some constants c, C, for all n and all α, β with $2\alpha + \beta < n - c \log n - C$ there is a string x of length n that is not (α, β)-stochastic. Use Corollary 5.5.3 to prove the stronger result. It follows from Items (d) and (e) that if we take $\alpha = \beta$, then for some boundary around $\frac{1}{2}n$, the last non-(α, β)-stochastic elements disappear if the complexity constraints

are sufficiently relaxed by having α, β exceed this boundary. Source for Item (e): N.K. Vereshchagin and P.M.B. Vitányi, *Ibid.*

5.5.11. [37] (a) Show that there are strings x of length n such that the algorithmic minimal sufficient statistic is essentially the singleton set consisting of the string itself. Formally, there are constants c, C such that for every given $k < n$ there exists a string x of length n with complexity $K(x|n) \leq k$ that is not $(k - c, n - k - C)$-stochastic in the sense of Exercise 5.5.10.

(b) Show that there are strings x of length n that have randomness deficiency $\log d(S) - K(x|S^*) \geq n - k$ with respect to every finite set $S \ni x$ of complexity $K(S|n) < k$, for every k.

Comments. Item (b) improves Item (e) of Exercise 5.5.10 by an additive logarithmic term to the best possible result on nonstochastic strings. Since x has $\log d(S) - K(x|S^*) \geq n - K(S|n)$ for every finite $S \ni x$, and in particular with $K(S|n)$ more than a fixed constant below n, the randomness deficiency $\log d(S) - K(x|S^*)$ exceeds that fixed constant: every sufficient statistic for x has complexity at least n. But setting $S = \{x\}$, we have $K(S|n) = K(x|n) < n$ and $\log d(S) - K(x|S^*) = n - K(S|n) = 0$. Together, this shows that the absolutely nonrandom objects x of length n of which we have established the existence have complexity $K(x|n) = n$ and have significant randomness deficiency with respect to every set $S \ni x$ that has complexity significantly below that of x. Here, the (in)equalities hold up to additive constants. These strings are absolutely nonstochastic objects; they are strangely reluctant to reveal their structure. Source: P. Gács, J.T. Tromp, and P.M.B. Vitányi, *Ibid.*

5.5.12. [35] Give a general uniform construction of the finite sets $S_{i,l}$ witnessing the structure functions $\lambda_x(i)$ and $\beta_x(i)$, at each argument i, in terms of indexes of x in the enumeration of strings of given complexity. That is, for every x there is a sequence $l_1 \leq l_2 \leq \cdots \leq l_{K(x)} \leq n + O(\log n)$ such that the functions $\lambda(i) = K(S_{i,l_i}) + \log d(S_{i,l_i})$, and $\beta(i) = \log d(S_{i,l_i}) - K(x|S_{i,l_i})$, follow the shapes of λ_x and β_x, respectively, up to error $O(\log n)$, according to Definition 5.5.8 on page 407. In view of the incomputability of structure functions, the construction is of course not computable.

Comments. This extends a technique introduced in P. Gács, J.T. Tromp, and P.M.B. Vitányi, *Ibid.* Source: N.K. Vereshchagin and P.M.B. Vitányi, *Ibid.*

5.5.13. [20] Consider strings x of length n. Define the conditional variant of Definition 5.5.6 on page 405 as $h_x(i|y) = \min_S\{\log d(S) : S \ni x,\ d(S) < \infty,\ K(S|y) \leq i\}$. Since $S_1 = \{0,1\}^n$ is a set containing x and can be described by $O(1)$ bits (given n), we find that $h_x(i|n) \leq n$ for $i = K(S_1|n) = O(1)$. For increasing i, the size of a set $S \ni x$, which

one can describe in i bits, decreases monotonically until for some i_0 we obtain a first set S_0 witnessing $h_x(i_0|n)+i_0 = K(x|n)+O(1)$. Then, S_0 is a minimal sufficient statistic for x.

(a) Consider $i \geq i_0$. Show that every increase of i about halves the witness set S_i of $h_x(i|n)$, for each additional bit of i, until $i = K(x|n)$, in the sense that $h_x(i_0 + d|n) = K(x|n) - (i_0 + d + O(\log d))$, provided the right-hand side is nonnegative, and 0 otherwise.

(b) Show that the number of different sufficient statistics for x is bounded by a universal constant independent of x. Argue that this causes the little bumps in the sufficient statistic region $\subseteq [K(K(x)), K(x)]$ in Figure 5.6.

Comments. Source: Item (a) N.K. Vereshchagin and P.M.B. Vitányi, *Ibid.*, and Item (b) P. Gács, J.T. Tromp, and P.M.B. Vitányi, *Ibid.*; N.K. Vereshchagin and P.M.B. Vitányi, *Ibid.*

5.5.14. [43] Consider $\lambda_x(i)$ as a two-argument function as in Example 5.5.12.

(a) Show that $\lambda_x(i)$ is upper semicomputable, but not computable.

(b) Show that λ_x is not computable, given $x, K(x)$, even in an approximate sense: There is no function λ that is computable given $x, K(x)$, such that $\lambda_x(i)$ follows the shape of $\lambda(i)$ with error at most $l(x)/(10 \log l(x))$, in the sense of Definition 5.5.8 on page 407.

(c) Show that given $x, K(x)$, and i_0 such that $\lambda_x(i_0) \approx K(x)$, but $\lambda_x(i)$ significantly greater than $K(x)$ for i significantly less than i_0 (so that i_0 is the complexity of the minimal sufficient statistic), then we can compute λ_x over all of its domain. This result underlines the significance of the information contained in the *minimal* sufficient statistic. Formally, there is a constant $c \geq 0$ and an algorithm that given any x, k, i_0 with $K(x) \leq k \leq \lambda_x(i_0)$ finds a nonincreasing function λ defined on $[0, k]$ such that $\lambda_x(i)$ follows the shape of λ with error $\lambda_x(i_0) - K(x) + O(1)$ on the interval $[i_0 - i_1 + c \log k, k]$, where $i_1 = \min\{i : \lambda_x(i) \leq k + c \log k\}$.

Comments. What holds for $\lambda_x(i)$ above equally holds for $h_x(i)$ and the finite set witnessing its value. Source: N.K. Vereshchagin and P.M.B. Vitányi, *Ibid.* Item (c) is attributed to An.A. Muchnik.

5.5.15. Show that it is impossible to approximate the complexity of the minimal sufficient statistic of x, even if we are given both x and $K(x)$.

Comments. Source: N.K. Vereshchagin and P.M.B. Vitányi, *Ibid.*

5.5.16. [42] Consider $\beta_x(i)$ as a two-argument function as in Example 5.5.12.

(a) Show that the function $\beta_x(i)$ is computable from x, i given an oracle for the halting problem.

(b) Show that the function $\beta_x(i)$ is upper semicomputable from $x, i, K(x)$ up to a logarithmic error.

(c) Show that the set $\{(x, S, \beta) : x \in S,\ \delta(x|S) < \beta\}$ is not recursively enumerable.

(d) Show that $\beta_x(i)$ is not lower semicomputable to within precision $l(x)/3$ (there is no lower semicomputable function $f(i)$ such that $|f(i) - \beta_x(i)| \leq l(x)/3$); and

(e) Show that $\beta_x(i)$ is not upper semicomputable to within precision $l(x)/\log^4 l(x)$. (There is no upper semicomputable function $f(i)$ such that $|f(i) - \beta_x(i)| \leq l(x)/\log^4 l(x)$).

Comments. Hint for Item (a): run all programs of length $\leq i$ dovetailed fashion and find all finite sets S containing x that are produced. With respect to all these sets determine the conditional complexity $K(x|S)$ and hence the randomness deficiency $\delta(x|S)$. Taking the minimum, we obtain $\beta_x(i)$. All these things are possible using information from the halting problem to determine whether a given program will terminate. Hint for Item (b): this follows from the upper semicomputability of $\lambda_x(i)$ and Theorem 5.5.1. Items (d) and (e) show that the function $\beta_x(i)$ is not semicomputable, not even within a large margin of error. Source: N.K. Vereshchagin and P.M.B. Vitányi, *Ibid.*

5.5.17. [35] (a) Prove that the function $\beta_x(i)$ is not upper semicomputable to within precision $l(x)/4$ (there is no upper semicomputable function $f(i)$ such that $|f(i) - \beta_x(i)| \leq l(x)/4$).

(b) Prove that there is no algorithm that for every n and every x of length n upper semicomputes a nonincreasing function that follows the shape of β_x with small error, in the sense of Definition 5.5.8 on page 407. In particular, there is a function $\delta(n) = O(\log n)$ with the following property. Let $\alpha(n)$ and $\epsilon(n)$ be computable natural-valued functions with $2\epsilon(n) \leq \alpha(n) \leq n - 2\epsilon(n) - \delta(n)$ and $\epsilon(n) > \delta(n)$. Then, there is no upper semicomputable function f, given x, such that for every n and every x of length n, we have $\beta_x(\alpha(n)) < f(\alpha(n)) < \beta_x(\alpha(n) - \epsilon(n)) + \epsilon(n)$.

Comments. Item (a) improves Exercise 5.5.16, Item (e). Item (b) resolves part of an open problem in N.K. Vereshchagin and P.M.B. Vitányi, *Ibid.* Source: M.A. Ustinov *Proc. Int. Comput. Sci. Symp. Russia (CSR)*, Lect. Notes. Comput. Sci., Vol. 3967, Springer-Verlag, Berlin, 2006, 364–368.

5.5.18. [O39] It is unknown whether there is an algorithm that for every x, lower semicomputes a nonincreasing function $f(i)$ that follows the shape of $\beta_x(i)$ with error $O(\log n)$ (or even $o(n)$), in the sense of Definition 5.5.8 on page 407.

Comments. The analogous question concerning upper semicomputability is settled in Exercise 5.5.17, Item (b). Source: N.K. Vereshchagin and P.M.B. Vitányi, *Ibid.*

5.5.19. • [28] The model class of *computable probability mass functions* (*probability models*) consists of the set of functions $P : \{0,1\}^* \to [0,1]$ with $\sum P(x) = 1$. 'Computable' means here that there is a Turing machine T_P that given x and a positive rational ϵ, computes $P(x)$ within precision ϵ. The prefix complexity $K(P)$ of a computable (possibly partial) function P is defined by $K(P) = \min_i\{K(i) : \text{Turing machine } T_i \text{ computes } P\}$. A string x is typical for a distribution P if the randomness deficiency $\delta(x|P) = \log 1/P(x) - K(x|P)$ is small. The conditional complexity $K(x|P)$ is defined as follows. Say that a function A approximates P if $|A(y,\epsilon) - P(y)| < \epsilon$ for every y and every positive rational ϵ. Then $K(x|P)$ is the minimum length of a program that given every function A approximating P as an oracle, prints x. Similarly, P is c-optimal for x if $K(P) + \log 1/P(x) \leq K(x) + c$.

(a) Show that, for every x and every finite set $S \ni x$ there is a computable probability mass function P with $\lfloor \log 1/P(x) \rfloor = \lfloor \log d(S) \rfloor$, $\delta(x|P) = \delta(x|S) + O(1)$, and $K(P) = K(S) + O(1)$.

(b) Show that there is a constant c such that for every string x, the following holds: For every computable probability mass function P there is a finite set $S \ni x$ such that $\log d(S) < \log 1/P(x) + 1$, $\delta(x|S) \leq \delta(x|P) + K(\lfloor \log 1/P(x) \rfloor) + c$, and $K(S) \leq K(P, \lfloor \log 1/P(x) \rfloor) + c$.

(c) Show that we can restrict consideration to models P such that $\log 1/P(x) \leq l(x) + 1$, in which case the additive terms in Item (b) are $O(\log l(x))$.

Comments. Hint for Item (b): Let $m = \lfloor \log 1/P(x) \rfloor$, that is $2^{-m-1} < P(x) \leq 2^{-m}$. Define $S = \{y : P(y) > 2^{-m-1}\}$. Then, $d(S) < 2^{m+1} \leq 2/P(x)$, which implies the claimed value for $\log d(S)$. To list S it suffices to compute all consecutive values of $P(y)$ to sufficient precision until the combined probabilities exceed $1 - 2^{-m-1}$. That is, $K(S) \leq K(P,m) + O(1)$. Finally, $\delta(x|S) = \log d(S) - K(x|S) < \log 1/P(x) - K(x|S) + 1 = \delta(x|P) + K(x|P) - K(x|S) + 1 \leq \delta(x|P) + K(S|P) + O(1)$. The term $K(S|P)$ can be upper bounded by $K(m) + O(1)$, which implies the claimed bound for $\delta(x|S)$. Hint for Item (c): for every n and every string x of length n, for every P consider P' defined as $P'(x) = (P(x) + 2^{-n})/2$ (the arithmetic mean between P and the uniform distribution on strings of length n). The model P' is not worse than P as an explanation for x, with respect to the considered parameters complexity, deficiency, and likelihood. All results that hold for finite set models extend, up to a logarithmic additive term, to computable probability models. Since the results for the finite set models hold only up to additive logarithmic term anyway, this means that all of them equivalently hold for the model class

of computable probability mass function models. Instead of the data-to-model code length $\log d(S)$ for finite set models, we consider the data-to-model code length $\log 1/P(x)$. The value $\log 1/P(x)$ measures also how likely x is under the hypothesis P. The mapping $x \mapsto P_{\min}$ where $P_{\min}$ minimizes $\log 1/P(x)$ over P with $K(P) \leq i$ is a *constrained maximum likelihood estimator*. This justifies naming of the $h_x(i)$ function the maximum likelihood estimator, in particular its probability-model version $h_x(i) = \min_P\{\log 1/P(x) : P(x) > 0,\ K(P) \leq i\}$, with P a computable probability mass function. The results thus imply that the maximum likelihood estimator always returns a hypothesis with almost minimum randomness deficiency. In classical statistics, unconstrained maximal likelihood is known to perform badly for model selection, because it tends to want the most complex models possible. This is closely reflected in our approach: unconstrained maximization will result in the computable probability distribution of complexity about $K(x)$ that concentrates all probability on x. But the structure function $h_x(i)$ tells us the stochastic properties of data x at all model complexities. Source: these and related results occur in A.K. Shen [*Soviet Math. Dokl.*, 28:1(1983), 295–299; *The Comput. J.*, 42:4(1999), 340–342]; V.V. Vyugin [*SIAM Theory Probab. Appl.*, 32(1987), 508–512]; N.K. Vereshchagin and P.M.B. Vitányi, *Ibid.*

5.5.20. • [27] The model class of *total recursive functions* consists of the set of total recursive functions $p : \{0,1\}^* \to \{0,1\}^*$. The (prefix-) complexity $K(p)$ of a total recursive function p is defined by $K(p) = \min_i\{K(i) : \text{Turing machine } T_i \text{ computes } p\}$. In place of $\log d(S)$ for finite set models, we consider the data-to-model code length $l_x(p) = \min_d\{l(d) : p(d) = x\}$. A string x is typical for a total recursive function p if the randomness deficiency $\delta(x|p) = l_x(p) - K(x|p)$ is small. Here the conditional complexity $K(x|p)$ is defined as the minimum length of a program that given p as an oracle prints x. The *sophistication* of a string x is the complexity $K(p)$ of the minimal sufficient statistic in the total recursive function model class satisfying $K(p)+l_x(d) = K(x)+O(1)$ and $p(d) = x$.

(a) Show that for every x and every finite set $S \ni x$ there is a total recursive function p such that $l_x(p) \leq \log d(S)$, $K(p) = K(S) + O(1)$, and $\delta(x|p) \leq \delta(x|S) + O(1)$.

(b) Show that there is a constant c such that for every string x, the following holds: For every total recursive function p there is a finite set $S \ni x$ with $\log d(S) \leq l_x(p)$, $\delta(x|S) \leq \delta(x|p) + K(l_x(p)) + c$, and $K(S) \leq K(p, l_x(p)) + c$.

(c) Show that the additive terms in Item (b) are $O(\log l(x))$ in the same sense as Exercise 5.5.19, Item (c).

Comments. Hint for Item (b): define $S = \{y : p(d) = y, \ \ l(d) = l_x(p)\}$. Then, $\log d(S) \leq l_x(p)$. To list S it suffices to compute $p(d)$ for every argument of length equal to $l_x(p)$. Hence, $K(S) \leq K(p, l_x(p)) + O(1)$. The upper bound for $\delta(x|S)$ is derived in the same way as in the proof of Exercise 5.5.19, Item (b). All results for finite set models extend, up to a logarithmic additive term, to total recursive function models. Since the results for the finite set models hold only up to additive logarithmic term anyway, this means that all of them equivalently hold for the model class of total recursive functions. The term 'sophistication' was coined by M. Koppel [*Complex Systems*, 1(1987), 1087–1091; pp. 435–452 in: *The Universal Turing Machine: A Half-Century Survey*, R. Herken, ed., Oxford Univ. Press, 1988] in a different, but related, setting of compression and prediction properties of infinite sequences. The issue has sparked the imagination and entered scientific popularization in [M. Gell-Mann, *The Quark and the Jaguar*, W. H. Freeman and Co., New York, 1994] as 'effective complexity' (here 'effective' is apparently used in the sense of 'producing an effect' rather than 'constructive' as is customary in the theory of computation). Source: P.M.B. Vitányi, *IEEE Trans. Inform. Theory*, 52:10(2006), 4617–4626; N.K. Vereshchagin and P.M.B. Vitányi, *Ibid.*

5.5.21. [25] Consider the model class of total recursive functions of Exercise 5.5.20.

(a) Define the structure functions β_x, h_x, λ_x, the sufficient statistic, sufficiency line, and minimal sufficient statistic for a string x in this setting.

(b) The prefix complexity of the minimal sufficient statistic is called the 'sophistication' in Exercise 5.5.20. Show that there are strings x of length n such that the sophistication is at least $n - \log n - 2 \log \log n - O(1)$.

(c) Let x be a string. Show that at every complexity level i, a total recursive function p with $K(p) \leq i$ such that $p(d) = x$ with $l(d)$ minimal is the best model for x in the sense of having minimal randomness deficiency $\delta(x|p)$ as in Exercise 5.5.20 (up to $O(\log n)$ precision).

(d) Investigate in this setting the relations between the structure functions and the shapes that are possible.

(e) Investigate the computability properties of these structure functions and sophistication.

Comments. Source: P.M.B. Vitányi, *Ibid.*; N.K. Vereshchagin and P.M.B. Vitányi, *Ibid.*

5.5.22. [25] In many cases, such as the case of grammatical inference, the data is not a single string but a set of strings, say a subset of $\{0,1\}^n$ for some given n. Develop the theory in Section 5.5 for this case of

multiple data, and indicate the differences with the case of singleton data.

Comments. Source: P. Adriaans and P.M.B. Vitányi, "Approximation of the two-part MDL code," *IEEE Trans. Informat. Theory*, 2008/2009.

5.5.23. • [27] Construct an example of candidate explanations (p_0, S_0) and (p_1, S_1) for data x, with p_i a program computing set $S_i \ni x$ $(i = 0, 1)$, such that (i) the two-part MDL codes satisfy $l(p_1) + \log d(S_1) < l(p_0) + \log d(S_0) - c \log n$ (c a fixed constant); and (ii) the randomness deficiencies satisfy $\delta(x|S_1) > \delta(x|S_0)$.

Comments. The example shows that shorter MDL code does not necessarily mean a better model. The situation is thus as follows: (i) By Theorem 5.5.1 on page 408 the process of finding shorter and shorter MDL codes will in the limit give us the approximately best-fitting model; (ii) since λ_x is upper semicomputable, but not computable, we cannot know when we are close to the limit; and (iii) during the approximation process the randomness deficiency of the candidate models may fluctuate wildly. Compare Lemma 5.5.4 on page 417. Hence, premature termination may result in a worse-fitting model than some models that preceded the terminal one. Source: P. Adriaans and P.M.B. Vitányi, *Ibid.*

5.6 History and References

The material on Epicurus can be found in E. Asmis [*Epicurus Scientific Method*, Cornell University Press, 1984]. The elegant paper "The use of simplicity in induction," by J.G. Kemeny [*Phil. Rev.,* 62(1953), 391–408], contains predecessors to the ideas formulated in this chapter. Occam's razor is usually ascribed to William of Ockham, but is actually a general principle that goes back via his teacher John Duns Scotus (1265–1308) and others at least to Aristotle (384 BC–322 BC) [J. Franklin, *The Science of Conjecture*, Johns Hopkins Univ. Press, Baltimore and London, 2001]. Newton is quoted from *Philosophiae Naturalis Principia Mathematica*, 1687, using a later Dover edition. The quotation of Aristotle in the main text, anticipating Occam's razor, occurs in *Posterior Analytics*, trans. G. R. G. Mure, in *Great Books of the Western World*, Vol. 8 (Chicago: Encyclopedia Britannica, 1952), p. 118. In fact, the principle of parsimony has been commonplace in the history of thought. Aristotle stated it variously again and again [*Physics*, 189a15; *On the Heavens*, 271a33], see [W. Thorburn, The myth of Occam's razor, *Mind*, 27(1977), 5–17]. Claudius Ptolomy (100–178) says, "And in general, we consider it a good principle to explain the phenomena by the simplest hypothesis possible [*Almagest 7*, ch. 1 at 321]. Nicole Oresme (1323–1382) agrees with Ptolomy's sentiment: "And Aristotle says in Chapter Eight of Book I [of *On the Heavens*] that God and nature do nothing without some purpose [*Le Livre du Ciel et du Monde*, 1377, 2 Chapter

25, 532–537, Selection in E. Grant, *A Source Book in Medieval Science*, Cambridge, Mass, 1974, 508–509]. The principle has been advocated and used in various forms by many thinkers between Aristotle and Ockham; see [J. Franklin, *Ibid.*, pp. 137, 145, 241].

Bayes's formula originates from Thomas Bayes's "An essay towards solving a problem in the doctrine of chances" [*Phil. Trans. Roy. Soc.*, 25(1763), 376–398, 54(1764), 298–310, R. Price, ed.] posthumously published by his friend Richard Price. Properly speaking, Bayes's rule as given in the text is not due to Bayes. P.S. Laplace stated Bayes's rule in its proper form and attached Bayes's name to it in *A philosophical essay on probabilities* (1819). In his original memoir, Bayes assumes the uniform distribution for the prior probability and derives $P(H_i|D) = P(D|H_i)/\sum_i P(D|H_i)$. This formula can be derived from Bayes's rule in its present form by setting all $P(H_i)$ equal. Bayes did not state the result in its general form, nor did he derive it through a formula similar to Bayes's rule. The books by B. de Finetti [*Probability, Induction, and Statistics,* John Wiley & Sons, 1972], I.J. Good [*Good Thinking*, University of Minnesota Press, 1983], P.S. Laplace [*Ibid.*], R. von Mises [*Probability, Statistics and Truth*, Macmillan, 1939], and T.L. Fine [*Theories of Probability*, Academic Press, 1973] contain good discussions on the Bayesian and nonBayesian views of inductive reasoning.

The idea of using Kolmogorov complexity in inductive inference, in the form of using a universal prior probability, is due to R.J. Solomonoff [*Inform. Contr.*, 7(1964), 1–22, 224–254]. Solomonoff's original definition of prior probability is problematic through the use of plain Kolmogorov complexity instead of the prefix complexity (as used here). Inductive inference, using $\mathbf{M}$ as universal prior, is also due to Solomonoff [*IEEE Trans. Inform. Theory*, 24(1978), 422–432]; see also [T.M. Cover, "Universal gambling schemes and the complexity measures of Kolmogorov and Chaitin," Tech. Rept. 12, 1974, Statistics Dept, Stanford Univ.]. Solomonoff gives a version, Exercise 5.2.2, Item (a), on page 366, of the estimate in Theorem 5.2.1 of the expected prediction error if one uses $\mathbf{M}$ instead of the actual distribution to predict. Our treatment in Section 5.2 is based on work extending the treatment in the first two editions of this book in [M. Hutter, *IEEE Trans. Inform. Theory*, 49:8(2003), 2061–2067; J. Poland, M. Hutter, *IEEE Trans. Inform. Theory*, 51:11(2005), 3780–3795; M. Hutter, *Universal Artificial Intelligence: Sequential Decisions Based on Algorithmic Probability*, Springer-Verlag, Berlin, 2005]. The topic has spawned an elaborate theory of prediction in both static and reactive unknown environments, based on universal distributions with arbitrary loss bounds (rather than just the logarithmic loss) using extensions and variations of the proof method, represented by the last reference. The dominance of Laplace's measure in Example 5.2.5 on page 356 is based on a more general treatment of sequence predic-

tion for different families of measures in [D. Ryabko, M. Hutter, *Proc. IEEE Int. Symp. Inform. Theory*, 2007, 2346–2350]. Theorem 5.2.2 was suggested to us by Gács, and seems to encapsulate the substance of the problem of inductive inference in statistics and AI in Chapter 5. The full statement of Theorem 5.2.2 in editions 1 and 2 corresponds to the current Theorem 5.2.2, Item (i). The old versions omitted to mention the on-sequence condition, and incorrectly claimed convergence for all μ-random sequences (in the on-sequence sense); see Exercise 5.2.8, Item (a), on page 368. Indeed, convergence holds for a set of μ-measure one, the μ-random sequences constitute a set of μ-measure one, but there may be μ-random sequences without the converge property. Trivially, the set of such sequences has measure zero.

The relation between good prediction and good data compression in Section 5.2.4 and Theorem 5.2.3 is based on [P.M.B. Vitányi and M. Li, *IEEE Trans. Inform. Theory*, 46(2000), 446-464; M. Hutter, *J. Comput. Syst. Sci.*, 72(2006), 95–117]. The current treatment improves and corrects that in editions 1 and 2 of this book. A good introduction to algorithmic probability, universal betting, and its relations to prefix complexity is given in [T.M. Cover and J.A. Thomas, *Elements of Information Theory*, Wiley, 1991].

Our approach in Section 5.2.5 is partially based on [M. Li and P.M.B. Vitányi, *J. Comput. System Sci.*, 44:2(1992), 343–384]. It rests on material in Chapter 4 and Section 5.2. An analysis that $\mathbf{m}$ has related approximative qualities to the actual computable prior like those that $\mathbf{M}$ has in the continuous setting appeared in [T.M. Cover, *Ibid.*; T.M. Cover, pp. 23–33 in: *The Impact of Processing Techniques on Communication*, J.K. Skwirzynski, ed., Martinus Nijhoff Publishers, 1985]. A related direction on prediction and Kolmogorov complexity, using various loss bounds, going by the name of 'predictive complexity,' in a time-limited setting was introduced by V. Vovk [*Problems Inform. Transmission*, 25(1989), 285–292; *Proc. 3rd ACM Conf. Comput. Learning Theory*, 1990, 371–386; *Inform. Comput.*, 96:2(1992), 245–277; *J. Comput. System Sc.*, 56(1998), 153–173; *Int. Statistical Review* 692001, 213–248]. The main technique is the aggregating algorithm. He investigates low-complexity algorithms that give fast probability estimates, or forecasts, of the next bit in a sequence (as opposed to incomputable ones in the Solomonoff procedure). A forecasting algorithm is said to be 'simple' if it has a short description that admits fast computation of the forecasts. There is a universal forecasting algorithm that for every time bound T computes forecasts within time T, and the quality of these forecasts is not much lower than that of any simple forecasting system (of which the descriptional complexity may increase if T increases). Solomonoff's forecasting algorithm is universal in this sense, but incomputable. See also the related work [A. DeSantis, G. Markowsky, and M. Wegman, *Proc.*

29th IEEE Symp. Found. Comput. Sci., 1988, pp. 110–119; N. Littlestone and M. Warmuth, *Proc. 30th IEEE Symp. Found. Comput. Sci.*, 1989, pp. 256–261]. The Gold paradigm was first introduced and studied by E.M. Gold [*Inform. Contr.,* 10(1967), 447–474]. D. Angluin and C. Smith [*Comput. Surveys*, 16(1983), 239–269] and D. Angluin [*Proc. 24th Symp. Theory Comput.*, 1992, pp. 351–369] give the standard survey of this field. The discussion of mistake bounds resulted from conversations with P. Gács, who attributes these and/or related ideas to J.M. Barzdins and R.V. Freivalds.

The pac-learning model was introduced by L.G. Valiant [*Comm. ACM*, 27(1984), 1134–1142], which also contains the learning algorithm for k-DNF. Although similar approaches were previously studied by V.N. Vapnik and A.Ya. Chervonenkis [*Theory Probab. Appl.*, 16:2(1971), 264–280] and J. Pearl [*Int. J. Gen. Syst.*, 4(1978), 255–264] in a somewhat different context, the field of computational learning theory emerged only after the publication of Valiant's paper. [D. Angluin, *Ibid.*] contains a survey of this field. Textbooks are [B.K. Natarajan, *Machine Learning: A Theoretical Approach*, Morgan Kaufmann, 1991; M. Anthony and N. Biggs, *Computational Learning Theory*, Cambridge University Press, 1992; M. Kearns and U. Vazirani, *Introduction to Computational Learning Theory*, MIT Press, 1994]. The important Occam's razor theorem is due to A. Blumer, A. Ehrenfeucht, D. Haussler, M. Warmuth [*Inform. Process. Lett.*, 24(1987), 377–380; *J. ACM*, 35:4(1989), 929–965]. The latter reference also introduced the VC-dimension into the pac-learning field. In the current (slightly stronger) version of the Occam's razor theorem, Theorem 5.3.1, we have replaced $size(h)$ by $C(h)$. This subtle change allows one to handle more-delicate cases when the structure of a concept requires a concept to have unreasonable size, while the Kolmogorov complexity of the concept is very small. This was useful in the computational molecular biology problem of efficiently learning a string [M. Li, *Proc. 31st IEEE Symp. Found. Comput. Sci.*, 1990, pp. 125–134; T. Jiang and M. Li, *Theory of Computing Systems*, 29:4(1996), 387-405]. The Occam's razor theorem is further sharpened in [M. Li, J.T. Tromp, and P.M.B. Vitányi, *Inform. Process. Lett.*, 85:5(2003), 267–274] to obtain better sample complexity than previous length-based and VC-based versions of the Occam's razor theorem, and to achieve a sharper reverse of the Occam's razor theorem than earlier work. The word 'pac,' coined by D. Angluin, stands for 'probably approximately correct.'

Pac-learning under simple distributions in Section 5.3.3 is from M. Li and P.M.B. Vitányi [*SIAM J. Comput.*, 20:5(1991), 911–935]. For the approximation of the set-cover problem we followed [V. Chvátal, *Math. Oper. Res.,* 4:3(1979), 233–235; D.S. Johnson, *J. Comput. System Sci.,* 9(1974), 256–276; L. Lovász, *Discrete Math.,* 13(1975), 383–390]. The definition of ϵ-cover and Lemma 5.3.5 are due to G. Benedek and A. Itai,

Theoret. Comput. Sci., 86:2(1991), 377–390. A drawback of the 'simple learning' approach in this section is that **m** is incomputable. Computable versions of **m** are treated in Section 7.6. Simple pac-learning proceeds the same as before. For example, simple learning of $\log n$-DNF with respect to all $O(n^2)$-time computable distributions P^* can be achieved using a time-bounded version of **m**. This version of universal distribution requires exponential time to compute—but this needs to be done only once for all applications. Once we have computed an (approximation of) a table for this distribution, we can draw examples (even deterministically) according to it in the learning phase as explained in Example 7.6.2 on page 585 for all time and for all concepts to be learned. Simple pac-learning, variations, and relations with query-learning are being developed in a growing body of literature, which we cannot survey here.

The minimum description length principle was introduced by J.J. Rissanen in [*Automatica*, 14(1978), 465–471] inspired by R. Solomonoff's ideas in Section 5.2. A related approach, the minimum message length (MML) approach, is associated with C.S. Wallace and his coauthors. It was introduced earlier and independently (also independently of Solomonoff's work) by C.S. Wallace and D.M. Boulton [*Computing Journal*, 11(1968), 185–195]. This approach is related to taking the negative logarithm of both sides of Bayes's rule, and then replacing the negative logarithms of the probabilities by the associated code-word lengths. In contrast to MDL, the MML method relies on priors. Although different in philosophy and derivation, in practical applications MML and MDL often turn out to be similar. For the fine points in the distinctions we refer to the abundant literature cited below. See also [C.S. Wallace and P.R. Freeman, *J. Royal Stat. Soc. B*, 49:3(1987), 240–252, 252–265, 54:1(1992), 195–209], where the MML method is refined and related to (or inspired by) Solomonoff's work and Kolmogorov complexity, apparently influenced by [G.J. Chaitin, *Scientific American*, 232:5(1975), 47–52]. MDL was developed by J.J. Rissanen in a series of papers [*Ann. Stat.*, 11(1983), 416–431, 14:3(1986), 1080–1100; *Encyclopedia Stat. Sci.*, V, S. Kotz and N.L. Johnson, eds., Wiley, 1986; *J. Royal Stat. Soc.*, 49(1987), 223–239, Discussion 252–265; and as a monograph *Stochastic Complexity and Statistical Inquiry*, World Scientific, 1989]. The MDL code consists of a two-part code comprising a model part and a data part conditional on the model, or as a one-part code called the 'stochastic complexity' of the data. A recent precise estimate of stochastic complexity is given in [J.J. Rissanen, *IEEE Trans. Inform. Theory*, 42:1(1996), 40–47].

Our derivation of MDL from first principles in Section 5.4 follows [M. Li and P.M.B. Vitányi, *J. Comput. Syst. Sci.*, 44(1992), 343–384 and primarily P.M.B. Vitányi and M. Li, *IEEE Trans. Inform. Theory*, 46(2000), 446–464]. Approximations to the optimum MDL code are considered in

[K. Yamanishi, *Proc. 9th ACM Conf. Comput. Learning Theory*, 1996, 99–109; and V. Vovk, *J. Comput. System Sci.*, 55:1(1997), 96-104]. Relations between pac-learning and MDL are explored in [K. Yamanishi, *Machine Learning*, 9(1993), 165–203]. The application of the MDL principle to fitting polynomials, as in Example 5.4.7, was originally considered by J.J. Rissanen in [*Ann. Stat.*, 14(1986), 1080–1100]. Example 5.4.8 is based on [J.R. Quinlan and R. Rivest, *Inform. Comput.,* 80(1989), 227–248]. Decision trees are coded using the MML principle in [Wallace C.S., Patrick J.D., *Machine Learning*, 11(1993), 7–22]. Using the MDL principle, an application of optimal stochastic complexity in coding decision trees is given in [J.J. Rissanen, *J. Comput. Syst. Theory*, 55(1997), 89–95]. The most recent complete treatments of MDL and stochastic complexity in statistical inference are [P.D. Grünwald, I.J. Myung, M.A. Pitt, eds., *Advances in Minimum Description Length; Theory and Applications*, MIT Press, 2005; P.D. Grünwald, *The Minimum Description Length Principle*, MIT Press, 2007], and J.J. Rissanen, *Information and Complexity in Statistical Modeling*, Springer-Verlag, New York, 2007. In the last of these works, the master of MDL does not treat the principle as evident and axiomatic, but formulates it as a statistical modeling version of the Kolmogorov structure function approach as treated in Section 5.5.2.

There is a plethora of applications of the MDL principle in many different areas. Some examples are learning online handwritten characters and robot arm movements [Q. Gao, M. Li,, and P.M.B. Vitányi, *Artificial Intelligence*, 121:1-2(2000), 1–29]; surface reconstruction problems in computer vision [E.P.D. Pednault, *11th IJCAI*, 1989, pp. 1603–1609]; and protein structure analysis in [H. Mamitsuka and K. Yamanishi, *Comput. Appl. Biosciences (CABIOS)*, 11:4(1995), 399–411]. Other applications of the MDL principle range from evolutionary tree reconstruction inference over DNA sequences pattern recognition; smoothing of planar curves to neural network computing [A.R. Barron, pp. 561–576 in: *Nonparametric Functional Estimation and Related Topics*, G. Roussas, ed., Kluwer, 1991]. See also [A.R. Barron and T.M. Cover, *IEEE Trans. Inform. Theory*, 37(1991), 1034–1054; Correction, Sept. 1991]. A survey on MDL and its applications to statistical problems is [A.R. Barron, J.J. Rissanen and B. Yu, *IEEE Trans. Inform. Theory*, 44(1998), 2743–2760].

The MML principle is applied to the alignment of macromolecules in [L. Allison, C.S. Wallace, C.N. Yee *J. Mol. Evol.*, 35(1992), 77–89] and to constructing minimal diagnostic decision trees for medical purposes in [D.P. McKenzie, P.D. McGorry, C.S. Wallace, L.H. Low, D.L. Copolov, B.S. Singh, *Meth. Inform. Medicine*, 32:2(1993), 161–166]. The state of the MML art and its applications is given in the definitive [C.S. Wallace, *Statistical and Inductive Inference by Minimum Message Length*, Springer-Verlag, New York, 2005].

A related method (which we have not discussed in the main text) for discovery by minimum length encoding, applied to molecular evolution and sequence similarity, is the 'algorithmic significance method' of A. Milosavljević and J. Jurka, [*Machine Learning*, 12(1993), 69–87; *Proc. 1st Int. Conf. Intelligent Systems for Molecular Biology*, AAAI Press, 1993, 284–291; *CABIOS*, 9:4(1993), 407–411]. This method seems a straight corollary of the fact demonstrated by Lemma 4.3.5 on page 279 that $\kappa_0(x|P) = \log(\mathbf{m}(x)/P(x))$ is a universal Martin-Löf test for randomness. This shows that $\kappa_0(x|P)$ has the associated properties of Definition 2.4.2 on page 136 and in particular the property of Definition 2.4.1 on page 135 that $\sum\{P(x|l(x) = n) : \kappa_0(x|P) \geq m\} \leq 2^{-m}$, for all n. It is interesting that such infeasible constructions can yet be directly applied in a practical inference setting. The maximum likelihood (ML) principle was introduced by R.A. Fisher in [*Phil. Trans. Royal Soc. London, Ser. A*, 222(1925), 309–368]. The ML principle has greatly influenced the research, as well as practice, in statistics. Jaynes's maximum entropy (ME) principle was introduced by E.T. Jaynes in [*IEEE Trans. Syst. Sci. Cyb.*, SSC-4(1968), 227–241; *Proc. IEEE*, 70(1982), 939–952]. See also [E.T. Jaynes, *Papers on Probability, Statistics, and Statistical Physics*, 2nd edition, Kluwer Academic Publishers, 1989]. It has exerted a great influence on statistics and statistical mechanics. The relationship between the maximum entropy principle and the MDL principle was established by J.J. Rissanen [*Ann. Stat.*, 14(1986), 1080–1100] and by M. Feder [*IEEE Trans. Inform. Theory*, 32(1986), 847–849]. Rissanen [*Ann. Stat.,* 11(1983), 416–431] demonstrates that the ML principle is a special case of the MDL principle.

The role of simplicity, Kolmogorov complexity, and the minimum description length principle in econometric modeling and forecasting is discussed in [H.A. Keuzenkamp and M. McAleer, *The Economic Journal*, 105(1995), 1–21].

Universal probability and Kolmogorov complexity have been applied in cognitive psychology to classical problems of perceptual organization and Gestalt psychology to resolve the central debate between the competing 'likelihood principle' and 'simplicity principle' as in [N. Chater, *Psych. Review*, 103:3(1996), 566–581; N. Chater, P.M.B. Vitányi, *J. Math. Psych.*, 47:3(2003), 346–369; P.A. van der Helm, *Psych. Bull.*, 126:5(2000), 770–800]. Cognitive psychology has a long tradition of applying formal models of simplicity and complexity, exemplified by the early work of E.L.J. Leeuwenberg, *American J. Psych.*, 84:3(1971), 307–349, and predating the advent of Kolmogorov complexity [J.E. Hochberg, E. McAllister, *J. Experimental Psych.*, 46(1953), 361–364]. Not surprisingly, this field has a large and significant literature on applications of Kolmogorov complexity.

Kolmogorov's proposal for nonprobabilistic statistics was presented at a talk for the Bernoulli Society in Tallinn, Estonia, in 1973, and in writing only in [A.N. Kolmogorov, Complexity of Algorithms and Objective Definition of Randomness, *Uspekhi Mat. Nauk* 29:4(1974), 155. (Russian abstract of a talk at Moscow Math. Soc. meeting 4/16/1974)], which we completely reproduce here:

"To each constructive object corresponds a function $\Phi_x(k)$ of a natural number k—the log of minimal cardinality of x-containing sets that allow definitions of complexity at most k. If the element x itself allows a simple definition, then the function Φ drops to 1 even for small k. Lacking such definition, the element is 'random' in a negative sense. But it is positively 'probabilistically random' only when function Φ having taken the value Φ_0 at a relatively small $k = k_0$, then changes approximately as $\Phi(k) = \Phi_0 - (k - k_0)$." [Kolmogorov]

The translation is by L.A. Levin, November 2002. The $\Phi_x(k)$ function is now known as the Kolmogorov structure function and we denote it by $h_x(i)$. In Exercises 5.5.10 and 5.5.11 on page 424, the existence of strings was shown for which essentially the singleton set consisting of the string itself is a minimal sufficient statistic. If the complexity of the minimal sufficient statistic is large, then we call the object nonstochastic. In contrast, if this complexity is small, say logarithmic, then we call the object stochastic since it has a simple satisfactory explanation (sufficient statistic). We depict the distinction in Figure 5.10: Data string x is positive random and stochastic, while data string y is only negative random and nonstochastic.

According to L.A. Levin, then a student of Kolmogorov, the latter told him in 1973 about $h_x(i)$ and asked how it could behave. Levin proved

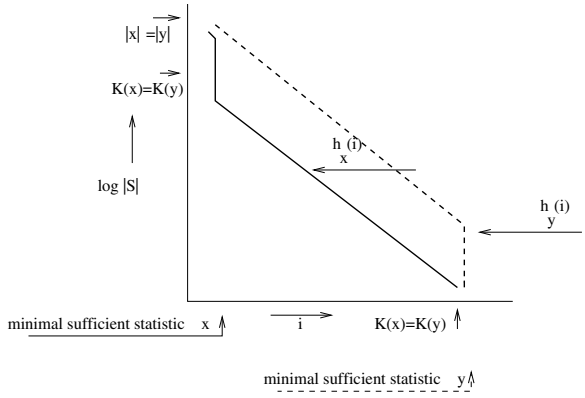


FIGURE 5.10. Positive randomness and negative randomness

that $i+h(i)+O(\log i)$ is monotone but otherwise arbitrary within $O(\sqrt{i})$ accuracy and that it stabilizes on $K(x)$ when i exceeds $I(x : \text{Halting})$. He never published anything on the topic [emails from LL to PV on February 7, 11, 20, 2002]. In writing, (α, β)-stochastic objects were first discussed in [A.N. Kolmogorov, *Lect. Notes Math.*, Vol. 1021, Springer-Verlag, 1983, 1–5; A.N. Kolmogorov, V.A. Uspensky, *Theory Probab. Appl.*, 32(1987), 389–412], and their existence proved, as well as the extension to probability models as in Exercise 5.5.19 on page 428, by A.K. Shen [*Soviet Math. Dokl.*, 28:1(1983), 295–299; *The Comput. J.*, 42:4(1999), 340–342]. V.V. Vyugin established in [*SIAM Theory Probab. Appl.*, 32:3(1987), 508–512; *The Comput. J.*, 42:4(1999), 294–317] for data x and model complexity $i = o(l(x))$ that the randomness deficiency function $\beta_x(i)$ can assume all possible shapes over this limited part of its domain (within the obvious constraints). The snooping curve was proposed by V.V. Vyugin, *Theoret. Comput. Sci.*, 276:1-2(2002), 407–415, who obtained partial results. The definition of algorithmic sufficient statistic in the form we used is apparently due to T.M. Cover [pp. 23–33 in: *The Impact of Processing Techniques on Communications*, J.K. Skwirzynski, ed., Martinus Nijhoff Publishers, 1985; T.M. Cover, P. Gács and R.M. Gray, *Ann. Probab.*, 17(1989), 840–865; T.M. Cover and J.A. Thomas, *Elements of Information Theory*, Wiley, New York, 1991].

Section 5.5 is based on [N.K. Vereshchagin and P.M.B. Vitányi, *IEEE Trans. Inform. Theory*, 50:12(2004), 3265–3290], but Section 5.5.8 follows primarily P. Adriaans and P.M.B. Vitányi, "Approximation of the two-part MDL code," *IEEE Trans. Informat. Theory*, 2008/2009. Interpretation and application of the structure function approach to standard statistical probability models in the MDL setting was begun by J.J. Rissanen, [*Proc. 2002 IEEE Information Theory Workshop*, held in Bangalore, India, IEEE Press, 2002, pp. 98–99; J.J. Rissanen, *Information and Complexity in Statistical Modeling*, Springer-Verlag, New York, 2007]. The complexity of the minimal sufficient statistic of a total recursive function model, as in Exercise 5.5.20 on page 429, was called the 'sophistication' of the object in [M. Koppel, *Complex Systems*, 1(1987), 1087–1091; M. Koppel, pp. 435–452 in: *The Universal Turing Machine: A Half-Century Survey*, R. Herken, ed., Oxford Univ. Press, 1988] in a different, but related, setting of compression and prediction properties of infinite sequences. Our treatment follows [P.M.B. Vitányi, *IEEE Trans. Inform. Theory*, 52:10(2006), 4617–4626; N.K. Vereshchagin and P.M.B. Vitányi, *IEEE Trans. Inform. Theory*, 50:12(2004), 3265–3290].

6

The Incompressibility Method

The incompressibility of random objects yields a simple but powerful proof technique. The incompressibility method is a general-purpose tool and should be compared with the pigeonhole principle or the probabilistic method. Whereas the older methods generally show the existence of an object with the required properties, the incompressibility argument shows that almost all objects have the required property. This follows immediately from the fact that the argument is typically used on a Kolmogorov random object. Since such objects are effectively indistinguishable, the proof holds for all such objects. Each class of objects has an abundance of objects that are Kolmogorov random in it.

The incompressibility method has been successfully applied to solve open problems and simplify existing proofs. We show its versatility and universal applicability by selecting examples from a wide range of applications. This includes combinatorics, random graphs, average-case analysis of Heapsort, Shellsort, routing in communication networks, formal language theory, time bounds on language recognition, string matching, Turing machine time complexity, and circuit complexity.

The method rests on a simple fact: a Kolmogorov random string cannot be compressed. Generally, a proof proceeds by showing that a certain property has to hold for some typical instance of a problem. Since typical instances are difficult to define and often impossible to construct, a classical proof usually involves all instances of a certain class.

By intention and definition, an individual Kolmogorov random object is a typical instance. These are the incompressible objects. Although individual objects cannot be proved to be incompressible in any given finite axiom system, a simple counting argument shows that almost all

M. Li and P.M.B. Vitányi, *An Introduction to Kolmogorov Complexity and Its Applications*,
DOI: 10.1007/978-0-387-49820-1_6, © Springer Science + Business Media, LLC 2008

objects are incompressible, Theorem 2.2.1 on page 117. In a typical proof using the incompressibility method, one first chooses a random object from the class under discussion. This object is incompressible. Then one proves that the desired property holds for this object. The argument invariably says that if the property does not hold, then the object can be compressed. This yields the required contradiction.

Because we are dealing with only one fixed object, the resulting proofs tend to be simple and natural. They are natural in that they supply rigorous analogues for our intuitive reasoning. In many cases a proof using the incompressibility method implies an average-case result, since almost all strings are incompressible.

6.1 Three Examples

The proposed methodology is best explained by example. The first example contains one of the earliest lower-bound proofs by the incompressibility argument. The second example shows how to use incompressibility to analyze the average-case complexity of an algorithm. The third example was first proved using an incompressibility argument.

6.1.1 Computation Time of Turing Machines

Consider the basic Turing machine model in Figure 6.1. This is the model explained in Section 1.7. It has a finite control and a single tape, serving as input tape, output tape, and work tape. The tape is a one-way infinite linear array of squares, each of which can hold a symbol from a finite, nonempty alphabet. The leftmost square is the initial square.

There is a two-way read/write head on the tape. The head movement is governed by the state of the finite control and the symbol in the tape square under scan. In one step, the head may print another symbol in the scanned tape square, move one square left or right (or not move at all), and the state of the finite control may change. At the start of the computation, the input occupies the initial tape segment (one symbol per square) and is delimited by a distinguished end marker. Initially, the tape head is on the leftmost tape square, and the finite control is in a distinguished initial state. If $x = x_1 \dots x_n$, then $x^R = x_n \dots x_1$.

Definition 6.1.1 Each pair of adjacent squares on the tape is separated by an intersquare boundary. Consider an intersquare boundary b and the sequence of states of T's finite control at the steps when the head crosses b, first from left to right, and then alternatingly in both directions. This ordered sequence of states is the *crossing sequence* at b.

Lemma 6.1.1 *A Turing machine of the model above requires order n^2 steps to recognize $L = \{xx^R : x \in \{0,1\}^*\}$.*

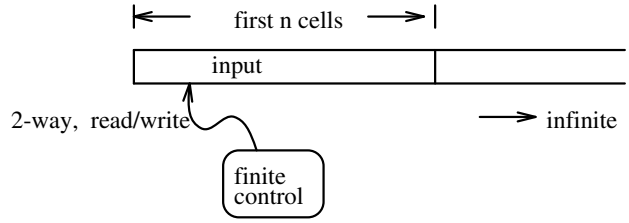


FIGURE 6.1. Single-tape Turing machine

Proof. By way of contradiction, assume that there is a Turing machine T of the above model that recognizes L in $o(n^2)$ steps. Without loss of generality assume that the machine halts with the tape head scanning the input end marker.

Fix a string x of length n with $C(x|T,n) \geq n$. Such strings exist by Theorem 2.2.1 on page 117. Consider the computation of T on $x0^{2n}x^R$. Let $l(T)$ and $l(c.s.)$ denote the lengths of the descriptions of T and a crossing sequence *c.s.*, respectively. If each crossing sequence associated with a square in the 0^{2n} segment in the middle is longer than $\frac{1}{2}nl(T)$, then T uses at least $n^2/l(T)$ steps. Otherwise there is a crossing sequence of length less than $n/2l(T)$. Assume that this is the crossing sequence *c.s.* associated with the square in position c_0. This *c.s.* is completely described by at most $\frac{1}{2}n$ bits. Using *c.s.*, one can reconstruct x by exhaustively checking all binary strings of length n.

For each candidate binary string y of length n, put $y0^{2n}$ on the leftmost $3n$-length initial segment of the input tape and simulate T's computation from its initial configuration. Each time the head moves from square c_0 to its right neighbor, skip the part of the computation of T with the head right of c_0, and resume the computation starting from the next state q in *c.s.* with the head scanning square c_0.

Suppose that in the computation with y, each time the head moves from square c_0 to its right neighbor, the current state of T is the correct next state as specified in *c.s.* Then T accepts input $y0^{2n}x^R$. Namely, the computation to the right of square c_0 will simply be identical to the computation to the right of square c_0 on input $x0^{2n}x^R$. Since T halts with its head to the right of square c_0, it must either accept both $y0^{2n}x^R$ and $x0^{2n}x^R$ or reject them both. Since T recognizes L, we must have $y = x$. Therefore, given *c.s.*, we can reconstruct x by a fixed program from the data T, n, and *c.s.* This means that

$$C(x|T,n) \leq l(c.s.) + O(1) \leq \frac{1}{2}n + O(1),$$

which contradicts $C(x|T,n) \geq n$, for large n. □

6.1.2 Adding Fast—On Average

In computer architecture design, efficient design of adders directly affects the length of the CPU clock cycle. Fifty years ago, Burks, Goldstine, and von Neumann obtained a $\log n$ expected upper bound on the longest carry sequence involved in the process of adding two n-bit binary numbers. This property suggests the design for an efficient adder hardware. We give a simple analysis using the incompressibility method. Let x and y be two n-bit binary numbers and let $\oplus$ denote the bitwise exclusive-or operator. The following algorithm adds x and y.

Step 1. $S := x \oplus y$ (add bitwise, ignoring carries); $C :=$ carry sequence;

Step 2. **While** $C \neq 0$ **do**
$S := S \oplus C$;
$C :=$ new carry sequence.

Let us call this the *no-carry adder* algorithm. The expected $\log n$ upper bound on carry sequence length implies that the algorithm runs in $1 + \log n$ expected rounds (Step 2). This algorithm is on average the most efficient addition algorithm currently known. But it takes n steps in the worst case. On average, it is exponentially faster than the trivial linear-time ripple-carry adder, and it is two times faster than the well-known carry-lookahead adder. In the ripple-carry adder, the carry ripples from right to left, bit by bit, and hence it takes $\Omega(n)$ steps to compute the sum of two n-bit numbers. The carry-lookahead adder is used in nearly all modern computers; it is based on a divide-and-conquer algorithm that adds two n-bit numbers in $1 + 2\log n$ steps. We give an easy proof of the $1 + \log n$ average-case upper bound, using the incompressibility method.

Lemma 6.1.2 *The no-carry adder algorithm has an average running time of at most* $1 + \log n$.

Proof. Assume that both inputs x and y have length $l(x) = l(y) = n$, with the lower-order bits on the right. If the computation takes precisely t steps (Step 2 loops t times), then some thinking shows that there exists a u such that x and y can be written as

$$x = x'bu1x'', \quad y = y'b\neg u1y'',$$

where $l(u) = t - 1$, $l(x') = l(y')$, b is 0 or 1, and $\neg u$ is the bitwise complement of u. Therefore, x can be described using y, n, and a program q of $O(1)$ bits to reconstruct x from the concatenation of

- the position of u in y encoded in exactly $\log n$ bits (padded with 0's if needed); and

- the literal representation of $x'x''$.

Since the concatenation of the two strings has length $n - t - 1 + \log n$, the value t can be deduced from n and this length. Therefore, $t + 1$ bits of x are saved at the cost of adding $\log n$ bits. (For $x' = \epsilon$, bit b may not exist. But then the algorithm also does not execute the last step because of overflow.)

This shows that $C(x|n, y, q) \leq n - t - 1 + \log n$. Hence, for each x with $C(x|n, y, q) = n - i$, the computation must terminate in at most $\log n + i - 1$ steps. By simple counting as in Theorem 2.2.1 on page 117, there are at most 2^{n-i} strings x of length n with Kolmogorov complexity $C(x|n, y, q) = n - i$. There are at most 2^{n-i} programs of length $n - i$, and hence at most 2^{n-i} strings x with $C(x|n, y, q) = n - i$. Let p_i denote the fraction of x's of length $l(x) = n$ satisfying $C(x|n, y, q) = n - i$. Then, $p_i \leq 2^{-i}$ and $\sum_i p_i = 1$. Hence, averaging over all x's (by having i range from 1 to n) with y fixed, the average computation time for each y is bounded above by

$$\sum_{i=2-\log n}^{n} p_i(i - 1 + \log n) = \sum_{i=2-\log n}^{n} p_i(i-1) + \sum_{i=2-\log n}^{n} p_i \log n$$
$$\leq \log n + \sum_{i=1}^{\infty} \frac{i-1}{2^i} = 1 + \log n.$$

Because this holds for every y, this is also the average running time of the algorithm. □

6.1.3 Boolean Matrix Rank

The rank of a matrix R is the least integer k such that each row of R can be written as a linear sum of k fixed rows. These k rows are linearly independent, which means that no row can be written as a linear sum of the others. Our problem is to show the existence of a Boolean matrix with all submatrices of high rank.

Such matrices were used to obtain an optimal lower bound $TS = \Omega(n^3)$ time–space tradeoff for multiplication of two n by n Boolean matrices on random access machines (T = time and S = space). Even with integer entries, it is difficult to construct such a matrix. There is no known construction with Boolean values.

Let $GF(2)$ be the Galois field over the two elements $0, 1$, with the usual Boolean multiplication and addition: $0 \times 0 = 0 \times 1 = 1 \times 0 = 0$, $1 \times 1 = 1$, $1 + 0 = 0 + 1 = 1$, and $1 + 1 = 0 + 0 = 0$.

Lemma 6.1.3 *Let $n, r, s \in \mathcal{N}$ with $2 \log n \leq r, s \leq \frac{1}{4}n$ and s even. For each n there is an $n \times n$ matrix over $GF(2)$ such that every submatrix of s rows and $n - r$ columns has at least rank $s/2$.*

Proof. Fix a binary string x of length n^2, with $C(x) \geq n^2$. This is possible by Theorem 2.2.1 on page 117. Arrange the bits of x into a square matrix R, one bit per entry in, say, row-major order. We claim that this matrix R satisfies the requirement.

Assume by way of contradiction that this were not true. Consider a submatrix of R of s rows and $n-r$ columns, with r, s as in the condition in the lemma. There are at most $(s/2)-1$ linearly independent rows in it. Therefore, each of the remaining $(s/2)+1$ rows can be expressed as a linear sum of the other $(s/2)-1$ rows. This can be used to describe R by the following items:

- The characteristic sequence of the $(s/2)-1$ independent rows out of the s rows of R in s bits.
- A list of the $(s/2)-1$ linearly independent rows in $((s/2)-1)(n-r)$ bits.
- List the remainder of $(s/2)+1$ rows in order. For each row give only the Boolean coefficients in the assumed linear sum. This requires $((s/2)-1)((s/2)+1)$ bits.

To recover x, we need only the additional items below:

- A description of this discussion in $O(1)$ bits.
- The values of n, r, s in self-delimiting form in $3\log n + 6\log\log n + 3$ bits. For large n, this is at most $4\log n$ bits including the $O(1)$ bits above.
- R without the bits of the submatrix in row-major order in $n^2-(n-r)s$ bits.
- The indices of the columns and rows of the submatrix, in $(n-r)\log n + s\log n$ bits.

To ensure unique decodability of these binary items, we concatenate them as follows: First list the self-delimiting descriptions of n, r, s, then append all other items in a fixed order. The length of each of the items can be calculated from n, r, s. Altogether, this is an effective description of x, a number of bits of at most

$$n^2 - (n-r)s + (n-r)\log n + s\log n + \left(\frac{s}{2}-1\right)(n-r) + \left(\frac{s}{2}-1\right)\left(\frac{s}{2}+1\right) + s + 4\log n.$$

For large n, this quantity drops below n^2. But we have assumed that $C(x) \geq n^2$, which yields the required contradiction. □

A proof obtained by the incompressibility method usually implies that the result holds for almost all strings, and hence it holds for the average case complexity.

Exercises

6.1.1. [26/M30] Let the Turing machine in Section 6.1.1 be probabilistic, which means that the machine can flip a fair coin to determine its next move.

(a) Assume that the machine is not allowed to err. Prove that such a machine still requires on average order n^2 steps to accept the palindrome language $L = \{xx^R : x \in \{0,1\}^*\}$. The average is taken over the uniform distribution of all inputs of length n and all coin tosses of the algorithm.

(b) Assume that the machine is allowed to err with probability ϵ. Show that the palindrome language L can be accepted in worst-case time $O(n \log n)$ by such a machine.

Comments. Hint for Item (a): use the symmetry of information theorem, Theorem 2.8.2, on page 190. With high uniform probability, a sequence of random coin tosses r and a random input x are random relative to each other. Thus, the deterministic argument given in the proof of Section 6.1.1 proceeds as before with r as an extra input or oracle. Hint for Item (b): generate random primes of size $\log n$ and check whether both sides are the same modulo these primes. Repeat this process to guarantee high accuracy. Source: [R. Freivalds, *Information Processing 77, Proc. IFIP Congress 77*, North-Holland, Amsterdam, 1977, 839–842].

6.1.2. [10] (Converting NFA to DFA) A *deterministic finite automaton* (DFA) A has a finite number of states, including a distinguished start state and some distinguished accepting states. At every step, A reads the next input symbol and changes its state according to the current state and the input symbol. If A has more than one alternative at some step, then A is nondeterministic (NFA). If A is in an accepting state when it reads a distinguished end marker, then A accepts the input. Otherwise A rejects it. It is well known that every NFA can be converted to a DFA. Use an incompressibility argument to prove that there exists an NFA with n states such that the smallest DFA accepting the same language has $\Omega(2^n)$ states.

Comments. Hint: use $L_k = \{x :$ the kth bit of x from the right is $1\}$. This problem can also be solved by a simple counting argument.

6.1.3. [15] Give a simple algorithm that multiplies two $n \times n$ Boolean matrices in $O(n^2)$ average time under uniform distribution. Use an incompressibility argument to show the time complexity.

Comments. Source: the original proof, without incompressibility, is given in [P.E. O'Neil, E.J. O'Neil, *Inform. Contr.*, 22:2(1973), 132–138].

6.2 High-Probability Properties

The theory of random individual objects, Sections 2.4 and 2.5, tells us that there is a close relation between high-probability properties and properties of incompressible objects. For infinite binary sequences $\omega \in \{0,1\}^\infty$ and λ the uniform (coin-toss) measure, classic probabilistic laws are formulated in global form by

$$\lambda\{\omega : A(\omega)\} = 1,$$

where $A(\omega)$ is some formula expressing some property. In contrast, in the algorithmic theory of random individual objects, the corresponding law is expressed in local form by

$$\text{if } \omega \text{ is random then } A(\omega) \text{ holds.}$$

The classical probabilistic laws as in the first displayed equation are uncountable. The properties tested by Martin-Löf tests to determine randomness as in the second displayed equation are the effectively testable properties and hence countable. Thus, there are classical probabilistic laws that do not hold in the pointwise sense of the second equation. On the other hand, a pointwise algorithmic law implies the corresponding classical probabilistic law: if the second displayed equation holds for formula A, then also the first displayed equation holds for A (by Theorem 2.5.3 on page 151). How do things work out quantitatively for finitely many finite objects? To fix our thoughts let us look at a simple example.

First we recall the notion of randomness deficiency of Section 2.2.1 on page 120. The randomness deficiency of an element in a certain class of objects is the difference between that element's Kolmogorov complexity and the maximal Kolmogorov complexity of an object in the class (typically the logarithm of the cardinality of the class). Formally, if x is an element of a finite set of objects S, then by Theorem 2.1.3 on page 111 we have $C(x|S) \leq l(d(S)) + c$ for some c independent of x but possibly dependent on S. The *randomness deficiency* of x *relative* to S is defined as $\delta(x|S) = \log d(S) - C(x|S)$.

Example 6.2.1 Let $G = (V, E)$ be a graph on n nodes where every pair of nodes is or is not connected by an edge according to the outcome of a fair coin flip. The probability that a particular node is isolated (has no incident edges) is $1/2^{n-1}$. Therefore, the probability that some node is isolated is at most $n/2^{n-1}$. Consequently, the probability that the graph has no isolated nodes is at least $1 - n/2^{n-1}$.

Using the *incompressibility method*, the proof that random graphs have this nonisolation property with high probability is as follows: Each labeled undirected graph $G = (V, E)$ on n nodes can be described by giving a characteristic sequence χ of the lexicographic enumeration of

$V \times V$ without repetition, namely, $\chi = \chi_1\chi_2 \ldots \chi_e$ with $e = \binom{n}{2}$ and $\chi_i = 1$ if the ith enumerated edge is in E and 0 otherwise. There are as many labeled n-node graphs as there are such characteristic sequences. Therefore, we can consider graphs G having randomness deficiency at most $\delta(n)$,

$$C(G|n) \geq \binom{n}{2} - \delta(n). \tag{6.1}$$

Assume by way of contradiction that there is an isolated node i. Add its identity in $\log n$ bits to the canonical description of G, and delete all $n-1$ bits indicating presence or absence of edges incident on i, saving $n-1$ bits. From the new description we can reconstruct G given n. Then the new description length cannot be smaller than $C(G|n)$. Substitution shows that $\delta(n) \geq n - 1 - \log n$. The number of programs of length at most $\binom{n}{2} - \delta(n)$ shows that at most a fraction of $2^{-\delta(n)}$ of all n-node graphs contain an isolated node. Hence, the nonisolation property for n-node graphs holds with probability at least $1 - n/2^{n-1}$. ◇

For every finite class of finite objects there is a close relation between properties that hold with high probability and properties that hold for objects with small randomness deficiency: the almost incompressible ones. However, the properties and the sets of objects concerned are not identical and should be carefully distinguished. In fact, the following distinctions also indicate in which cases use of which method is preferable:

- In the probabilistic method, the subset of objects on which the probability of a property is based is the subset of *all* objects satisfying that property. As an example, consider the nonisolation property of labeled graphs again. The graphs satisfying this property include the complete graph on n nodes, the star graph on n nodes, and the binary hypercube on n nodes, provided n is a power of 2. These graphs are certainly not incompressible or random and in fact have complexity $O(1)$ given n.

- If each object with suitable randomness deficiency at most $\delta(n)$ has a certain property, then every such object is included in the subset of objects on which the high probability of the property is based.

- If we prove that properties P and Q each hold with probability at least $1 - \epsilon$ with the probabilistic method, then we can conclude that properties P and Q *simultaneously* hold with probability at least $1 - 2\epsilon$. In contrast, if both properties P and Q hold separately for objects with randomness deficiency at most $\delta(n)$, then they vacuously also hold simultaneously for objects with randomness deficiency $\delta(n)$.

More general, suppose that every high-probability property separately holds for an overwhelming majority (say, at least a $(1-1/n)$th fraction) of all objects. Now consider a situation of n different properties each of which holds for a $(1-1/n)$th fraction. Since possibly the subsets on which the different properties fail may be disjoint, possibly their union may constitute the set of all objects. Therefore it is possible that no object at all possesses all the high-probability properties simultaneously.

In contrast, if we prove properties separately for objects with randomness deficiency at most $\delta(n)$, then all these properties hold simultaneously for each of these objects.

These considerations show that high-probability properties and incompressibility properties are not a priori the same. However, we shall prove that they almost coincide under mild conditions on the properties considered. In fact, the objects with a certain small randomness deficiency satisfy all *simply described* properties that hold with high probability. This is not just terminology: If $\delta(x|S)$ is small enough, then x satisfies *all* properties of low Kolmogorov complexity that hold with high probability for the elements of S. To be precise: Consider strings of length n and let S be a subset of such strings. A *property* P represented by S is a subset of S, and we say that x satisfies property P if $x \in P$. (The lemma below can also be formulated in terms of probabilities instead of frequencies if we are talking about a probabilistic ensemble S.)

Lemma 6.2.1 *Let $S \subseteq \{0,1\}^n$ and let $\delta : \mathcal{N} \to \mathcal{N}$ be such that $\delta(n) \leq \log d(S)$.*

(i) *If P is a property satisfied by all $x \in S$ with $\delta(x|S) \leq \delta(n)$, then P holds for a fraction of at least $1 - 1/2^{\delta(n)}$ of the elements in S.*

(ii) *Let P be a property that holds for a fraction of at least $1 - 1/2^{\delta(n)}$ of the elements of S. Then there is a constant c such that P holds for every $x \in S$ with $\delta(x|S) \leq \delta(n) - K(P|S) - c$.*

Proof. (i) There are only $\sum_{i=0}^{\log d(S)-\delta(n)} 2^i$ programs of length not greater than $\log d(S) - \delta(n)$ and there are $d(S)$ elements in S.

(ii) Suppose, by way of contradiction, that P does not hold for an object $x \in S$ whose randomness deficiency satisfies $\delta(x|S) \leq \delta(n) - K(P|S) - c$. Then we can reconstruct x from a description of S and P, and x's index j in an effective enumeration of all objects in $S - P$. There are at most $d(S)/2^{\delta(n)}$ such objects by assumption. Therefore there is a constant c_1 such that

$$K(x|S,P) \leq \log j + c_1 \leq \log d(S) - \delta(n) + c_1.$$

Using the contradictory assumption, we obtain $K(x|P,S) \leq K(x|S) - K(P|S) - c + c_1$. Also, trivially, there is a constant c_2 such that $K(x|S) \leq K(x|P,S) + K(P|S) + c_2$. Therefore, $c \leq c_1 + c_2$. Choosing $c > c_1 + c_2$ we have the desired contradiction. □

These results mean that if we want to establish that a property holds with *high probability* or for objects with *small randomness deficiency*, then it suffices to establish either one to prove both. Moreover, the small-randomness-deficiency objects satisfy all highly probable simple properties simultaneously.

If a property P satisfies $K(P|n) = O(1)$, that is P is recursive in n, then P is simple. An example of such a property is the upper bound of $2 \log n$ on the size of the largest complete subgraph in a graph on n nodes with randomness deficiency $\delta(n) = \log n$ in Equation 6.1. The quantity $K(P|n)$ grows unboundedly for more complex properties that require us to describe a number of parameters that grows unboundedly as n grows unboundedly. An example is the property of containing a labeled subgraph H on $\log n$ nodes with $K(H|n) \geq \binom{\log n}{2}$.

Corollary 6.2.1 (i) The strings of length n of randomness deficiency at most $\delta(n)$ possess all properties P that hold with probability at least $1 - 2^{-\delta(n) - K(P|n) - O(1)}$.

(ii) All recursive properties P with $K(P|n) = O(1)$, each of which holds separately for strings of length n with probability tending to 1 as n grows unboundedly, hold simultaneously with probability tending to 1 as n grows unboundedly.

These results mean that if we want to establish that a property holds with *high probability* or for objects with *high Kolmogorov complexity* (which equals small randomness deficiency in the set of all such objects of the same length), then it suffices to establish either one to prove both. Moreover, the high-Kolmogorov-complexity objects satisfy all highly probable simple properties simultaneously.

6.3 Combinatorics

Combinatorial properties are traditionally established by counting arguments or by the probabilistic method. Probabilistic arguments are usually aimed at establishing the existence of an object in a nonconstructive sense. It is ascertained that a certain member of a class has a certain property without actually exhibiting that object. Usually, the method proceeds by exhibiting a random process that produces the object with positive probability. Alternatively, a quantitative property is determined from a bound on its average in a probabilistic situation.

We demonstrate the utility of the incompressibility method in combinatorial theory on several examples. The general pattern is as follows:

When we want to prove a certain property of a group of objects (such as graphs), we first fix an incompressible instance of the object, justified by Theorem 2.2.1 on page 117. It is always a matter of using the assumed regularity in this instance to compress the object to reach a contradiction.

6.3.1 Transitive Tournament

A *tournament* is defined to be a complete directed graph. That is, for every pair of nodes i and j, exactly one of the edges (i,j) and (j,i) is in T. The nodes of a tournament can be viewed as players in a game tournament. If (i,j) is in T, we say player j dominates player i. We call T *transitive* if $(i,j),(j,k)$ in T implies (i,k) in T.

Let $\Gamma = \Gamma_n$ be the set of all tournaments on $N = \{1,\ldots,n\}$. Given a tournament $T \in \Gamma$, fix a standard encoding $E : T \to \{0,1\}^{n(n-1)/2}$, one bit for each edge. The bit for edge (i,j) is set to 1 if $i < j$ (j dominates i) and 0 otherwise. There is a one-to-one correspondence between the members of Γ and the binary strings of length $n(n-1)/2$.

Let $v(n)$ be the largest integer such that every tournament on N contains a transitive subtournament on $v(n)$ nodes.

Theorem 6.3.1 $v(n) \leq 1 + \lfloor 2 \log n \rfloor$.

Proof. For $n = 1$, trivially $v(n) = 1$. Therefore, we can assume $n \geq 2$. Fix $T \in \Gamma$ such that

$$C(E(T)|n,p) \geq n(n-1)/2,$$

where p is a fixed program that on input n and $E'(T)$ (below) outputs $E(T)$. Let S be the transitive subtournament of T on $v(n)$ nodes. We try to compress $E(T)$, to an encoding $E'(T)$, as follows:

1. Prefix the list of nodes in S in order of dominance to $E(T)$, every node using $\lfloor \log n \rfloor$ bits, by encoding integers $n = 2,3,4,\ldots$ by binary strings $0,1,00,\ldots$, as in Exercise 1.4.2 on page 14. This adds $v(n)\lfloor \log n \rfloor$ bits.
2. Delete all redundant bits from the $E(T)$ part, representing the edges between nodes in S, saving $v(n)(v(n)-1)/2$ bits.

Then

$$l(E'(T)) = l(E(T)) - \frac{v(n)}{2}(v(n) - 1 - 2\lfloor \log n \rfloor).$$

Given n, the program p reconstructs $E(T)$ from $E'(T)$. Therefore,

$$C(E(T)|n,p) \leq l(E'(T)).$$

The three displayed equations are true only when $v(n) \leq 1 + 2\lfloor \log n \rfloor$. Since it is easy to verify that $2\lfloor \log n \rfloor = \lfloor 2 \log n \rfloor$ for all $n \geq 1$, this proves the theorem. □

The general idea used in the incompressibility proof of Theorem 6.3.1 is the following: If every tournament contains a large transitive subtournament, or any other regular property for that matter, then also a tournament T of maximal complexity contains one. But the regularity induced by too large a transitive subtournament can be used to compress the description of T to below its complexity, leading to the required contradiction.

P. Stearns showed by induction that $v(n) \geq 1 + \lfloor \log n \rfloor$. This is the first problem illustrating the probabilistic method in [P. Erdős and J.H. Spencer, *Probabilistic Methods in Combinatorics*, Academic Press, 1974]. They collected many combinatorial properties accompanied by elegant proofs using probabilistic arguments. The thrust was to show how to replace counting arguments by pleasant and short probabilistic arguments. To compare the incompressibility method, we include their proofs of Theorem 6.3.1 by counting and probabilistic methods.

Proof. (*by counting*) Let $\Gamma = \Gamma_n$ be the class of all tournaments on $\{1, \ldots, n\}$ and $\Gamma' =$ the class of tournaments on $\{1, \ldots, n\}$ that contain a transitive subtournament on $v = 2 + \lfloor 2 \log n \rfloor$ players. Then

$$\Gamma' = \bigcup_A \bigcup_\sigma \Gamma_{A,\sigma},$$

where $A \subseteq \{1, \ldots, n\}$, $d(A) = v$, σ is a permutation on A, and $\Gamma_{A,\sigma}$ is the set of T such that $T|A$ is generated by σ. If $T \in \Gamma_{A,\sigma}$, the $\binom{v}{2}$ games of $T|A$ are determined. Thus,

$$d(\Gamma_{A,\sigma}) = 2^{\binom{n}{2} - \binom{v}{2}},$$

and by elementary estimates

$$d(\Gamma') \leq \sum_{A,\sigma} 2^{\binom{n}{2} - \binom{v}{2}} = \binom{n}{v} v! 2^{\binom{n}{2} - \binom{v}{2}} < 2^{\binom{n}{2}} = d(\Gamma).$$

Thus, $\Gamma - \Gamma' \neq \varnothing$. That is, there exists $T \in \Gamma - \Gamma'$ not containing a transitive subtournament on v players. □

Proof. (*by the probabilistic method*) Assume the same notation and suppositions as in the proof by counting. Let $\mathbf{T} = T_n$ be a random variable. Its values are the members of Γ, where for every $T \in \Gamma$, $\Pr(\mathbf{T} = T) = 2^{-\binom{n}{2}}$. That is, all members of Γ are equally probable values of $\mathbf{T}$. Then

the probability that an outcome T of $\mathbf{T}$ contains a transitive subtournament on v players is at most

$$\sum_A \sum_\sigma \Pr(\mathbf{T}|A \text{ generated by } \sigma) = \binom{n}{v} v! 2^{-\binom{v}{2}} < 1.$$

Thus, some value T of $\mathbf{T}$ does not contain a transitive subtournament on v players. □

6.3.2 Tournament with k-Dominators

Tournament T has *property* $S(k)$ if for every subset A of k nodes (players) there is a node (player) in $N - A$ that dominates (beats) all nodes (players) in A. Let $s(k)$ be the minimum number of nodes (players) in a tournament with property $S(k)$.

Theorem 6.3.2 $s(k) \leq 2^k k^2 (\ln 2 + o(1))$.

Proof. Choose $n = 2^k k^2 (\ln 2 + o(1))$. Assume the notation of the previous example. Select T on n nodes such that

$$C(E(T)|n, k, p) \geq n(n-1)/2,$$

where p is a fixed program to compute $E(T)$ from $E'(T)$ (given below) and n, k. By way of contradiction, assume that $S(k)$ is false for T. Fix a set A of k nodes of T with no common dominator in $N - A$. Describe T as follows by a compressed description $E'(T)$:

- List the nodes in A first, using $\lfloor \log n \rfloor$ bits each. As before, code integers $n = 2, 3, 4, \ldots$ by strings $0, 1, 00, \ldots$.
- List $E(T)$ with bits representing edges between $N - A$ and A deleted (saving $(n-k)k$ bits).
- Code the edges between $N - A$ and A. From every $i \in N - A$, there are $2^k - 1$ possible ways of directing edges to A, in total $t = (2^k - 1)^{n-k}$ possibilities. To encode the list of these edges, $\lfloor \log t \rfloor$ bits suffice.

This shows that $C(E(T)|n, k, p) \leq l(E'(T))$. For large k, $l(E'(T)) < n(n-1)/2$ bits, which is a contradiction. □

6.3.3 Ramsey Numbers

The previous examples demonstrate a general principle that a random graph (or its complement) cannot contain too large a subgraph that is easily describable. We apply the incompressibility method to obtain a lower bound on Ramsey numbers. A *clique* of a graph is a complete subgraph of that graph. The Ramsey number $r(k, k)$ is the least integer such that for every graph G of size $r(k, k)$, either G or G's complement contains a clique of size k. P. Erdős proved in 1947, using the probabilistic method, the following result:

Theorem 6.3.3 $r(k,k) \geq k2^{k/2}\left(\frac{1}{e\sqrt{2}} - o(1)\right)$.

Proof. To describe a clique (or empty subgraph) of size k in a graph G of $r(k,k)$ vertices we need $\log\binom{r(k,k)}{k} \leq k\log r(k,k) - \log k!$ bits. Choose G to be incompressible. Then we must have $k\log r(k,k) - \log k! \geq k(k-1)/2$, since otherwise we can compress G as in the proof of Theorem 6.3.1, Using Stirling's formula we obtain $k! \approx k^k e^{-k}\sqrt{2\pi k}$, and a simple calculation shows the theorem. □

6.3.4 Coin-Weighing Problem

A family $\mathcal{D} = \{D_1, D_2, \ldots, D_j\}$ of subsets of $N = \{1, 2, \ldots, n\}$ is called a *distinguishing family* for N if for every two distinct subsets M and M' of N there exists an i $(1 \leq i \leq j)$ such that $d(D_i \bigcap M)$ is different from $d(D_i \bigcap M')$. Let $f(n)$ denote the minimum of $d(\mathcal{D})$ over all distinguishing families for N. To determine $f(n)$ is commonly known as the *coin-weighing problem*. It is known that

$$f(n) = \frac{2n}{\log n} + O\left(\frac{n\log\log n}{\log^2 n}\right).$$

The $\leq$ side of this equation, with small-o instead of big-O, was independently established by [B. Lindström, *Canad. Math. Bull.*, 8(1965), 477–490] and [D.G. Cantor, W.H. Mills, *Canad. J. Math.*, 18(1966), 42–48]. The $\geq$ side, Theorem 6.3.4, was established by P. Erdős and A. Rényi [*Publ. Hungar. Acad. Sci.*, 8(1963), 241–254], L. Moser [*Combinatorial Structures and Their Applications*, Gordon and Breach, 1970, pp. 283–384], and N. Pippenger [*J. Combinat. Theory, Ser. A*, 23(1977), 99–104] using probabilistic and information theory methods.

We prove the $\geq$ side using the incompressibility method. Encode every subset M of N by $E(M) \in \{0,1\}^n$ such that the ith bit of $E(M)$ is 1 if i is in M, and 0 otherwise.

Theorem 6.3.4 $f(n) \geq (2n/\log n)[1 + O(\log\log n/\log n)]$.

Proof. Choose M such that

$$C(E(M)|\mathcal{D}) \geq n. \tag{6.2}$$

Let $d_i = d(D_i)$ and $m_i = d(D_i \bigcap M)$. Let s_i be the subsequence of $E(M)$ selected from the positions corresponding to 1's in $E(D_i)$. Thus, $l(s_i) = d_i$ and the number of 1's in s_i is precisely m_i. Moreover,

$$C(s_i) \geq d_i - O(\log i),$$

since we can use $\mathcal{D}$, i, the shortest program for s_i, and $E(M)$ minus the bits in s_i to reconstruct $E(M)$.

By Equation 2.3 on page 167, the value m_i is within range $d_i/2 \pm O(\sqrt{d_i \log i})$. Therefore, given d_i, every m_i can be described by its discrepancy with $d_i/2$, which gives

$$C(m_i|D_i) \leq \frac{1}{2}\log d_i + O(\log\log i).$$

Pad every description of m_i, given D_i, to a block of fixed length $\frac{1}{2}\log n + O(\log\log n)$. Since $\mathcal{D}$ is a distinguishing family for N, given $\mathcal{D}$, the values $m_1, \ldots, m_j$ determine M. Hence, by the established inequalities,

$$C(E(M)|\mathcal{D}) \leq C(m_1, \ldots, m_j|\mathcal{D}) \leq \sum_{i=1}^{j}\left(\frac{1}{2}\log n + O(\log\log n)\right).$$

Together with Equation 6.2 this implies the theorem. □

6.3.5 High-Probability Properties Revisited

Almost all strings have high complexity. Therefore, almost all tournaments and almost all undirected graphs have high complexity. Any combinatorial property proven about an arbitrary complex object in such a class will hold for almost all objects in the class. For example, the proof in Section 6.3.1 can trivially be strengthened as follows: By Theorem 2.2.1, page 117, there are at least $2^{n(n-1)/2}(1-1/n)$ tournaments T on n nodes with

$$C(E(T)|n,p) \geq n(n-1)/2 - \log n.$$

This is a $(1-1/n)$th fraction of all tournaments on n nodes. Using the displayed equation in the proof yields the corollary below:

Corollary 6.3.1 For almost all tournaments on n nodes (at least a $(1-1/n)$th fraction), the largest transitive subtournament has at most $1 + 2\lfloor 2\log n\rfloor$ nodes, from some n onward.

Similarly, choosing $C(E(T)|n,k,p) \geq n(n-1)/2 - \log n$ in the proof in Section 6.3.2 yields the following:

Corollary 6.3.2 For all large enough k, there is some n with $n \leq 2^k k^2(\ln 2 + o(1))$ such that almost all tournaments on n nodes (at least a $(1-1/n)$th fraction) have property $S(k)$.

The Kolmogorov complexity argument generally yields results on *expected* and *high-probability* properties rather than worst-case properties, and is especially suited to obtaining results on random structures. Other such applications (such as the expected maximum vertex degree of randomly generated trees and a related result on random mappings) can be found in the exercises and in Section 6.4.

Exercises

6.3.1. [17] Let $w(n)$ be the largest integer such that for every tournament T on $N = \{1, \ldots, n\}$ there exist disjoint sets A and B, each of cardinality $w(n)$, in N such that $A \times B \subseteq T$. Prove $w(n) \leq 2\lfloor \log n \rfloor$.

Comments. Hint: add $2w(n)\lfloor \log n \rfloor$ bits to describe nodes, and save $w(n)^2$ bits on edges. Source of the problem: P. Erdős and J.H. Spencer, *Probabilistic Methods in Combinatorics,* Academic Press, 1974.

6.3.2. [25] Let T be a tournament on $N = \{1, \ldots, n\}$. Define a ranking R as an ordering of N. For $(i, j) \in T$, if $R(i) < R(j)$, we say that R *agrees* with (i, j). Otherwise, it *disagrees* with that edge. We are interested in a ranking that is most consistent with T, that is, such that the number of edges that agree with R is maximized. Show that for large enough n, there exist tournaments such that any ranking disagrees with at least 49% of its edges.

Comments. A simple incompressibility argument is given by M. Fouz and P. Nicholson, CS798 Course Report, University of Waterloo, December 2007. Relevant literature on this problem can be found in [N. Alon, J.H. Spencer, *The Probabilistic Method*, Wiley, 2000, p. 134].

6.3.3. [17] Let $G = (V, E)$ with $V = \{1, \ldots, n\}$ be an undirected graph on n nodes with $C(G|n, p) \geq n(n-1)/2$, where p is a fixed program to be used to reconstruct G. A *clique* of a graph is a complete subgraph of that graph. Show that G does not contain a clique on more than $1 + \lfloor 2 \log n \rfloor$ nodes.

Comments. Hint: use Section 6.3.1. To compare this result with a similar one about randomly generated graphs, N. Alon, J.H. Spencer, P. Erdős, *The Probabilistic Method*, Wiley, 1992, pp. 86–87, show that a random graph with edge probability $\frac{1}{2}$ contains a clique on $2 \log n$ nodes with probability at least $1 - 1/e^{n^2}$.

6.3.4. [36] Let $K(N)$ denote the complete undirected graph of n nodes $N = \{1, \ldots, n\}$. If A and B are disjoint subsets of N, then $K(A, B)$ denotes the complete bipartite graph on sets A and B. A set $\mathbf{C} = (K(A_1, B_1), \ldots, K(A_j, B_j))$ is called a covering family of $K(N)$ if for every edge $\{u, v\} \in K(N)$ there exists an i $(1 \leq i \leq j)$ such that $\{u, v\} \in K(A_i, B_i)$. Let $g(n)$ denote the minimum of $\sum_{1 \leq i \leq j} d(A_i \bigcup B_i)$ over all covering families for $K(N)$. Prove by incompressibility that $g(n)/n \geq \log n + O(\log \log n)$.

Comments. An information-theoretic proof appears in [N. Pippenger, *J. Comb. Theory, Ser. A,* 23(1977), 105–115]. Hint: use the symmetry of information, Theorem 2.8.2, on page 190. Source: M. Li and P.M.B. Vitányi, *J. Comb. Theory, Ser. A*, 66:2(1994), 226–236.

6.3.5. [25] Consider a random directed graph whose n^2 nodes are on the intersections of a two-dimensional n by n grid. All vertical edges

(the grid edges) are present and directed upward. For every pair of horizontally neighboring nodes, we flip a three-sided coin; with probability $p < \frac{1}{2}$ we add an edge from left to right, with probability p we add an edge from right to left, and with probability $1 - 2p$ we add no edge. Use incompressibility to prove that the expected maximum path length over all such random graphs is bounded by $O(n)$.

Comments. Source: T. Jiang and Z.Q. Luo, personal communication, 1992. This problem was studied in connection with communication networks.

6.3.6. [36] From among $\binom{n}{3}$ triangles with vertices chosen from n points in the unit square, let T_n be the one with the smallest area, and let A_n be the area of T_n. Heilbronn's triangle problem asks for the maximum value Δ_n assumed by A_n over all choices of n points. We consider the average case: Show that if the n points are chosen independently and at random (with a uniform distribution), then there exist positive constants c and C such that $c/n^3 < \mu_n < C/n^3$ for all large enough values of n, where μ_n is the expectation of A_n. Moreover, $c/n^3 < A_n < C/n^3$, with probability close to one.

Comments. Hint: put the n points on the intersections of a $k \times k$ grid and show that the description of the arrangement can be compressed significantly below the maximum, both if the smallest triangle has too large an area and if it has too small an area, independent of k. Source: T. Jiang, M. Li, P.M.B. Vitányi, *Random Struct. Alg.*, 20:2(2002), 206–219, which contains literature pointers to other related results. A generalization of the average case result is given in [G. Grimmett, S. Janson, *Random Struct. Alg.*, 23:2(2003), 206–223]. History: H.A. Heilbronn conjectured that $\Delta_n = O(1/n^2)$ in 1950, and P. Erdős proved that $\Delta_n = \Omega(1/n^2)$ in 1950. K.F. Roth proved that $\Delta_n = O(1/n\sqrt{\log\log n})$ in 1951. W.M. Schmidt improved Roth's bound to $O(1/n\sqrt{\log n})$ in 1972. Roth further improved this to $O(1/n^{1.105})$ and $O(1/n^{1.117})$ in 1972. J. Komlós, J. Pintz, and E. Szemerédi further improved this to $O(1/n^{8/7-\epsilon})$ in 1981 and they proved an $\Omega(\log n/n^2)$ lower bound in 1982. The problem has many generalizations and several dedicated websites.

6.3.7. [35] Given an n-dimensional cube and a permutation π of its nodes, each node v wants to send an information packet to node $\pi(v)$ as fast as possible. Label every edge in the cube with its dimension from $\{1, \ldots, n\}$. A route $(v_1 \to v_2 \to \cdots \to v_k)$ is ascending if (v_i, v_{i+1}) has higher dimension than (v_{i-1}, v_i) for all $2 < i < k - 1$. If two packets use the same edge in the same direction at the same time, then a *collision* occurs, and one packet has to wait. How do we avoid too many collisions on each route? Consider the following probabilistic algorithm A_π: **Step 1.** For every node v, choose randomly a node w. Node v sends its packet over the uniquely determined ascending route to w. **Step 2.** Send the

packet from w to $\pi(v)$ through the unique ascending route. Prove that for every constant c, algorithm A_π finishes with probability greater than $1 - 2^{-(c-5n-O(1))/2}$ after at most $2n + 2c$ steps.

Comments. Hint: show that the description of a route on which too many collisions occur can be compressed. Source: L.G. Valiant and G. Brebner, *Proc. 13th ACM Symp. Theory Comput.*, 1981, pp. 263–277; S. Reisch and G. Schnitger give an incompressibility proof in [*Proc. 23rd IEEE Found. Comput. Sci.*, 1982, pp. 45–52].

6.3.8. [39] Let $L \subset \{0,1\}^{2n}$ be a language to be recognized by two parties P and Q with unlimited computation power. Party P knows the first n bits of the input and party Q knows the last n bits. P and Q exchange messages to recognize L according to some bounded-error two-way probabilistic protocol. An input is *accepted* if the probability of acceptance is at least $1 - \epsilon$ for some fixed ϵ, $0 \leq \epsilon < \frac{1}{2}$; an input is *rejected* if the probability of rejection is at least $1 - \epsilon$; and every input must be either rejected or accepted. The *probabilistic communication complexity* of an input $(x_1, \ldots, x_{2n})$ is the worst case, over all sequences of fair coin tosses, of the number of bits exchanged. The probabilistic communication complexity of the language is the maximum of this over all inputs. The set intersection language SETIN is defined to be the set of all sequences $a_1 \ldots a_n b_1 \ldots b_n$ over $\{0,1\}$ with $\sum_{i=1}^{n} a_i b_i \geq 1$. ($P$ knows $a_1, \ldots, a_n$ and Q knows $b_1, \ldots, b_n$.) Prove that the probabilistic communication complexity of SETIN is $\Omega(n)$.

Comments. Source: B. Kalyanasundaram and G. Schnitger, *SIAM J. Discrete Math.*, 5:4(1992), 545–557.

6.3.9. [37] An (n, d, m)-graph is a bipartite multigraph with n vertices on the left side and m vertices on the right side, with every vertex on the left having degree d, and every vertex on the right having degree dn/m (assuming $m|dn$). An (n, d, m)-graph is (α, β)-expanding if every subset S of αn vertices on the left has more than βm neighbors on the right, for $0 < \alpha \leq \beta < 1$. Prove that for every n, $0 < \alpha \leq \beta < 1$, $\lambda > 0$, there is a

$$d > \frac{h(\alpha) + h(\beta)\lambda}{h(\alpha) - h(\alpha/\beta)\beta}$$

such that there is an (α, β)-expanding $(n, d, \lambda n)$-graph.

Comments. Hint: take a $(n, d, \lambda n)$-graph of maximal complexity. Source: U. Schöning, *Random Struct. Alg.*, 17(2000), 64–77. The original probabilistic proof with $\lambda = 1$ is in [L.A. Bassalygo, *Prob. Inform. Transmission*, 17(1981), 206–211].

6.3.10. [25] An (n, d, α, c) OR-concentrator is a bipartite graph $G(L + R, E)$ on the independent vertex sets L and R with $d(L) = d(R) = n$

such that (i) every vertex in L has degree d, and (ii) every subset $S \subseteq L$ with $d(S) \leq \alpha n$ is connected to at least cn neighbors (in R). Show that there exist $(n, 9.48, \frac{1}{3}, 2)$ OR-concentrators.

Comments. A simple incompressibility proof is given by M. Fouz, CS798 Course Report, University of Waterloo, December 2007. A probabilistic proof (with worse constants) is found in [R. Motwani, P. Raghavan, *Randomized Algorithms*, Cambridge Univ. Press, 1995, pp. 108–110].

6.3.11. [40] Let $s_1 \ldots s_n$ be a string over an alphabet of cardinality c. A *monochromatic arithmetic progression* (m.a.p.) of length k is a subsequence $s_i s_{i+t} s_{i+2t} \ldots s_{i+(k-1)t}$ with all characters equal. The van der Waerden number $w(k; c)$ is the least number n such that every string of length n contains a m.a.p. of length k. Use the incompressibility method in the problems below.

(a) Show that $w(k; c) > \sqrt{k-1} \cdot c^{\frac{k}{2}-1}$.

(b) Strengthen the bound to $w(k; c) > \frac{c^{k-2}}{4k} \cdot \frac{k-1}{k}$.

Comments. The lower bound of Item (b) matches the one obtained originally by L. Lovász, and is worse than later applications of Lovász's local lemma by a factor $4/e$; see for example [Z. Szábo, *Random Struct. Alg.*, 1:3(1990), 343–360]. The method can also be used for other Ramsey-type lower bounds. Hint for Item (b):
(i) Show that $\lceil \log_c(n \cdot k \cdot \frac{k}{k-1}) \rceil + 1$ characters suffice to encode a m.a.p., if it is known to intersect some other fixed progression of length k.
(ii) Consider a procedure which in a long incompressible string repeatedly does the following: Find a m.a.p. within the first $w(k; c)$ characters. Encode it by some string, delete the corresponding characters, and replace them with characters from the end of the string.
(iii) Use the following fact without proof: With $\log_c 4$ additional characters in the encoding of every deleted progression one can guarantee that there is always a m.a.p. in the first $w(k; c)$ characters that intersects the positions of a previously deleted progression (determinable without additional characters).
Implement Items (i) through (iii) as follows: maintain a stack whose elements each contain the positions of a progression that has been deleted. Upon deletion of a progression, push its positions onto the stack. If a progression in the current string intersects the positions on top of the stack, encode it this way; otherwise, delete the top of the stack. Encoding which case happened can be done with $\log_c 2$ characters and every case happens at most once per deleted progression. Source: P. Schweitzer, Using the incompressibility method to obtain local lemma results for Ramsey-type problems, *Inform. Process. Lett.*, to appear.

6.4 Kolmogorov Random Graphs

Statistical properties of strings with high Kolmogorov complexity were studied in Section 2.6. The interpretation of strings as more complex combinatorial objects leads to a completely new set of properties and problems that have no direct counterpart in the flatter string world. Here we derive topological, combinatorial, and statistical properties of graphs with high Kolmogorov complexity. Every such graph possesses simultaneously all properties that hold with high probability for randomly generated graphs. They constitute almost all graphs, and the derived properties a fortiori hold with probability that goes to 1 as the number of nodes grows unboundedly, in the sense of Section 6.2.

Definition 6.4.1 Every labeled graph $G = (V, E)$ on n nodes $V = \{1, 2, \ldots, n\}$ can be coded (up to automorphism) by a binary string $E(G)$ of length $n(n-1)/2$. We enumerate the $n(n-1)/2$ possible edges (i, j) in a graph on n nodes in standard lexicographic order without repetitions and set the ith bit in the string to 1 if the edge is present and to 0 otherwise. Conversely, every binary string of length $n(n-1)/2$ encodes a graph on n nodes. Hence we can identify every such graph with its corresponding binary string.

Definition 6.4.2 A labeled graph G on n nodes has *randomness deficiency* at most $\delta(n)$, and is called $\delta(n)$-*random*, if it satisfies

$$C(E(G)|n, \delta) \geq n(n-1)/2 - \delta(n). \tag{6.3}$$

Lemma 6.4.1 *A fraction of at least* $1 - 1/2^{\delta(n)}$ *of all labeled graphs* G *on* n *nodes is* $\delta(n)$*-random.*

This is a corollary of Lemma 6.2.1. For example, the $c \log n$-random labeled graphs constitute a fraction of at least $(1 - 1/n^c)$ of all graphs on n nodes, where $c > 0$ is an arbitrary constant.

High-complexity labeled graphs have many specific topological properties, which seems to contradict their randomness. However, randomness is not lawlessness but rather enforces strict statistical regularities, for example, to have diameter exactly two.

Lemma 6.4.2 *The degree* d *of every node of a* $\delta(n)$*-random labeled graph satisfies*

$$|d - (n-1)/2| = O\left(\sqrt{(\delta(n) + \log n)n}\right).$$

Proof. Assume that there is a node such that the deviation of its degree d from $(n-1)/2$ is greater than k. From the lower bound on $C(E(G)|n, \delta)$ corresponding to the assumption that G is random, we can estimate an upper bound on k as follows:

In a description of $G = (V, E)$ given n, δ we can indicate which edges are incident on node i by giving the index of the interconnection pattern (the characteristic sequence of the set $V_i = \{j \in V - \{i\} : (i, j) \in E\}$ in $n - 1$ bits where the jth bit is 1 if $j \in V_i$ and 0 otherwise) in the ensemble of

$$m = \sum_{|d-(n-1)/2|>k} \binom{n-1}{d} \leq 2^n e^{-2k^2/3(n-1)} \tag{6.4}$$

possibilities. The last inequality follows from a general estimate of the tail probability of the binomial distribution, with s_n the number of successful outcomes in n experiments with probability of success $p = \frac{1}{2}$. Namely, by Chernoff's bounds, Equation 2.4 on page 167,

$$\Pr(|s_n - pn| > k) \leq 2e^{-k^2/3pn}. \tag{6.5}$$

To describe G it then suffices to modify the old code of G by prefixing it with

- A description of this discussion in $O(1)$ bits;
- the identity of node i in $\lfloor \log(n+1) \rfloor$ bits;
- the value of k in $\lfloor \log(n+1) \rfloor$ bits, possibly adding nonsignificant 0's to pad up to this amount;
- the index of the interconnection pattern in $\log m$ bits (we know n, k and hence $\log m$); followed by
- the old code for G with the bits in the code denoting the presence or absence of the possible edges that are incident on node i deleted.

Clearly, given n we can reconstruct the graph G from the new description. The total description we have achieved is an effective program of

$$\log m + 2\log n + n(n-1)/2 - n + O(1)$$

bits. This must be at least the length of the shortest effective binary program, which is $C(E(G)|n, \delta)$, satisfying Equation 6.3. Therefore,

$$\log m \geq n - 2\log n - O(1) - \delta(n).$$

Since we have estimated in Equation 6.4 that

$$\log m \leq n - (2k^2/3(n-1)) \log e,$$

it follows that $k \leq \sqrt{\frac{3}{2}(\delta(n) + 2\log n + O(1))(n-1)/\log e}$. □

Lemma 6.4.3 *All $o(n)$-random labeled graphs have $\frac{1}{4}n + o(n)$ disjoint paths of length 2 between every pair of nodes i, j. In particular, all $o(n)$-random labeled graphs have diameter 2.*

Proof. The only graphs with diameter 1 are the complete graphs that can be described in $O(1)$ bits, given n, and hence are not random. It remains to consider an $o(n)$-random graph $G = (V, E)$ with diameter greater than or equal to 2. Let i, j be a pair of nodes connected by r disjoint paths of length 2. Then we can describe G by modifying the old code for G as follows:

- a program to reconstruct the object from the various parts of the encoding in $O(1)$ bits;
- the identities of $i < j$ in $2 \log n$ bits;
- the old code $E(G)$ of G with the $2(n-2)$ bits representing presence or absence of edges (j, k) and (i, k) for every $k \neq i, j$ deleted;
- a shortest program for the string $e_{i,j}$ consisting of the (reordered) $n - 2$ pairs of bits deleted above.

From this description we can reconstruct G in

$$O(\log n) + \binom{n}{2} - 2(n-2) + C(e_{i,j}|n)$$

bits, from which we may conclude that $C(e_{i,j}|n) \geq l(e_{i,j}) - o(n)$. As shown in Lemma 2.6.1 this implies that the frequency of occurrence in $e_{i,j}$ of the aligned 2-bit block 11—which by construction equals the number of disjoint paths of length 2 between i and j—is $\frac{1}{4}n + o(n)$.

□

A graph is *k-connected* if there are at least k node-disjoint paths between every pair of nodes.

Corollary 6.4.1 *All $o(n)$-random labeled graphs are $(\frac{1}{4}n + o(n))$-connected.*

Lemma 6.4.4 *Let $G = (V, E)$ be a graph on n nodes with randomness deficiency $O(\log n)$. Then the largest clique in G has at most $\lfloor 2 \log n \rfloor + O(1)$ nodes.*

Proof. This is the same proof as that for the largest transitive subtournament in a high-complexity tournament, Theorem 6.3.1. □

With respect to the related property of random graphs, in [N. Alon, J.H. Spencer, and P. Erdős, *The Probabilistic Method*, 1992, pp. 86, 87] it is shown that a random graph with edge probability $\frac{1}{2}$ contains a clique on asymptotically $2 \log n$ nodes with probability at least $1 - e^{-n^2}$.

Lemma 6.4.5 *Let c be a fixed constant. If G is a $c\log n$-random labeled graph, then from every node i all other nodes are either directly connected to i or are directly connected to one of the least $(c+3)\log n$ nodes directly adjacent to i.*

Proof. Given i, let A be the set of the least $(c+3)\log n$ nodes directly adjacent to i. Assume by way of contradiction that there is a node k of G that is not directly connected to a node in $A\bigcup\{i\}$. We can describe G as follows:

- a description of this discussion in $O(1)$ bits;
- a literal description of i in $\log n$ bits;
- a literal description of the presence or absence of edges between i and the other nodes in $n-1$ bits;
- a literal description of k and its incident edges in $\log n + n - 2 - (c+3)\log n$ bits;
- the encoding $E(G)$ with the edges incident with nodes i and k deleted, saving at least $2n-2$ bits.

Altogether the resultant description has

$$n(n-1)/2 + 2\log n + 2n - 3 - (c+3)\log n - 2n + 2$$

bits, which contradicts the $c\log n$-randomness of G by Equation 6.3 on page 461. The lemma is proven. □

In the description we have explicitly added the adjacency pattern of node i, which we deleted later again. This zero-sum swap is necessary to be able to unambiguously identify the adjacency pattern of i in order to reconstruct G. Since we know the identities of i and the nodes adjacent to i (they are the prefix where no bits have been deleted), we can reconstruct G from this discussion and the new description, given n.

6.4.1 Statistics of Subgraphs

We start by defining the notion of labeled subgraph of a labeled graph. Let $G = (V, E)$ be a labeled graph on n nodes. Consider a labeled graph H on k nodes $\{1, 2, \ldots, k\}$. Each subset of k nodes of G induces a subgraph G_k of G. The subgraph G_k is an ordered labeled *occurrence* of H when we obtain H by relabeling the nodes $i_1 < i_2 < \cdots < i_k$ of G_k as $1, 2, \ldots, k$.

It is easy to conclude from the statistics of high-complexity strings in Lemma 2.6.1 that the frequency of every of the two labeled two-node subgraphs in a $\delta(n)$-random graph G is

$$\frac{n(n-1)}{4} \pm \sqrt{\frac{3}{4}(\delta(n) + O(1))n(n-1)/\log e}.$$

This case is easy, since the frequency of such subgraphs corresponds to the frequency of 1's or 0's in the $\binom{n}{2}$-length standard encoding $E(G)$ of G. However, to determine the frequencies of labeled subgraphs on k nodes (up to isomorphism) for $k > 2$ is a matter more complicated than the frequencies of substrings of length k. Clearly, there are $\binom{n}{k}$ subsets of k nodes out of n and hence that many occurrences of subgraphs. Such subgraphs may overlap in more complex ways than substrings of a string. Let $\#H(G)$ be *the number of times H occurs* as an ordered labeled subgraph of G (possibly overlapping). Let p be the probability that we obtain H by flipping a fair coin to decide for every pair of nodes whether it is connected by an edge. Then, $p = 2^{-k(k-1)/2}$. The proof of the following theorem is deferred to Exercise 6.4.2 on page 468.

Theorem 6.4.1 *Assume the terminology above with $G = (V, E)$ a labeled graph on n nodes, k a positive integer dividing n, and H a labeled graph on $k \leq \sqrt{2 \log n}$ nodes. Let $C(E(G)|n) \geq \binom{n}{2} - \delta(n)$. Then*

$$\left| \#H(G) - \binom{n}{k} p \right| \leq \binom{n}{k} \sqrt{\alpha(k/n)p},$$

with $\alpha = (K(H|n) + \delta(n) + \log \binom{n}{k}/(n/k) + O(1))3/\log e$.

6.4.2 Unlabeled Graph Counting

An unlabeled graph is a graph with no labels. For convenience we can define this as follows: Call two labeled graphs *equivalent* (up to relabeling) if there is a relabeling that makes them equal. An *unlabeled graph* is an equivalence class of labeled graphs. An *automorphism* of $G = (V, E)$ is a permutation π of V such that $(\pi(u), \pi(v)) \in E$ iff $(u, v) \in E$. Clearly, the set of automorphisms of a labeled graph forms a group with group operation of function composition and the identity permutation as unity. It is easy to verify that π is an automorphism of G iff $\pi(G)$ and G have the *same binary string standard encoding*, that is, $E(G) = E(\pi(G))$. This contrasts with the more general case of permutation relabeling, where the standard encodings may be different. A labeled graph is *rigid* if its only automorphism is the identity automorphism. It turns out that Kolmogorov random labeled graphs are rigid graphs. To obtain an expression for the number of unlabeled graphs we have to estimate the number of automorphisms of a graph in terms of its randomness deficiency. Below, 'graph' means 'labeled graph' unless indicated otherwise.

In [F. Harary and E.M. Palmer, *Graphical Enumeration*, Academic Press, 1973] an asymptotic expression for the number of unlabeled graphs is derived using sophisticated methods. We give a new elementary proof by incompressibility. Denote by g_n the number of unlabeled graphs on n nodes—that is, the number of isomorphism classes in the set $\mathcal{G}_n$ of undirected labeled graphs on nodes $\{0, 1, \ldots, n-1\}$.

Theorem 6.4.2 $g_n \sim \frac{2^{\binom{n}{2}}}{n!}$.

Proof. Clearly,

$$g_n = \sum_{G \in \mathcal{G}_n} \frac{1}{d([G])},$$

where $[G]$ is the isomorphism class of graph G. By elementary group theory,

$$d([G]) = \frac{d(S_n)}{d(\mathrm{Aut}(G))} = \frac{n!}{d(\mathrm{Aut}(G))},$$

where S_n is the group of permutations on n elements, and $\mathrm{Aut}(G)$ is the automorphism group of G. Let us partition $\mathcal{G}_n$ into $\mathcal{G}_n = \mathcal{G}_n^0 \bigcup \cdots \bigcup \mathcal{G}_n^n$, where $\mathcal{G}_n^m$ is the set of graphs for which m is the number of nodes moved (mapped to another node) by any of its automorphisms.

Claim 6.4.1 For $G \in \mathcal{G}_n^m$, $d(\mathrm{Aut}(G)) \leq n^m = 2^{m \log n}$.

Proof. $d(\mathrm{Aut}(G)) \leq \binom{n}{m} m! \leq n^m$. □

Consider every graph $G \in \mathcal{G}_n$ having a probability $\Pr(G) = 2^{-\binom{n}{2}}$.

Claim 6.4.2 $\Pr(G \in \mathcal{G}_n^m) \leq 2^{-m(\frac{1}{2}n - \frac{3}{8}m - \log n)}$.

Proof. By Lemma 6.4.1 it suffices to show that if $G \in \mathcal{G}_n^m$ and

$$C(E(G)|n,m) \geq \binom{n}{2} - \delta(n,m)$$

then $\delta(n,m)$ satisfies

$$\delta(n,m) \geq m\left(\frac{1}{2}n - \frac{3}{8}m - \log n\right). \tag{6.6}$$

Let $\pi \in \mathrm{Aut}(G)$ move m nodes. Suppose π is the product of k disjoint cycles of sizes $c_1, \ldots, c_k$. Spend at most $m \log n$ bits describing π: For example, if the nodes $i_1 < \cdots < i_m$ are moved, then list the sequence $\pi(i_1), \ldots, \pi(i_m)$. Writing the nodes of the latter sequence in increasing order, we obtain $i_1, \ldots, i_m$ again, that is, we execute permutation π^{-1}, and hence we obtain π.

Select one node from each cycle—say, the lowest-numbered one. Then for every unselected node on a cycle, we can delete the $n - m$ bits corresponding to the presence or absence of edges to stable nodes, and $m - k$

half-bits corresponding to presence or absence of edges to the other, unselected, cycle nodes. In total we delete

$$\sum_{i=1}^{k}(c_i-1)\left(n-m+\frac{m-k}{2}\right)=(m-k)\left(n-\frac{m+k}{2}\right)$$

bits. Observing that $k=\frac{1}{2}m$ is the largest possible value for k, we arrive at the claimed $\delta(n,m)$ of G (difference between savings and spending is $\frac{1}{2}m(n-\frac{3}{4}m)-m\log n$) of Equation 6.6. □

We continue the proof of the main theorem:

$$g_n=\sum_{G\in\mathcal{G}_n}\frac{1}{d([G])}=\sum_{G\in\mathcal{G}_n}\frac{d(\mathrm{Aut}(g))}{n!}=\frac{2^{\binom{n}{2}}}{n!}E_n,$$

where $E_n:=\sum_{G\in\mathcal{G}_n}\Pr(G)d(\mathrm{Aut}(G))$ is the expected size of the automorphism group of a graph on n nodes. Clearly, $E_n\geq 1$, yielding the lower bound on g_n. For the upper bound on g_n, noting that $\mathcal{G}_n^1=\varnothing$ and using the above claims, we find that

$$\begin{aligned}E_n&=\sum_{m=0}^{n}\Pr(G\in\mathcal{G}_n^m)\mathrm{Avg}_{G\in\mathcal{G}_n^m}d(\mathrm{Aut}(G))\\&\leq 1+\sum_{m=2}^{n}2^{-m(\frac{1}{2}n-\frac{3}{8}m-2\log n)}\\&\leq 1+2^{-(n-4\log n-2)},\end{aligned}$$

with Avg meaning 'the average,' which proves the theorem. □

The proof of the theorem shows that the error in the asymptotic expression is very small:

Corollary 6.4.2 $\frac{2^{\binom{n}{2}}}{n!}\leq g_n\leq\frac{2^{\binom{n}{2}}}{n!}\left(1+\frac{4n^4}{2^n}\right)$.

Equation 6.6 yields the following (note that $m=1$ is impossible):

Corollary 6.4.3 If a graph G has randomness deficiency slightly less than n (more precisely, $C(E(G)|n)\geq\binom{n}{2}-n-\log n-2$) then G is rigid.

The expression for g_n can be used to determine the maximal complexity of an unlabeled graph on n nodes. Namely, we can effectively enumerate all unlabeled graphs as follows:

Step 1. Effectively enumerate all labeled graphs on n nodes by enumerating all binary strings of length n and, and for every enumerated labeled graph G **do Step 2**

Step 2. If G cannot be obtained by relabeling from any previously enumerated labeled graph **then** G is added to the set of unlabeled graphs.

In this way, we obtain every unlabeled graph by precisely one labeled graph representing it. Since we can describe every unlabeled graph by its index in this enumeration, we find by Theorem 6.4.2 and Stirling's formula that if G is an unlabeled graph, then

$$C(E(G)|n) \leq \binom{n}{2} - n \log n + O(n).$$

Lemma 6.4.6 *Let G be a labeled graph on n nodes and let G_0 be the unlabeled version of G. There exists a graph G' and a label permutation π such that $G' = \pi(G)$ and up to additional constant terms $C(E(G')) = C(E(G_0))$ and $C(E(G)|n) = C(E(G_0), \pi|n)$.*

By Lemma 6.4.6, for *every* graph G on n nodes with maximum complexity there is a relabeling (permutation) that causes the complexity to drop by as much as $n \log n$. Our proofs of topological properties by the incompressibility method required the graph G to be Kolmogorov random in the sense of $C(E(G)|n) \geq \binom{n}{2} - O(\log n)$ or for some results $C(E(G)|n) \geq \binom{n}{2} - o(n)$. Hence by relabeling such a graph we can always obtain a labeled graph that has a complexity too low to use our incompressibility proof. Nonetheless, topological properties do not change under relabeling.

Exercises

6.4.1. [M40] Use the terminology of Theorem 6.4.1. A *cover* of G is a set $C = \{S_1, \ldots, S_N\}$ with $N = n/k$, where the S_i's are pairwise disjoint subsets of V and $\bigcup_{i=1}^{N} S_i = V$. There is a partition of the $\binom{n}{k}$ different k-node subsets into $h = \binom{n}{k}/N = \binom{n-1}{k-1}$ distinct covers of G, every cover consisting of $N = n/k$ disjoint subsets. That is, every subset of k nodes of V belongs to precisely one cover.

Comments. Source: Zs. Baranyai, pp. 91–108 in: A. Hajnal, R. Rado, and V.T. Sós, eds., *Infinite and Finite Sets, Proc. Coll. Keszthely, Colloq. Math. Soc. János Bolyai*, 10, Vol. 1, North-Holland, Amsterdam, 1975.

6.4.2. [27] Use Exercise 6.4.1 to prove Theorem 6.4.1.

Comments. Hint: similar to the proof of Theorem 2.6.1, with the labeled graph G in the part of the overall string, and cover elements (subsets of labeled nodes inducing subgraphs) taking the part of the blocks. Source: H.M. Buhrman, M. Li, J.T. Tromp, and P.M.B. Vitányi, *SIAM J. Comput*, 29:2(1999), 590–599. This is also the source for the next exercise.

6.4.3. [20] In Section 2.6 we investigated up to which length l all blocks of length l occurred at least once in every $\delta(n)$-random string of length n. Let $\delta(n) = 2^{\sqrt{2\log n}/2}/4\log n$ and G be a $\delta(n)$-random graph on n nodes. Show that for sufficiently large n, the graph G contains all subgraphs on $\sqrt{2\log n}$ nodes.

6.4.4. [26] Show that almost every labeled tree on n nodes has maximum degree of $O(\log n/\log\log n)$.

Comments. Hint: represent a labeled tree by a binary sequence of length $(n-2)\log n$ (the Prüfer code). Prove a one-to-one correspondence between labeled trees and binary sequences of such length. Use incompressibility to show that if a tree has larger degree, then one can compress the corresponding binary sequence. Since most binary sequences cannot be compressed, most trees do not have larger degree. Source: W.W. Kirchherr, *Inform. Process. Lett.*, 41(1992), 125–130.

6.5 Compact Routing

In very large networks such as the global telephone network or the Internet, the mass of messages being routed creates major bottlenecks, degrading performance. We analyze a tiny part of this issue by determining the optimal space to represent routing schemes in communication networks on average for all static networks.

A universal routing strategy for static communication networks will, for every network, generate a *routing scheme* for that particular network. Such a routing scheme comprises a *local routing function* for every node in this network. The routing function of node u returns for every destination $v \neq u$ an edge incident to u on a path from u to v. In this way, a routing scheme describes a path, called a *route*, between every pair of nodes u, v in the network.

It is easy to see that we can do shortest-path routing by entering a *routing table* in every node u that for every destination node v indicates to what adjacent node w a message to v should be routed first. If u has degree d, it requires a table of at most $n\log d$ bits, and the overall number of bits in all local routing tables never exceeds $n^2\log n$. Several factors may influence the cost of representing a routing scheme for a particular network. We use a basic model and leave variations to the exercises. Here we consider point-to-point communication networks on n nodes described by an undirected labeled graph $G = (V, E)$, where $V = \{1, \ldots, n\}$. Assume that nodes know the identities of their neighbors.

In [H.M. Buhrman, J.H. Hoepman, and P.M.B. Vitányi, *SIAM J. Comput.*, 28:4(1999), 1414–1432], it is shown that in most models, for almost all graphs (that is, networks), $\Theta(n^2)$ bits are necessary and sufficient for shortest-path routing. By 'almost all graphs' we mean the Kolmogorov random graphs that constitute a fraction of $1 - 1/n^c$ of all graphs on n nodes, where $c \geq 3$ is an

arbitrary fixed constant. In contrast, there is a model that causes the average-case lower bound to rise to $\Omega(n^2 \log n)$ and another model whose average-case upper bound drops to $O(n \log^2 n)$. This clearly exposes the sensitivity of such bounds to the model under consideration.

6.5.1 Upper Bound

In general (on almost all networks), one can use shortest-path routing schemes occupying at most $O(n^2)$ bits. Relaxing the requirement of shortest path is expressed in the *stretch factor* of a routing scheme. This equals the maximum ratio between the length of a route it produces and the shortest path between the endpoints of that route. The stretch factor of a routing strategy equals the maximal stretch factor attained by any of the routing schemes it generates. The shortest-path routing strategy has stretch factor equal to 1. Allowing stretch factors larger than 1 reduces the space requirements—to as low as $O(n)$ bits for stretch factors of $O(\log n)$, Exercise 6.5.2.

Theorem 6.5.1 *For shortest-path routing in $O(\log n)$-random graphs, local routing functions can be stored in $6n$ bits per node. Hence the complete routing scheme is represented by $6n^2$ bits.*

Proof. Let G be an $O(\log n)$-random graph on n nodes. By Lemma 6.4.5 we know that from every node u we can route via shortest paths to every node v through the $O(\log n)$ directly adjacent nodes of u that have the least indexes. By Lemma 6.4.3, G has diameter 2. Once the message has reached node v its destination is either node v or a direct neighbor of node v (which is known in node v by assumption). Therefore, routing functions of size $O(n \log \log n)$ can be used to do shortest-path routing. We can do better than this.

Let $A_0 \subseteq V$ be the set of nodes in G that are not directly connected to u. Let $v_1, \ldots, v_m$ be the $O(\log n)$ least indexed nodes directly adjacent to node u (Lemma 6.4.5), through which we can route via shortest paths to all nodes in A_0. For $t = 1, 2 \ldots, l$ define $A_t = \{w \in A_0 - \bigcup_{s=1}^{t-1} A_s : (v_t, w) \in E\}$. Let $m_0 = d(A_0)$ and define $m_{t+1} = m_t - d(A_{t+1})$. Let l be the first t such that $m_t < n/\log \log n$. Then we claim that v_t is connected by an edge in E to at least $\frac{1}{3}$ of the nodes not connected by edges in E to nodes $u, v_1, \ldots, v_{t-1}$.

Claim 6.5.1 $d(A_t) > m_{t-1}/3$ for $1 \leq t \leq l$.

Proof. Suppose by way of contradiction that there exists a least $t \leq l$ such that $|d(A_t) - m_{t-1}/2| > m_{t-1}/6$. Then we can describe G, given n, as follows:

- This discussion in $O(1)$ bits.

- Nodes u, v_t in $2 \log n$ bits, padded with 0's if need be.
- The presence or absence of edges incident with nodes $u, v_1, \ldots, v_{t-1}$ in $r = n - 1 + \cdots + n - (t-1)$ bits. This gives us the characteristic sequences of $A_0, \ldots, A_{t-1}$ in V, where a *characteristic sequence* of A in V is a string of $d(V)$ bits such that for every $v \in V$, the vth bit equals 1 if $v \in A$ and the vth bit is 0 otherwise.
- A self-delimiting description of the characteristic sequence of A_t in $A_0 - \bigcup_{s=1}^{t-1} A_s$, using Chernoff's bound, Equation 2.4 on page 167, in at most $m_{t-1} - \frac{2}{3}\left(\frac{1}{6}\right)^2 m_{t-1} \log e + O(\log m_{t-1})$ bits.
- The description $E(G)$ with all bits corresponding to the presence or absence of edges between v_t and the nodes in $A_0 - \bigcup_{s=1}^{t-1} A_s$ deleted, saving m_{t-1} bits. Furthermore, we also delete all bits corresponding to presence or absence of edges incident with $u, v_1, \ldots, v_{t-1}$, saving a further r bits.

This description of G uses at most

$$\frac{1}{2}n(n-1) + O(\log n) + m_{t-1} - \frac{2}{3}\left(\frac{1}{6}\right)^2 m_{t-1} \log e - m_{t-1}$$

bits, which contradicts the $O(\log n)$-randomness of G by Equation 6.3 on page 461, because $m_{t-1} > n/\log\log n$. □

Recall that l is the least integer such that $m_l < n/\log\log n$. We construct the local routing function $F(u)$ as follows:

- A table of intermediate routing node entries for all the nodes in A_0 in increasing order. For every node w in $\bigcup_{s=1}^{l} A_s$ we enter in the wth position in the table the unary representation of the least intermediate node v, with $(u, v), (v, w) \in E$, followed by a 0. For the nodes that are not in $\bigcup_{s=1}^{l} A_s$ we enter a 0 in their position in the table indicating that an entry for this node can be found in the second table. By Claim 6.5.1, the size of this table is bounded by

$$n + \sum_{s=1}^{l} \frac{1}{3}\left(\frac{2}{3}\right)^{s-1} sn \leq n + \sum_{s=1}^{\infty} \frac{1}{3}\left(\frac{2}{3}\right)^{s-1} sn \leq 4n.$$

- A table with explicitly binary-coded intermediate nodes on a shortest path for the ordered set of the remaining destination nodes. Those nodes have a 0 entry in the first table and there are at most $m_l < n/\log\log n$ of them, namely the nodes in $A_0 - \bigcup_{s=1}^{l} A_s$. Each entry consists of the code of length $\log\log n + O(1)$ for the position in increasing order of a node out of $v_1, \ldots, v_m$ with $m = O(\log n)$ by Lemma 6.4.5. Hence this second table requires at most $2n$ bits.

The routing algorithm is as follows: The direct neighbors of u are known in node u and are routed without the routing table. If we route from start node u to target node w that is not directly adjacent to u, then we do the following. If node w has an entry in the first table then route over the edge coded in unary; otherwise find an entry for node w in the second table.

Altogether, we have $d(F(u)) \leq 6n$. Slightly more precise counting and choosing l such that m_l is the first such quantity $< n/\log n$ shows that $d(F(u)) \leq 3n$. □

6.5.2 Lower Bound

We show that $\Omega(n^2)$ bits are required to perform routing on Kolmogorov random graphs. Hence the upper bound in Theorem 6.5.1 is tight up to order of magnitude.

Theorem 6.5.2 *For shortest-path routing in $o(n)$-random graphs, every local routing function must be stored in at least $\frac{1}{2}n - o(n)$ bits per node. Hence the complete routing scheme requires at least $n^2/2 - o(n^2)$ bits to be stored.*

Proof. Let G be an $o(n)$-random graph. Let $F(u)$ be the local routing function of node u of G, and let $d(F(u))$ be the number of bits used to store $F(u)$. Let $E(G)$ be the standard encoding of G in $n(n-1)/2$ bits as in Definition 6.4.1. We now give another way to describe G using some local routing function $F(u)$:

- A description of this discussion in $O(1)$ bits.
- A description of u in exactly $\log n$ bits, padded with 0's if needed.
- A description of the presence or absence of edges between u and the other nodes in V in $n-1$ bits.
- A self-delimiting description of $F(u)$ in $d(F(u)) + 2\log d(F(u))$ bits.
- The code $E(G)$ with all bits deleted corresponding to edges $(v,w) \in E$ for every v and w such that $F(u)$ routes messages to w through the least intermediary node v. This saves at least $\frac{1}{2}n - o(n)$ bits, since there are at least $\frac{1}{2}n - o(n)$ nodes w such that $(u,w) \notin E$ by Lemma 6.4.2, and since the diameter of G is 2 by Lemma 6.4.3, there is a shortest path $(u,v),(v,w) \in E^2$ for some v. Furthermore, we delete all bits corresponding to the presence or absence of edges between u and the other nodes in V, saving another $n-1$ bits. This corresponds to the $n-1$ bits for edges connected to u that we added in one connected block above.

In the description we have explicitly added the adjacency pattern of node u that we deleted elswhere. This zero-sum swap is necessary in order to unambiguously identify the adjacency pattern of u to reconstruct G given n, as follows: Reconstruct the bits corresponding to the deleted edges using u and $F(u)$ and subsequently insert them in the appropriate positions of the remnants of $E(G)$. We can do so because these positions can be simply reconstructed in increasing order. In total, this new description has

$$\frac{1}{2}n(n-1) + O(1) + O(\log n) + d(F(u)) - \frac{1}{2}n + o(n)$$

bits, which must be at least $n(n-1)/2 - o(n)$ by Equation 6.3. Hence, $d(F(u)) \geq \frac{1}{2}n - o(n)$, which proves the theorem. □

6.5.3 Average Case

Consider the average cost, taken over all labeled graphs of n nodes, of representing a routing scheme for graphs over n nodes. For a graph G, let $T(G)$ be the number of bits used to store its routing scheme. The *average* total number of bits to store the routing scheme for routing over labeled graphs on n nodes is $\sum T(G)/2^{n(n-1)/2}$, with the sum taken over all graphs G on nodes $\{1, 2, \ldots, n\}$, that is, the uniform average over all the labeled graphs on n nodes.

The results on Kolmogorov random graphs above have the following corollaries. Consider the subset of $(3 \log n)$-random graphs within the class of $O(\log n)$-random graphs on n nodes. They constitute a fraction of at least $(1 - 1/n^3)$ of the class of all graphs on n nodes. The trivial upper bound on the minimal total number of bits for all routing functions together is $O(n^2 \log n)$ for shortest-path routing on all graphs on n nodes (or $O(n^3)$ for full-information shortest-path routing as in Exercise 6.5.5). Simple computation of the average of the total number of bits used to store the routing scheme over all graphs on n nodes shows that Theorem 6.5.1, Theorem 6.5.2, and Exercise 6.5.2 all hold for the average case.

Exercises

6.5.1. [19] Show that there exist labeled graphs on n nodes such that each local routing function must be stored in at least $\frac{1}{2}n \log \frac{1}{2}n - O(n)$ bits per node (hence the complete routing scheme requires at least $(n^2/2) \log \frac{1}{2}n - O(n^2)$ bits to be stored).

Comments. Source: H.M. Buhrman, J.H. Hoepman, and P.M.B. Vitányi, *SIAM J. Comput.*, 28:4(1999), 1414–1432. This is also the source for the next four exercises.

6.5.2. [22] (a) Show that routing with any stretch factor > 1 in $c \log n$-random graphs can be done with $n - 1 - (c + 3) \log n$ nodes with local

routing functions stored in at most $\log(n+1)$ bits per node, and $1+(c+3)\log n$ nodes with local routing functions stored in $6n$ bits per node (hence the complete routing scheme is represented by fewer than $(6c+20)n\log n$ bits).

(b) Show that routing with stretch factor 2 in $c\log n$-random graphs can be done using $n-1$ nodes with local routing functions stored in at most $\log\log n$ bits per node and 1 node with its local routing function stored in $6n$ bits (hence the complete routing scheme is represented by $n\log\log n+6n$ bits).

(c) Show that routing with stretch factor $(c+3)\log n$ in $c\log n$-random graphs can be done with local routing functions stored in $O(1)$ bits per node (hence the complete routing scheme is represented by $O(n)$ bits).

Comments. Hint: use Lemma 6.4.5 on page 464 and restricted use of tables (Items (a) and (b)) as in the proof of Theorem 6.5.1 and no tables in Item (c).

6.5.3. [31] Prove the following: for shortest-path routing on $c\log n$-random graphs, if nodes know their neighbors and nodes may be relabeled by arbitrary identifiers (which therefore can code information), then with labels of size at most $(1+(c+3)\log n)\log n$ bits the local routing functions can be stored in $O(1)$ bits per node. Hence the complete routing scheme including the label information is represented by $(c+3)n\log^2 n+n\log n+O(n)$ bits.

6.5.4. [34] Show that for shortest-path routing in graphs that are $o(n)$-random, if the neighbors are not known, then the complete routing scheme requires at least $n^2/32-o(n^2)$ bits to be stored. This holds also under a slightly weaker model.

6.5.5. [29] In a *full-information* shortest-path routing scheme, the routing function in u must, for every destination v, return all edges incident to u on shortest paths from u to v. These schemes allow alternative shortest paths to be taken whenever an outgoing link is down. Show that for full-information shortest-path routing on $o(n)$-random graphs, the local routing function requires $n^2/4-o(n^2)$ bits for every node (hence the complete routing scheme requires at least $n^3/4-o(n^3)$ bits to be stored). This is also the trivial upper bound.

6.5.6. [30] In interval routing on a graph $G=(V,E)$, $V=\{1,\ldots,n\}$, each node i has for each incident edge e a (possibly empty) set of pairs of node labels representing disjoint intervals with wraparound. Each pair indicates the initial edge on a shortest path from i to any node in the interval, and for every node $j\neq i$ there is such a pair. We are allowed to permute the labels of graph G to optimize the interval setting.

(a) Show that there are graphs such that for each interval routing scheme some incident edge on each of $\Omega(n)$ nodes are labeled by $\Omega(n)$ intervals.

(b) Show that for every $d \geq 3$ there are graphs of maximal node degree d such that for each interval routing scheme some incident edge on each of $\Omega(n)$ nodes is labeled by $\Omega(n/\log n)$ intervals.

Comments. Source: E. Kranakis and D. Krizanc, *Proc. 13th Symp. Theoret. Aspects Comput. Sci.*, *Lect. Notes Comput. Sci.*, Vol. 1046, Springer-Verlag, 1996, pp. 529–540. Item (b) is improved by C. Gavoile and S. Pérennès [*Proc. 15th ACM Symp. Principles Distr. Comput.*, 1996, pp. 125–133], who showed that for every interval routing scheme, each of $\Omega(n)$ edges is labeled by $\Omega(n)$ intervals. This shows that interval routing can be worse than straightforward coding of routing tables, which can be done in $O(n^2 \log d)$ bits total.

6.5.7. Consider routing schemes for n-node graphs $G = (V, E)$, $V = \{1, \ldots, n\}$, with maximal node degree d. Choose the most convenient labeling to facilitate compact routing schemes.

(a) Show that for every $d \geq 3$ there are networks for which any shortest-path routing scheme requires a total of $\Omega(n^2/\log n)$ bits.

(b) Same as Item (a) but now with stretch factor < 2 requiring a total of $\Omega(n^2/\log^2 n)$ bits.

Comments. Source: E. Kranakis and D. Krizanc, *Ibid.* Item (a) is improved by C. Gavoile and S. Pérennès [*Ibid.*] for $3 \leq d \leq \epsilon n$ $(0 < \epsilon < 1)$ to $\Theta(n^2 \log d)$. This is optimal, since straightforward coding of routing tables takes $O(n^2 \log d)$ bits total.

6.5.8. Consider a computer network consisting of n computers connected in a ring by bidirectional communication channels. The message transmission takes unknown time, but messages do not overtake each other. The computers are *anonymous*, that is, they do not have unique identities. To be able to discuss them individually we number them $1, \ldots, n$. Let x be any string in $\{0,1\}^n$. At the start of the computation every computer i in the ring owns a copy of x and a bit y_i. Define $y \equiv x$ if there is an s $(0 \leq s < n)$ such that $y_{i+s \bmod n} = x_i$ for all i $(1 \leq i \leq n)$. The problem is to compute a Boolean function $f_x : \{0,1\}^n \to \{0,1\}$ defined by $f_x(y) = 1$ if $y \equiv x$ and 0 otherwise. Each computer executes the same algorithm to compute f_x and eventually outputs the value $f_x(y)$. Show that there is an algorithm to compute $f_x(\cdot)$, with $C(x) \geq n - O(\log n)$, on an anonymous ring of n computers using $O(n \log n)$ bit exchanges for a fraction of at least $1 - 1/n$ of all 2^n inputs, and hence $\Theta(n \log n)$ bit exchanges on average.

Comments. S. Moran and M. Warmuth [*SIAM J. Comput.*, 22:2(1993), 379–399] have shown that to compute nonconstant functions f, the computers need to exchange $\Omega(n \log n)$ bits, and that this bound is tight.

This creates a gap with the case of computing constant f requiring zero messages. Source: E. Kranakis, D. Krizanc, and F.L. Luccio, pp. 392–401 in: *Proc. 13th Symp. Math. Found. Comput. Sci.*, Lect. Notes Comput. Sci., Vol. 969, Springer-Verlag, 1995.

6.6 Average-Case Analysis of Sorting

For many algorithms, it is very difficult to analyze their average-case complexity. In average-case analysis, the incompressibility method has an advantage over a probabilistic approach. In the latter approach, one deals with expectations or variances over some ensemble of objects changing over the course of the computation. Using Kolmogorov complexity, we can reason about a fixed incompressible individual object. Because it is incompressible, it has all statistical properties with certainty, rather than having them hold with some (high) probability as in a probabilistic analysis. This fact greatly simplifies the resulting analysis.

6.6.1 Heapsort

Heapsort is a widely used sorting algorithm. One reason for its prominence is that its running time is *guaranteed* to be of order $n \log n$, and it does not require extra memory space. The method was first discovered by J.W.J. Williams [*Comm. Assoc. Comp. Mach.*, 7(1964), 347–348], and subsequently improved by R.W. Floyd. Only recently has one succeeded in giving a precise analysis of its average-case performance. I. Munro has suggested the simple solution using incompressibility presented here.

A 'heap' can be visualized as a complete directed binary tree with possibly some rightmost nodes being removed from the deepest level. The tree has n nodes, each of which is labeled with a different key, taken from a linearly ordered domain. The largest key k_1 is at the root (on top of the heap), and every other node is labeled with a key that is less than the key of its parent.

Definition 6.6.1 Let *keys* be elements of $\mathcal{N}$. An array of keys $k_1, \ldots, k_n$ is a *heap* if they are partially ordered such that

$$k_{\lfloor j/2 \rfloor} \geq k_j \text{ for } 1 \leq \lfloor j/2 \rfloor < j \leq n.$$

Thus, $k_1 \geq k_2, k_1 \geq k_3, k_2 \geq k_4$, and so on. We consider in place sorting of n keys in an array $A[1..n]$ without use of additional memory.

Heapsort {Initially, $A[1..n]$ contains n keys. After sorting is completed, the keys in A will be ordered as $A[1] < A[2] < \cdots < A[n]$}

Heapify: {Regard A as a tree: the root is in $A[1]$; the two children of $A[i]$ are at $A[2i]$ and $A[2i+1]$, when $2i, 2i+1 \leq n$. We convert the tree in A to a heap} **Repeat for** $i = \lfloor n/2 \rfloor, \lfloor n/2 \rfloor - 1, \ldots, 1$: {the subtree rooted at $A[i]$ is now almost a heap except for $A[i]$} push the key, say k, at $A[i]$ down the tree (determine which of the two children of $A[i]$ possesses the greatest key, say k' in child $A[2i+j]$ with j equal 0 or 1); **if** $k' > k$ **then** put k in $A[2i+j]$ and **repeat** this process, pushing k' at $A[2i+j]$ down the tree **until** the process reaches a node that does not have a child whose key is greater than the key now at the parent node.

Sort: Repeat for $i = n, n-1, \ldots, 2$: {$A[1..i]$ contains the remaining heap and $A[i+1..n]$ contains the already sorted list $k_{i+1}, \ldots, k_n$ of largest elements; by definition, the element on top of the heap in $A[1]$ must be k_i} switch the key k_i in $A[1]$ with the key k in $A[i]$, extending the sorted list to $A[i..n]$. Rearrange $A[1..i-1]$ to a heap with the largest element at $A[1]$.

It is well known that the Heapify step can be performed in $O(n)$ time. It is also known that the Sort step takes no more than $O(n \log n)$ time. We analyze the precise average-case complexity of the Sort step. There are two ways of rearranging the heap: Williams's method and Floyd's method.

Williams's Method: {Initially, $A[1] = k$}

Repeat compare the keys of k's two direct children; **if** m is the larger of the two **then** compare k and m; **if** $k < m$ **then** switch k and m in $A[1..i-1]$ **until** $k \geq m$.

Floyd's Method: {Initially, $A[1]$ is empty} Set $j := 1$;

while $A[j]$ is not a leaf **do**:

if $A[2j] > A[2j+1]$ **then** $j := 2j$

else $j := 2j + 1$;

while $k > A[j]$ **do**:

{back up the tree until the correct position for k} $j := \lfloor j/2 \rfloor$;

move keys of $A[j]$ and each of its ancestors one node upward;

Set $A[j] := k$.

The difference between the two methods is as follows. Williams's method goes from the root at the top down the heap. It makes two comparisons with the child nodes and one data movement at every step until the key

k reaches its final position. Floyd's method first goes from the root at the top down the heap to a leaf, making only one comparison at every step. Subsequently, it goes from the bottom of the heap up the tree, making one comparison at each step, until it finds the final position for key k. Then it moves the keys, shifting every ancestor of k one step up the tree. The final positions in the two methods are the same; therefore both algorithms make the same number of key movements. Note that in the last step of Floyd's algorithm, one needs to move the keys carefully up the tree, avoiding swaps that would double the number of moves.

The heap is of height $\log n$. If Williams's method uses $2d$ comparisons, then Floyd's method uses $d + 2\delta$ comparisons, where $\delta = \log n - d$. Intuitively, δ is generally very small, since most elements tend to be near the bottom of the heap. This makes it likely that Floyd's method performs better than Williams's method. We analyze whether this is the case. Assume a uniform probability distribution over the lists of n keys, so that all input lists are equally likely.

Theorem 6.6.1 *With probability going to* 1 *for* $n \to \infty$*, and on average, Heapsort makes* $n \log n + O(n)$ *data movements. Williams's method makes* $2n \log n - O(n)$ *comparisons on average. Floyd's method makes* $n \log n + O(n)$ *comparisons on average.*

Proof. Given n keys, there are $n!$ permutations. Hence we can choose a permutation π of n keys such that

$$C(\pi|n, A, P) \geq \log n! - n,$$

justified by Theorem 2.2.1, page 117. In fact, a $(1 - 1/2^n)$ fraction of all permutations of n keys satisfy this. Here A represents the Heapsort algorithms involved and P represents the reconstruction programs used below. Since $n! \approx n^n e^{-n} \sqrt{2\pi n}$ by Stirling's formula, $\log n! < n \log n - 2n$.

Claim 6.6.1 Let h be the heap constructed by the **Heapify** step with input π that satisfies the last displayed equation. Then,

$$C(h|n, A, P) \geq \log n! - 5n. \tag{6.7}$$

Proof. Assume the contrary, $C(h|n, A, P) < \log n! - 5n$. Then we show how to describe π, using h and n, in fewer than $\log n! - n$ bits as follows. We will encode the **Heapify** process that constructs h from π. At each loop, when we push $k = A[i]$ down the subtree, we record the path that key k traveled: 0 indicates a left branch, 1 means a right branch, 2 means halt. In total, this requires $(n \log 3) \sum_j j/2^{j+1} \leq 2n \log 3$ bits. Given the final heap h and the above description of updating paths, we can reverse the procedure of **Heapify** and reconstruct p. Hence,

$C(\pi|n, A, P) < C(h|n, A, P) + 2n \log 3 + O(1) < \log n! - n$, which is a contradiction. (The term $5n$ above can be reduced by a more careful encoding and calculation.) □

We give a description of h using the history of the $n-1$ heap rearrangements during the Sort step. We need to record, for $i := n-1, \ldots, 2$, at the $(n-i+1)$st round of the Sort step, only the final position where $A[i]$ is inserted into the heap. Both algorithms insert $A[i]$ into the same slot using the same number of data moves, but a different number of comparisons.

We encode such a final position by describing the path from the root to the position. A path can be represented by a sequence s of 0's and 1's, with 0 indicating a left branch and 1 indicating a right branch. Each path i is encoded in self-delimiting form by giving the value $\delta_i = \log n - l(s_i)$ encoded in self-delimiting binary form, followed by the literal binary sequence s_i encoding the actual path. This description requires at most

$$l(s_i) + 2 \log \delta_i \tag{6.8}$$

bits. Concatenate the descriptions of all these paths into sequence H.

Claim 6.6.2 We can effectively reconstruct heap h from H and n.

Proof. Assume that H is known and the fact that h is a heap on n different keys. We simulate the Sort step in reverse. Initially, $A[1..n]$ contains a sorted list with the least element in $A[1]$.

for $i := 2, \ldots, n-1$ **do:** **{**now $A[1..i-1]$ contains the partially constructed heap and $A[i..n]$ contains the remaining sorted list with the least element in $A[i]$**}** Put the key of $A[i]$ into $A[1]$, while shifting every key on the $(n-i)$th path in H one position down starting from the root at $A[1]$. The last key on this path has nowhere to go and is put in the empty slot in $A[i]$.

termination **{**Array $A[1..n]$ contains heap h**}**

□

It follows from Claim 6.6.2 that $C(h|n, A, P) \leq l(H) + O(1)$. Therefore, by Equation 6.7, we have $l(H) \geq \log n! - 5n - O(1)$. By the description in Equation 6.8, we therefore have

$$\sum_{i=1}^{n} (l(s_i) + 2 \log \delta_i) = \sum_{i=1}^{n} ((\log n) - \delta_i + 2 \log \delta_i) \geq \log n! - 5n - O(1).$$

It follows that $\sum_{i=1}^{n}(\delta_i - 2\log\delta_i) \leq 5n$. This is possible only if $\sum_{i=1}^{n}\delta_i = O(n)$. Therefore, the average path length is at least $\log n - c$, for some fixed constant c. In every round of the Sort step the path length equals the number of data moves. The combined total path length is at least $n\log n - nc$.

It follows that starting with heap h, Heapsort performs at least $n\log n - O(n)$ data moves. Trivially, the number of data moves is at most $n\log n$. Together this shows that Williams's method makes $2n\log n - O(n)$ key comparisons, and Floyd's method makes $n\log n + O(n)$ key comparisons.

Since a $(1 - 1/2^n)$ fraction of all permutations π on n keys satisfies $C(\pi|n, A, P) \geq \log n! - n$, these bounds for one such permutation π also hold for all permutations *on average*. □

6.6.2 Shellsort

D.L. Shell [*Comm. Assoc. Comp. Mach.*, 2:7(1959), 30–32)] proposed the Shellsort algorithm in 1959. Since then, the question of the average-case complexity of Shellsort has been open. Recently, the first general lower bound for this problem was proven using the incompressibility method.

Shellsort sorts a list of n elements in p passes using a sequence of increments $h_1, \ldots, h_p$. In the kth pass the main list is divided into h_k separate sublists, called h_k-*chains*, each of length $\lceil n/h_k \rceil$, where the ith sublist consists of the elements at positions j ($j \bmod h_k \equiv i - 1$) of the main list ($i = 1, \ldots, h_k$). Every sublist is sorted using a straightforward Bubblesort or Insertion sort, and $h_p = 1$ to ensure sortedness of the final list.

In Bubblesort or Insertion sort we go from left to right over the list, comparing every key with its right neighbor and switching them if the left key is larger. At the end, the largest key is in the rightmost position. Then repeat this process with the remaining list, and so on.

The efficiency of the Shellsort method is governed by the number of passes p and the selected increment sequence $h_1, \ldots, h_p$. For example, the original $\log n$-pass increment sequence $\lfloor n/2 \rfloor, \lfloor n/4 \rfloor, \ldots, 1$ of Shell uses worst case $\Theta(n^2)$ time. Many increment sequences have been proposed. The elegant method of V.R. Pratt uses all $\log^2 n$ increments of the form $2^i 3^j < \lfloor n/2 \rfloor$ to obtain time $O(n\log^2 n)$ in the worst case. Moreover, since every pass takes at least n steps, the average-case time complexity using Pratt's increment sequence is $\Theta(n\log^2 n)$. D.E. Knuth proved an average-case time complexity of $\Theta(n^{5/3})$ for the best choice of increments in $p = 2$ passes; A.C.C. Yao analyzed the average case for $p = 3$ but did not obtain an analytic form; Yao's analysis was used by S. Janson and D.E. Knuth, who proved an $O(n^{23/15})$ average-case time-complexity upper bound for particular increments in $p = 3$ passes. Apart from this, no nontrivial results had been known for the average case.

The idea of the proof is simple. For every incompressible permutation π, encode the moves of Shellsort in the most compressed manner. If the used

algorithm does not make a certain number of moves, then one obtains too short an encoding of π. Since most permutations are incompressible, like π, the particular bound for an incompressible π holds on average. The average is taken over the uniform distribution on all permutations of n keys.

Theorem 6.6.2 *With probability going to* 1 *for* $n \to \infty$*, and on average,* p*-pass Shellsort takes* $\Omega\left(pn^{1+1/p}\right)$ *steps, for every increment sequence.*

Proof. Let A be a p-pass Shellsort algorithm with increments $(h_1, \ldots, h_p)$, where h_k is the increment in the kth pass and $h_p = 1$. Since the running time is at least pn (every key is compared in every pass), the theorem is true for $p = \Omega(\log n)$. It remains to prove the theorem for $p = o(\log n)$. There are $n!$ permutations of n keys. Choose a permutation π on n keys $\{1, \ldots, n\}$ such that

$$C(\pi|n, A, P) \geq \log n! - n,$$

where P is a constant-size reconstruction program to be specified later.

For all $1 \leq i \leq n$ and $1 \leq k \leq p$, let $m_{i,k}$ be the distance the ith key moves in the h_k-chain containing key i, in pass k. Then,

$$M = \sum_{k=1}^{p} \sum_{i=1}^{n} m_{i,k}$$

is precisely the number of data movements made by A to sort π, and therefore is a lower bound on the time complexity T of A.

Claim 6.6.3 Given all the $m_{i,k}$'s in lexicographic order, we can reconstruct the original permutation π.

Proof. Given $m_{i,k}$, for $i = 1, \ldots, n$, and the final permutation of pass k, we can reconstruct the initial permutation of pass k. □

The lexicographically ordered $m_{i,k}$'s can be described as a partition of M in nonnegative integer summands. There are

$$D(M) = \binom{M + np - 1}{np - 1}$$

distinct partitions of M into np ordered nonnegative integral $m_{i,k}$'s. Since every $m_{i,k} \leq n$ and $p \leq n$, we have $M \leq n^3$. Given n, we can first describe p and M self-delimitingly in $O(\log n)$ bits, and second describe the partition of M, yielding the lexicographically ordered $m_{i,k}$'s, in total $O(\log n) + \log D(M)$ bits.

By Claim 6.6.3, and letting P be the program reconstructing π from this description, given n, p, A, we must have

$$\log D(M) + O(\log n) \geq C(\pi|n, A, P) \geq \log n! - n.$$

Rewriting $\log D(M)$ using the identity in Exercise 1.3.3 on page 10, we can estimate the resulting terms asymptotically for $n \to \infty$ and $p = o(\log n)$, yielding

$$M = \Omega(pn^{1+1/p}).$$

The running time T of the algorithm A on permutation π satisfies $T \geq M$. The number of permutations with $C(\pi|n, A, P) \geq \log n! - n$ is at least a $(1 - 1/2^n)$ fraction of all permutations, which proves the theorem. □

Example 6.6.1 The question whether there exists an increment sequence for Shellsort to achieve $O(n \log n)$ average performance is still open. Theorem 6.6.2 implies that such an increment sequence, if it exists, must be of length $\Theta(\log n)$. ◇

Example 6.6.2 The initial idea to prove Theorem 6.6.2 was to simply to describe the $m_{i,k}$'s by standard self-delimiting codes, giving a total length of

$$\sum_{k=1}^{p} \sum_{i=1}^{n} (\log m_{i,k} + 2 \log \log m_{i,k} + 1).$$

Then,

$$\sum_{k=1}^{p} \sum_{i=1}^{n} (\log m_{i,k} + 2 \log \log m_{i,k} + 1) \geq C(\pi|n, A, P) \geq \log n! - n.$$

By the concavity of the logarithm function, the left-hand side of the above is maximized when all the $m_{i,k}$'s are equal, say m. Therefore,

$$np \log m + 2np \log \log m + np \geq \log n! - n,$$

and since $\log n! \geq n \log n - 2n$, we have

$$m = \Omega\left(\frac{n^{1/p}}{((\log n)/p)^2}\right) \text{ and } T \geq pnm \geq \Omega\left(\frac{pn^{1+1/p}}{((\log n)/p)^2}\right).$$

This is the result of Theorem 6.6.2 for $p = \Theta(\log n)$, but the less optimal code results in a slightly weaker result for $p = o(\log n)$. ◇

Exercises

6.6.1. [25] Consider the following game. Carole chooses a number from $\{1, 2, \ldots, n\}$. Paul has to guess the secret number using only "yes/no" questions. Prove the following lower bounds on the number of questions needed for Paul to determine the number: (i) $\log n$ if Carole answers every question truthfully; (ii) $\log n + \log\log n$ if Carole is allowed to lie at most once; (iii) $\log n + \log \sum_{i \le k} \binom{n}{i}$ if Carole is allowed to lie at most k times; (iv) $\log n + \log\log n + k$ if Carole is allowed to lie at most k times, but all lies (possibly fewer than k and possibly nonconsecutive) have to occur in k consecutive rounds.

Comments. Simple proofs using the incompressibility method are due to M. Fouz, CS798 Course Report, University of Waterloo, December 2007. The one-lie game was fully analyzed by A. Pelc in [*J. Comb. Theory, Ser. A*, 44:1(1987), 129–140]. J.H. Spencer generalized this result to the k-lies game in [*Theoret. Comput. Sci.*, 95:2(1992), 307–321]. The interval variant was introduced and analyzed by B. Doerr, J. Lengler, D. Steurer in [*Proc. 17th Int. Symp. Algor. Comput.*, Lect. Notes Comput. Sci., Vol. 4288, Springer-Verlag, Berlin, 2006, 318–327].

6.6.2. [40] In computational biology, evolutionary trees are represented by unrooted unordered binary trees with uniquely labeled leaves and unlabeled internal nodes. Measuring the distance between such trees is useful in biology. A *nearest neighbor interchange* (*nni*) operation swaps two subtrees that are separated by an internal edge (u, v), as shown in Figure 6.2.

(a) Show that in Figure 6.3 it takes 2 nni moves to convert (i) to (ii).

(b) Show that $n \log n + O(n)$ nni moves are sufficient to transform a tree of n leaves to any other tree with the same set of leaves.

(c) Prove an $\Omega(n \log n)$ lower bound for Item (b), using the incompressibility method.

Comments. Item (b) is from [K. Culik II and D. Wood, *Inform. Process. Lett.*, 15(1982), 39–42; M. Li, J.T. Tromp, and L. Zhang, *J. Theoret. Biology*, 182(1996), 463–467]. The latter paper contains principal references related to the nni metric. Item (c) is by D. Sleator, R.E. Tarjan, and W. Thurston [*SIAM J. Discr. Math.*, 5(1992), 428–450], who proved the $\Omega(n \log n)$ lower bound for a more general graph transformation system.

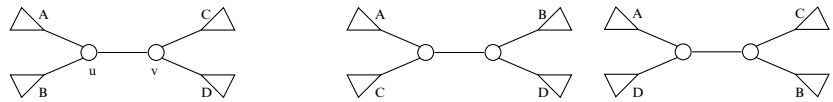


FIGURE 6.2. The two possible nnis on (u, v): swap $B \leftrightarrow C$ or $B \leftrightarrow D$

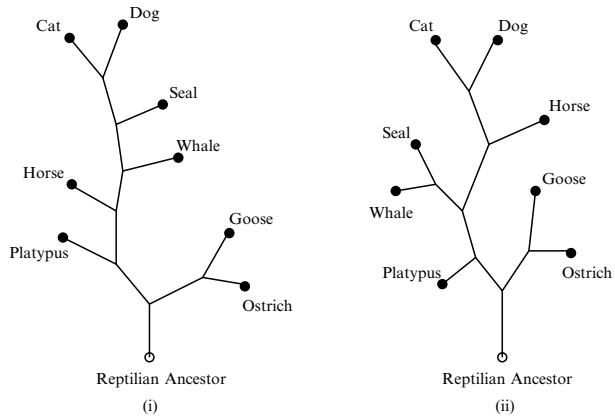


FIGURE 6.3. The nni distance between (i) and (ii) is two

6.6.3. [25] Improve the $\log n! - 5n$ bound in Equation 6.7, page 478, by reducing $5n$ via a better encoding and more precise calculation.

6.6.4. [41] Show that the worst-case time complexity of p-pass Shellsort of n items is at least $\Omega(n \log^2 n/(\log \log n)^2)$ for every number p of passes and every increment sequence.

Comments. This shows that the best possible average-case time complexity of Shellsort for any number of passes and all increment sequences may be of larger order of magnitude than $n \log n$. Source: C.G. Plaxton, B. Poonen, T. Suel, *Proc. 33rd IEEE Symp. Found. Comput. Sci.*, pp. 226–235, 1992.

6.6.5. [O48] (a) Prove or disprove that there is a number of passes p and an increment sequence such that Shellsort has average-case time complexity $O(\log n)$.

(b) Find a better lower bound on average-case time complexity of Shellsort than Theorem 6.6.2; give a good or optimal upper bound on average-case time complexity of p-pass Shellsort for the best increment sequences.

Comments. See Exercise 6.6.4 and the comment following the proof of Theorem 6.6.2. Source: M. Li and P.M.B. Vitányi, *J. Assoc. Comp. Mach.*, 47:5(2000), 905–911.

6.6.6. [10] Use the idea in the proof for Theorem 6.6.2 to obtain $\Omega(n^2)$ average-case lower bounds for Bubblesort, Selection sort, and Insertion sort.

6.6.7. [22/O46] Sorting by stacks. The input is a permutation of n integers. These integers, one at a time, pass through a sequence of m first-in-last-out stacks $S_1, \ldots, S_m$, from S_1 to S_m. If an integer k is to

be pushed on S_i, then this stack can pop some integers from the top down, pushing them on S_{i+1} in that order, before pushing k into S_i. The output sequence from S_m gives the final permutation.

(a) Show that we can sort integers with $\log n$ stacks.

(b) Use the incompressibility method to show that $\frac{1}{2}\log n$ stacks are needed on average.

(c) Open: Close the gap between Item (a) and (b).

Comments. The problem was investigated by R.E. Tarjan [*J. Assoc. Comp. Mach.*, 19(1972), 341–346] and D.E. Knuth [*The Art of Computer Programming, Vol. 3: Sorting and Searching*, 1998 (2nd edition), Section 5.2.4, Exercises 19 and 20]. Item (b), and related studies such as sorting with parallel stacks and queues can be found in [T. Jiang, M. Li and P.M.B. Vitányi, *J. Comput. Sci. Tech.*, 15:5(2000), 402–408].

6.6.8. [36/O39] Consider the following algorithm.
QuickSort(Array $\pi[1..n]$): **If** $n = 1$ **then return** π; $p := \pi[1]$; $\pi_L := (x \in \pi, x < p)$ in stable order; $\pi_R := (x \in \pi, x > p)$ in stable order; QuickSort(π_L); QuickSort(π_R); $\pi := \pi_L p \pi_R$

(a) Use the incompressibility method to show that the average height of a Quicksort tree (its deepest recursion level), or equivalently a binary insertion tree, is $O(\log n)$. This also gives an alternative $O(n \log n)$ average-case analysis of Quicksort.

(b) Obtain the $4.31107 \log n$ upper bound using the incompressibility method.

Comments. Hint: Consider a pivot sequence $(p_1, p_2, \ldots, p_k)$, where p_{i+1} is a pivot for one of the subranges defined by p_i for all i. The longest such sequence corresponds to the binary search tree height. This sequence can be encoded efficiently. Suppose π has a pivot chain of length $c \log n$, where c is sufficiently large. Let x be the string of length $c \log n$ such that $x[i] = 1$ iff p_i occurs in the middle half of its range. Let z be the string of length $c \log n$ such that $z[i] = 1$ iff p_{i+1} is the pivot for the smaller range defined by p_i. Then the number of ones in x is at most $c' \log n$, where $c' = 1/\log \frac{4}{3}$, and the number of ones in z is at most $\log n$, since otherwise the size of the ranges for the p_i will reach 1. Now note that if we are given x and z, we can save one bit for every entry p_i in π, since p_i begins in 01 or 10 iff $x[i] = 1$, and z tells us which of p_i's subpivots is p_{i+1}. Thus, given x and z, we save $c \log n$ bits from the encoding of π. Now π can be recursively encoded by $\log(A!) = \log \binom{A}{B} + \log(B!) + \log((A-B)!)$ while compressing the pivots along the pivot sequence. Source: B. Lucier, T. Jiang, and M. Li, *Inform. Process. Lett.*, 103:2(2007), 45–51. A tight bound for this problem by probabilistic analysis is given in [L. Devroye, *J. Assoc. Comp. Mach.* 33:3(1986), 480–498].

6.6.9. [22] Consider two variants of p-pass Shellsort. In each pass, instead of fully sorting every sublist, we make only one pass of Bubblesort, or two such passes in opposite directions, for every sublist. In both cases the sequence may stay unsorted, even if the last increment is 1. A final phase, a straight insertion sort, is required to guaranty a fully sorted list.

(a) Prove an $\Omega(n^2/2^p)$ lower bound on the average-case running time for the one-pass variant of p-pass Shellsort.

(b) Prove an $\Omega(n^2/4^p)$ lower bound on the average-case running time for the two-pass variant of p-pass Shellsort.

Comments. The one-pass variant of Shellsort is called Dobosiewicz sort by D.E. Knuth [*The Art of Computer Programming, Vol. 3, Sorting and Searching*, Addison-Wesley, 1973, Exercise 5.2.1.40, page 105]. Source: W. Dobosiewicz, *Inform. Process. Lett.* 11:1(1980), 5–6. The two-pass variant, proposed by J. Incerpi and R. Sedgewick [*Inform. Process. Lett.* 26:1(1980), 37–43], is called Shakersort. Solutions for both (a) and (b) were given by B. Brejová [*Inform. Process. Lett.*, 79:5(2001), 223–227].

6.7 Longest Common Subsequence

Certain problems concerning subsequences and supersequences of a given set of sequences arise naturally in quite practical situations. For example, in molecular biology, the longest common subsequence of some DNA sequences is commonly used as a measure of similarity of these sequences. Other applications of longest common subsequences include data compression and syntactic pattern recognition.

Definition 6.7.1 If $s = s_1 \dots s_m$ and $t = t_1 \dots t_n$ are two sequences, then s is a *subsequence* of t, and equivalently, t is a *supersequence* of s, if for some sequence of indices $i_1 < \cdots < i_m$, we have $s_j = t_{i_j}$ for all j $(1 \leq j \leq m)$. Given a finite set of sequences S, a *shortest common supersequence* (SCS) of S is a shortest sequence s such that each sequence in S is a subsequence of s. A *longest common subsequence* (LCS) of S is a longest sequence s such that each sequence in S is a supersequence of s.

It is well known that the SCS and LCS problems are NP-hard. In the worst case, the SCS and LCS problems cannot even be efficiently approximated unless P = NP. For example, the following is known for the LCS problem. If there is a polynomial-time algorithm that on some input sequences always finds a common subsequence of length $c > 0$ times the length of the longest common subsequence, then P = NP. This holds also for the problem as stated in the theorem below. However, many simple heuristic algorithms for SCS and LCS turn out to work well in practice. An incompressibility argument shows that indeed, these algorithms perform well *on average*.

Definition 6.7.2 Consider LCS problems on an alphabet $\Sigma = \{a_1, \ldots, a_k\}$. Let $\mathrm{lcs}(S)$ denote the length of an LCS of a set $S \subseteq \Sigma^*$ of sequences.

Algorithm Long-Run

Step 1. Determine the greatest m such that a^m is a common subsequence of all input sequences, for some $a \in \Sigma$.

Step 2. Output a^m as a common subsequence.

Theorem 6.7.1 *Assume the notation above. Let $S \subseteq \Sigma^*$ be a set of n sequences each of length n, and let $\epsilon > 0$ be a constant. Algorithm Long-Run outputs a common subsequence of S of length $\mathrm{lcs}(S) - O(\mathrm{lcs}(S)^{1/2+\epsilon})$ for a fraction of least $1 - 1/n^2$ of all inputs, and hence on average.*

Proof. Assume the notation above. Fix a string x of length n^2 over Σ with

$$C(x) \geq (n^2 - 2\log n)\log k\,. \tag{6.9}$$

Divide x into n equal-length segments $x_1, \ldots, x_n$. Choose the set S in the theorem as $S = \{x_1, \ldots, x_n\}$.

The following claim is a corollary of the proof of Theorem 2.6.1 on page 170, counting each letter as a block of size $\log k$, assuming that k is a power of 2.

Claim 6.7.1 Let $a \in \Sigma, x_i \in S$, and let $\epsilon > 0$ be a constant. Denote the number of occurrences of a in x_i by m. If $|m - n/k| > n^{1/2+\epsilon}$, then there is a constant $\delta > 0$ such that

$$C(x_i|k) \leq (n - \delta n^{2\epsilon})\log k\,.$$

A direct proof of this claim is also easy. There are only $D = \binom{n}{m}(k-1)^{n-m}$ strings of length n with m occurrences of a. Therefore, one can specify x_i by n, k, m and its index j, with $l(j) = \log D$, in this ensemble. An elementary estimate by Stirling's formula yields, for some $\delta > 0$,

$$\log \binom{n}{m}(k-1)^{n-m} \leq (n - \delta n^{2\epsilon})\log k.$$

Claim 6.7.2

$$\mathrm{lcs}(S) < \frac{n}{k} + n^{1/2+\epsilon}.$$

Proof. Let s be an LCS of S. Then $l(s) = \text{lcs}(S) \leq n$ by definition. Assume, by way of contradiction, that $l(s)$ is greater than claimed in the lemma. We give a short description of x, for some fixed $\delta > 0$, by saving $n^\delta \log k$ bits on the description of every x_i through the use of s.

Let $s = s_1 s_2 \ldots s_p$, with $p = l(s)$. Fix an x_i. We will do another encoding of x_i. We align s with the corresponding letters in x_i as far to the left as possible, and rewrite

$$x_i = \alpha_1 s_1 \alpha_2 s_2 \ldots \alpha_p s_p z.$$

Here α_1 is the longest prefix of x_i containing no s_1; α_2 is the longest substring of x_i starting from s_1 containing no s_2; and so on. The string z is the remaining part of x_i after s_n. In this way, α_j does not contain an occurrence of letter s_j, for $j = 1, \ldots, p$. That is, every α_j contains only letters in $\Sigma - s_j$.

Then x_i can be considerably compressed with the help of s. Divide $x_i = yz$ such that the prefix y is

$$y = \alpha_1 s_1 \alpha_2 s_2 \ldots \alpha_p s_p.$$

From s we can reconstruct which $k-1$ letters from Σ appear in α_i, for every i. We map y to an equally long string y' as follows: For $i = 1, \ldots, p$, change s_i to a_k, in y. Moreover, change each occurrence of a_k in α_j to the letter s_j. We can map y' back to y, using s, by reversing this process (because the original α_j did not contain an occurrence of s_j).

The letter a_k occurs at least $(n/k) + n^{1/2+\epsilon}$ times in y', since $l(s)$ is at least this long. Then, by Claim 6.7.1, for some constant $\delta > 0$, we have

$$C(y'|k) \leq (l(y') - \delta n^{2\epsilon}) \log k \,.$$

From y', s, z, k we can reconstruct x_i. (We have defined $x_i = yz$.) Giving also the lengths of y', s, z in self-delimiting format in $O(\log n)$ bits, we can describe x_i, given k and s, by the number of bits in the right side of the equation below (using $l(y') + l(z) \leq n$):

$$C(x_i|k, s) \leq (n - \delta n^{2\epsilon}) \log k + O(\log n). \tag{6.10}$$

We repeat this for every x_i. In total, we save more than $\Omega(n^{1+2\epsilon} \log k)$ bits to encode x. Thus,

$$\frac{C(x|k)}{\log k} \leq n^2 - \Omega(n^{1+2\epsilon}) + l(s) + O(n \log n) < n^2 - 2 \log n.$$

This contradicts the incompressibility of x asserted in Equation 6.9. □

It follows from Claim 6.7.1 and Equation 6.9, by repeating the argument following Equation 6.10 in the proof of Claim 6.7.2, that for some $\epsilon > 0$,

each $a \in \Sigma$ occurs in each $x_1, \ldots, x_n$ at least $(n/k) - O(n^{1/2+\epsilon})$ times. This means that a^m with

$$l(m) = \frac{n}{k} - O(n^{1/2+\epsilon}) \tag{6.11}$$

is a common subsequence of $x_1, \ldots, x_n$. By Claim 6.7.2, $\mathrm{lcs}(S) - l(m) = O(n^{1/2+\epsilon})$.

Altogether, we have shown that the statement in the theorem is true for this particular input $x_1, \ldots, x_n$. The fraction of strings of length n^2 satisfying the theorem is at least $1 - 1/n^2$, since that many strings satisfy Equation 6.9. The theorem follows by taking the average. □

Exercises

6.7.1. [35/O41] (a) Prove that the expected length of the longest common subsequence of two random binary sequences of length n is bounded above by $0.867n$.

(b) Open: Obtain tight bounds on expected length of the longest common subsequence of two random binary sequences of length n.

Comments. Hint: use the same encoding scheme as in Section 6.7 and count the number of encodings. The number 0.867 is roughly the root of the equation $x - 2x \log x - 2(1-x)\log(1-x) = 2$. Source: T. Jiang, M. Li, and P.M.B. Vitányi, *Comput. J.*, 42:4(1999), 287–293. R.A. Baeza-Yates, R. Gavaldà, G. Navarro, and R. Scheihing [*Theor. Comput. Syst.*, 32:4(1999), 435–452] generalized the above analysis to alphabet size $k > 2$, and improved the constant to 0.860. This bound was first proved by V. Chvátal and D. Sankoff in [*J. Appl. Probab.*, 12(1975), 306–315]. The current best lower and upper bounds are $0.7739n$ and $0.8376n$, respectively, due to V. Dančik and M. Paterson [*Random Struct. Alg.*, 6(1995), 449–458; *Proc. 19th Symp. Math. Found. Comput. Sci.*, 1994, pp. 127–142].

6.7.2. [39] Consider the SCS problem defined in Section 6.7. Prove by incompressibility the following: Let $S \subseteq \Sigma^*$ be a set of n sequences of length n, and let $\delta = \sqrt{2}/2 \approx 0.707$. Let $\mathrm{scs}(S)$ be the length of an SCS of S. The algorithm Majority-Merge below produces a common supersequence of length $\mathrm{scs}(S) + O(\mathrm{scs}(S)^{\delta})$ on the average.

Algorithm Majority-Merge {Input: n sequences, each of length n}

Step 1. Set supersequence $s := \epsilon$. {ϵ is the null string}

Step 2. {Let the letters a form a majority among the leftmost letters of the remaining sequences} Set $s := sa$ and delete the front a from these sequences. Repeat this step until no sequences are left.

Step 3. Output s.

Comments. Source: T. Jiang and M. Li, *SIAM J. Comput.*, 24:5(1995), 1122–1139. Part of the proof was from [D. Foulser, M. Li, and Q. Yang, *Artificial Intelligence*, 57(1992), 143–181].

6.8 Formal Language Theory

Part of formal language theory consists in establishing a hierarchy of language families. The main division is the Chomsky hierarchy, with regular languages, context-free languages, context-sensitive languages, and recursively enumerable languages.

A 'pumping' lemma (for regular languages) shows that some languages are not regular, but often does not decide which languages are regular and which languages are not. There are many different pumping lemmas, each of them appropriate for limited use. Therefore, some effort has been made to present pumping lemmas that are exhaustive, in the sense that they *characterize* the regular languages [J. Jaffe, *SIGACT News*, 10:2(1978), 48–49; D. Stanat and S. Weiss, *SIGACT News*, 14:1(1982), 36–37; A. Ehrenfeucht, R. Parikh, and G. Rozenberg, *SIAM J. Comput.*, 10(1981), 536–541]. These pumping lemmas are complicated and hard to use, while the last reference uses Ramsey theory. Using incompressibility we find a characterization of the regular languages that makes our intuition about the finite stateness of these languages rigorous and that is easy to apply.

Definition 6.8.1 Let Σ be a finite nonempty alphabet, and let Q be a (possibly infinite) nonempty set of states. A *transition function* is a function $\delta : \Sigma \times Q \to Q$. We extend δ to δ' on Σ^* by $\delta'(\epsilon, q) = q$ and

$$\delta'(a_1 \dots a_n, q) = \delta(a_n, \delta'(a_1 \dots a_{n-1}, q)).$$

Clearly, if δ' is not one-to-one, then the automaton forgets because some x and y from Σ^* drive δ' into the same memory state. An *automaton* A is a quintuple $(\Sigma, Q, \delta, q_0, Q_f)$, where everything is as above, and $q_0 \in Q$ is a distinguished *initial state* and $Q_f \subseteq Q$ is a set of *final states*. We call A a *finite automaton* (FA) if Q is finite.

An alternative way of looking at it is as follows: We denote 'indistinguishability' of a pair of histories $x, y \in \Sigma^*$ by $x \sim y$, defined as $\delta'(x, q_0) = \delta'(y, q_0)$. 'Indistinguishability' of strings is reflexive, symmetric, transitive, and right-invariant ($\delta'(xz, q_0) = \delta'(yz, q_0)$ for all z). Thus, 'indistinguishability' is a right-invariant equivalence relation on Σ^*. It is a simple matter to ascertain this formally.

Definition 6.8.2 The language accepted by automaton A as above is the set $L = \{x : \delta'(x, q_0) \in Q_f\}$. A *regular language* is a language accepted by a finite automaton.

It is a straightforward exercise to verify from the definitions the following fact (which will be used later):

Theorem 6.8.1 **(Myhill, Nerode)** *The following statements are equivalent.*

(i) $L \subseteq \Sigma^*$ *is accepted by some finite automaton.*

(ii) L *is the union of equivalence classes of a right-invariant equivalence relation of finite index on* Σ^*.

(iii) *For all* $x, y \in \Sigma^*$ *define right-invariant equivalence* $x \sim y$ *by the following: for all* $z \in \Sigma^*$ *we have* $xz \in L$ *iff* $yz \in L$. *Then the number of* $\sim$ *equivalence classes is finite.*

Subsequently, closure of finite automaton languages under complement, union, and intersection follows by simple construction of the appropriate δ functions from given ones. Details can be found in any textbook on the subject.

Example 6.8.1 Consider the language $\{0^k 1^k : k \geq 1\}$. If it were regular, then the state q of the accepting finite automaton A, subsequent to processing 0^k, together with A, is a description of k. Namely, by running A, initialized in state q, on input consisting of only 1's, the first time A enters an accepting state is after precisely k consecutive 1's. The size of the description of A and q is bounded by a constant, say c, that is independent of k. Altogether, it follows that $C(k) \leq c + O(1)$. But choosing k with $C(k) \geq \log k$ we obtain a contradiction for all large enough k. We generalize this observation in the lemma below. ◇

Lemma 6.8.1 **(KC-regularity)** *Let* $L \subseteq \Sigma^*$ *be regular,* $L_x = \{y : xy \in L\}$. *There is a constant* c *such that for every* x, *if* y *is the* n*th string in* L_x, *then* $C(y) \leq C(n) + c$.

Proof. Let L be a regular language. A string y such that $xy \in L$ for some x can be described by

- this discussion, and a description of the automaton that accepts L;
- the state of the automaton after processing x, and the number n.

The first item requires $O(1)$ bits. Thus $C(y) \leq C(n) + O(1)$. □

Example 6.8.2 Prove that $\{1^p : p \text{ is prime}\}$ is not regular. Consider the string $xy = 1^p$ with p the $(k+1)$th prime. Set $x = 1^{p'}$, with p' the kth prime. Then $y = 1^{p-p'}$, and y is the lexicographic first element in L_x. Hence, by Lemma 6.8.1, $C(p-p') = O(1)$. But the difference between two consecutive primes grows unboundedly. Since there are only $O(1)$ descriptions of length $O(1)$, we have a contradiction. ◇

Example 6.8.3 Prove that $L = \{xx^Rw : x, w \in \{0,1\}^* - \{\epsilon\}\}$ is not regular (if $x = x_1 \dots x_m$, then $x^R = x_m \dots x_1$). Set $x = (01)^m$, where $C(m) \geq \log m$. Then the lexicographically first word in L_x is $y = (10)^m 0$. Hence, $C(y) = \Omega(\log m)$, contradicting the KC-regularity lemma. ◇

Example 6.8.4 Prove that $L = \{0^i 1^j : \gcd(i,j) = 1\}$ is not regular. Set $x = 0^{(p-1)!}1$, where $p > 3$ is a prime, $l(p) = n$, and $C(p) \geq \log n - \log\log n$. Then the lexicographically first word in L_x is 1^{p-1}, contradicting the KC-regularity lemma. ◇

Example 6.8.5 Prove that $\{p : p \text{ is the standard binary representation of a prime}\}$ is not regular. Suppose the contrary, and p_i denotes the ith prime, $i \geq 1$. Consider the least binary $p_m = uv$ $(= u2^{l(v)} + v)$, with $u = \Pi_{i<k} p_i$ and v not in $\{0\}^*\{1\}$. Such a prime p_m exists, since every interval $[n, n + n^{11/20}]$ of the natural numbers contains a prime [D. Heath-Brown and H. Iwaniec, *Invent. Math.,* 55(1979), 49–69].

Consider p_m now as an integer, $p_m = 2^{l(v)}\Pi_{i<k} p_i + v$. Since the integer $v > 1$, and v is not divided by any prime less than p_k (because p_m is prime), the binary length $l(v)$ is at least $l(p_k)$. Because p_k goes to infinity with k, the value $C(v) \geq C(l(v))$ also goes to infinity with k. But since v is the lexicographic first suffix, with integer $v > 1$ such that $uv \in L$, we have $C(v) = O(1)$ by the KC-regularity lemma, which is a contradiction. ◇

Characterizations (such as the Myhill–Nerode theorem, Theorem 6.8.1) of regular languages seem to be practically useful only to show regularity. The need for pumping lemmas stems from the fact that characterizations tend to be very hard to use to show nonregularity. In contrast, the compressibility characterization below is useful for both purposes.

Definition 6.8.3 Enumerate $\Sigma^* = \{y_1, y_2, \dots\}$ with y_i the ith element in the total order. For $L \subseteq \Sigma^*$ and $x \in \Sigma^*$, let $\chi = \chi_1\chi_2 \dots$ be the *characteristic sequence* of $L_x = \{y : xy \in L\}$, defined by $\chi_i = 1$ if $xy_i \in L$, and $\chi_i = 0$ otherwise. We denote $\chi_1 \dots \chi_n$ by $\chi_{1:n}$.

Theorem 6.8.2 **(Regular KC-characterization)** *There is a constant c_L depending only on $L \subseteq \Sigma^*$ such that the following statements are equivalent:*

(i) *L is regular;*

(ii) *for all $x \in \Sigma^*$, for all n, $C(\chi_{1:n}|n) \leq c_L$;*

(iii) *for all $x \in \Sigma^*$, for all n, $C(\chi_{1:n}) \leq C(n) + c_L$;*

(iv) *for all $x \in \Sigma^*$, for all n, $C(\chi_{1:n}) \leq \log n + c_L$.*

Proof. (i) $\rightarrow$ (ii). By similar proof as for the KC-regularity lemma.

(ii) $\rightarrow$ (iii) $\rightarrow$ (iv). Trivial.

(iv) $\rightarrow$ (i): By (iv) and Claim 6.8.1 below, there are only finitely many distinct χ's associated with the x's in Σ^*. Define the right-invariant equivalence relation $\sim$ by $x \sim x'$ if $\chi = \chi'$. This relation induces a partition of Σ^* into equivalence classes $[x] = \{y : y \sim x\}$. Since there is a one-to-one correspondence between the $[x]$'s and the χ's, and there are only finitely many distinct χ's, there are also only finitely many $[x]$'s. This implies that L is regular by the Myhill–Nerode theorem: define a finite automaton using one state for each equivalent class, and define the transition function accordingly. The proof of the theorem is finished, apart from proving Claim 6.8.1.

Claim 6.8.1 For each constant c there are only finitely many sequences $\omega \in \{0,1\}^\infty$ such that for all n, we have $C(\omega_{1:n}) \leq \log n + c$.

This claim is a weaker version of Item (e) of Exercise 2.3.4, page 131, which recalls that D.W. Loveland in [*Inform. Contr.*, 15(1969), 510–526] credits the following result to A.R. Meyer: For each constant c there are only finitely many $\omega \in \{0,1\}^\infty$ with $C(\omega_{1:n}|n) \leq c$ for all n and each such ω is a recursive real. G.J. Chaitin [*Theoret. Comput. Sci.*, 2(1976), 45–48] improves the condition first to $C(\omega_{1:n}) \leq C(n) + c$, and then further to $C(\omega_{1:n}) \leq \log n + c$. We provide an alternative and simpler proof, which is sufficient for our purpose, avoiding establishing that the ω's are recursive reals.

Proof. Let c be a positive constant, and let

$$A_n = \{x \in \{0,1\}^n : C(x) \leq \log n + c\}, \tag{6.12}$$
$$A = \{\omega \in \{0,1\}^\infty : \forall_{n \in \mathcal{N}}[C(\omega_{1:n}) \leq \log n + c]\} .$$

If the cardinality $d(A_n)$ of A_n dips below a fixed constant c' for infinitely many n, then c' is an upper bound on $d(A)$. This is because it is an upper bound on the cardinality of the set of prefixes of length n of the elements in A for *all* n.

Fix any $l \in \mathcal{N}$. Choose a binary string y of length $2l + c + 1$ satisfying

$$C(y) \geq 2l + c + 1. \tag{6.13}$$

Choose i maximal such that for the division of y into mn with $l(m) = i$ we have

$$m \leq d(A_n). \tag{6.14}$$

(This holds at least for $i = 0 = m$.) Define similarly a division $y = sr$ with $l(s) = i + 1$. By maximality of i, we have $s > d(A_r)$. From the easily proven $s \leq 2m + 1$, it then follows that

$$d(A_r) \leq 2m. \tag{6.15}$$

We prove $l(r) \geq l$. Since by Equations 6.14 and 6.12 we have

$$m \leq d(A_n) \leq 2^c n,$$

it follows that $l(m) \leq l(n) + c$. Therefore,

$$2l + c + 1 = l(y) = l(n) + l(m) \leq 2l(n) + c,$$

which implies that $l(n) > l$. Consequently, $l(r) = l(n) - 1 \geq l$.

We prove $d(A_r) = O(1)$. By dovetailing the computations of the reference universal machine U (Theorem 2.1.1, page 105) for all programs p with $l(p) \leq \log n + c$, we can enumerate all elements of A_n. We can reconstruct y from the mth element, say y_0, of this enumeration. Namely, from y_0 we reconstruct n, since $l(y_0) = n$, and we obtain m by enumerating A_n until y_0 is generated. By concatenation we obtain $y = mn$. Therefore,

$$C(y) \leq C(y_0) + O(1) \leq \log n + c + O(1). \tag{6.16}$$

From Equation 6.13 we have

$$C(y) \geq \log n + \log m. \tag{6.17}$$

Combining Equations 6.16 and 6.17, it follows that $\log m \leq c + O(1)$. Therefore, by Equation 6.15,

$$d(A_r) \leq 2^{c+O(1)}.$$

Here, c is a fixed constant independent of n and m. Since $l(r) \geq l$ and we can choose l arbitrarily, $d(A_r) \leq c_0$ for a fixed constant c_0 and infinitely many r, which implies $d(A) \leq c_0$, and hence the claim. □ □

The KC-regularity lemma may be viewed as a corollary of the KC-characterization theorem. If L is regular, then trivially L_x is regular. It follows immediately that there are only finitely many associated χ's, and each can be specified in at most c bits, c a constant depending only on L. If y is, say, the mth string in L_x, then we can specify y as the string corresponding to the mth 1 in χ, using only $C(m) + O(1)$ bits to specify y (absorbing c in the $O(1)$ term). Hence, $C(y) \leq C(m) + O(1)$.

Exercises

6.8.1. [10] The KC-regularity lemma can be generalized in several ways. Prove the following version. Let L be regular and $L_x = \{y : xy \in L\}$. Let ϕ be a partial recursive function depending only on L that enumerates strings in Σ^*. For each x, if y is the nth string in the complement of L_x enumerated by ϕ, then $C(y) \leq C(n)+c$, with c a constant depending only on L. Use this generalization to give an alternative proof of Example 6.8.4.

Comments. Source: M. Li and P. Vitányi, *SIAM J. Comput.*, 24:2(1995), 398–410.

6.8.2. [10] Prove that $\{0^n1^m : m > n\}$ is not regular.

6.8.3. [18] Prove that $L = \{x\#y : x$ appears (possibly nonconsecutively) in $y\}$ is not regular.

6.8.4. [20] Prove that $L = \{x\#y :$ at least half of x is a substring in $y\}$ is not regular.

6.8.5. [20] Prove that $L = \{x\#y\#z : xy = z\}$ is not regular.

6.8.6. [37] A DCFL language is a language that is accepted by a deterministic pushdown automaton.

(a) Show that $\{xx^R : x \in \Sigma^*\}$ and $\{xx : x \in \Sigma^*\}$ are not DCFL languages, using an incompressibility argument.

(b) Similar to Lemma 6.8.1, the following is a criterion separating DCFL from CFL. Prove it. Let $L \subseteq \Sigma^*$ be a DCFL and c a constant. Let x and y be fixed finite words over Σ and ω a recursive sequence over Σ. Let u be a suffix of $yy \ldots yx$, v a prefix of ω, and $w \in \Sigma^*$ such that

1. v can be described in c bits given L_u in lexicographic order;
2. w can be described in c bits given L_{uv} in lexicographic order; and
3. $C(v) \geq 2 \log \log l(u)$.

Then there is a constant c' depending only on L, c, x, y, ω such that $C(w) \leq c'$.

(c) Use (b) to prove (a).

Comments. Source: M. Li and P. Vitányi, *SIAM J. Comput.*, 24:2(1995), 398–410. In this paper, an incompressibility criterion more general than Item (b) is given for separating DCFL from CFL. See also [S. Yu, *Inform. Process. Lett.*, 31(1989), 47–51] and [M.A. Harrison, *Introduction to Formal Language Theory*, Addison-Wesley, 1978] for basics of formal language theory and traditional approaches to this problem such as iteration lemmas.

6.8.7. [35] We have characterized the regular languages using Kolmogorov complexity. It is immediately obvious how to characterize recursive languages in terms of Kolmogorov complexity. If $L \subseteq \Sigma^*$ and $\Sigma^* = \{v_1, v_2, \ldots\}$ is an effective enumeration, then we define the characteristic sequence $\chi = \chi_1\chi_2\ldots$ of L by $\chi_i = 1$ if $v_i \in L$ and $\chi_i = 0$ otherwise. A language L is recursive if χ is a recursive sequence.

(a) If a set $L \subseteq \Sigma^*$ is recursive then there exists a constant c_L (depending only on L) such that for all n, we have $C(\chi_{1:n}|n) < c_L$.

(b) If L is recursively enumerable, then there is a constant c_L such that for all n, we have $C(\chi_{1:n}|n) \leq \log n + c_L$.

(c) There exists a recursively enumerable set L such that $C(\chi_{1:n}) > \log n$, for all n.

Comments. Item (a) is straightforward. Its converse is hard: see the text preceding the proof of Claim 6.8.1 on page 493. This converse is given by Item (e) of Exercise 2.3.4 on page 131. Items (b) and (c) are Barzdins's lemma, Theorem 2.7.2, restated. It quantitatively characterizes all recursively enumerable languages in terms of Kolmogorov complexity. Hint for Item (c): Exercise 2.3.4. With L as in Item (c), $\Sigma^* - L$ also satisfies Item (b), so Item (b) cannot be extended to a Kolmogorov complexity characterization of recursively enumerable sets.

6.8.8. [23] Assume the terminology in Exercise 6.8.7. Consider χ defined in the proof for Item (ii) of Barzdins's lemma, Theorem 2.7.2. Essentially, $\chi_i = 1$ if the ith program started on the ith input string halts and outputs 0, and $\chi_i = 0$ otherwise. Let A be the language with χ as its characteristic sequence.

(a) Show that A is a recursively enumerable set and its characteristic sequence satisfies $C(\chi_{1:n}) \geq \log n$, for all n.

(b) Let χ be as in Item (a). Define a sequence h by

$$h = \chi_1 0^2 \chi_2 0^{2^2} \ldots \chi_i 0^{2^i} \chi_{i+1} \ldots \ .$$

Prove that $C(h_{1:n}) = O(C(n)) + \Theta(\log\log n)$. Therefore, if h is the characteristic sequence of a set B, then B is not recursive, but more sparsely nonrecursive, as is A.

Comments. Item (a) follows from the proof of Barzdins's lemma, Theorem 2.7.2. Source: J.M. Barzdins, *Soviet. Math. Dokl.*, 9(1968), 1251–1254; D.W. Loveland, *Proc. 1st ACM Symp. Theory Comput.*, 1969, pp. 61–66.

6.8.9. [19] The probability that the universal prefix machine U halts on self-delimiting binary input p, randomly supplied by tosses of a fair coin, is Ω ($0 < \Omega < 1$). Let $v_1, v_2, \ldots$ be an effective enumeration without

repetitions of Σ^*. Define $L \subseteq \Sigma^*$ such that $v_i \in L$ iff $\Omega_i = 1$. Section 3.6.2 implies that $K(\Omega_{1:n}) \geq n$ for all but finitely many n. Show that L and its complement are not recursively enumerable.

Comments. It can be proved that $L \in \Delta_2^0 - (\Sigma_1^0 \bigcup \Pi_1^0)$, in the arithmetic hierarchy. See Section 3.6.2, page 225, and Exercise 1.7.21, page 46.

6.9 Online CFL Recognition

The incompressibility proof below demonstrates a lower bound on the time for language recognition by a multitape Turing machine, as shown in Figure 6.4. A multitape Turing machine recognizes a language *online* if before reading each successive input symbol, it decides whether the partial input string scanned so far belongs to the language.

A context-free language is *linear* if it is generated by a linear context-free grammar in which no production rule contains more than one nonterminal symbol on the right-hand side. The known upper bound on the time required for online recognition of a linear context-free language by a multitape Turing machine is $O(n^2)$, even if only one work tape is available. We prove a corresponding $\Omega(n^2/\log n)$ lower bound. Let x_i^R denote x_i written in reverse, and let $y, x_1, \ldots, x_k \in \{0,1\}^*$. Define a language L as

$$L = \{y\#x_1@x_2@\cdots@x_k : \text{ for some } i,\ x_i^R \text{ is a substring of } y\}.$$

The language L is linear context-free, since it is generated by the following linear grammar, with starting symbol S: $S \rightarrow S_1|S@|S0|S1$; $S_1 \rightarrow 0S_1|1S_1|S_2$; $S_2 \rightarrow 0S_20|1S_21|S_3@$; $S_3 \rightarrow S_30|S_31|S_3@|S_4\#$; $S_4 \rightarrow 0S_4|1S_4|\epsilon$.

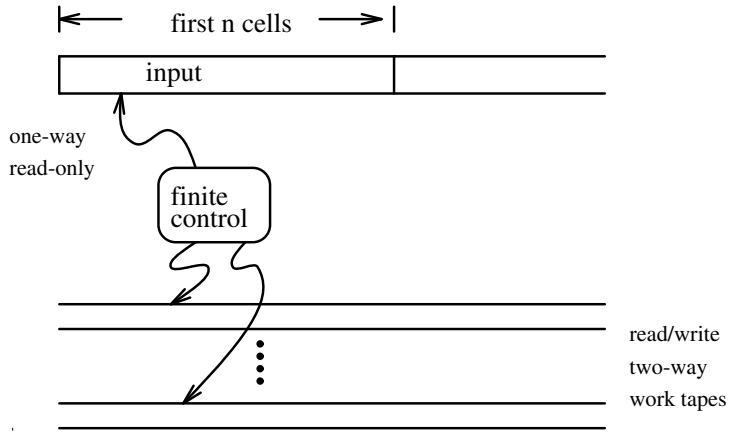


FIGURE 6.4. Multitape Turing machine

Theorem 6.9.1 *A multitape Turing machine that online recognizes L requires time $\Omega(n^2/\log n)$.*

Proof. Assume that a multitape Turing machine T accepts L online in $o(n^2/\log n)$ steps. Choose y such that $C(y) \geq l(y) = n$. Using y, we will construct a hard input of length $O(n)$ for T. The idea of the proof is to construct an input

$$y\#x_1@\cdots@x_k@$$

such that no x_l is a reverse of a substring of y and yet each x_l is hard enough to make T use ϵn steps, for some $\epsilon > 0$ not depending on n. If $k = \Omega(n/\log n)$, then T will be forced to take $\Omega(n^2/\log n)$ steps. Our task is to prove the existence of such x_l's. We need two lemmas:

Lemma 6.9.1 *Let $n = l(x)$, and let p be a program described in the proof below. Assume that $C(x|n,p) \geq n$. Then no substring of length longer than $2\log n$ occurs, possibly overlapping, in x more than once.*

Proof. Let $x = uvw$, with v of length greater than $2\log n$ occurring exactly twice in uv. Let this discussion be formulated in terms of a program p that reconstructs x from the description below using the value of n (given for free). To describe x, given p and n, we need only to concatenate the following information:

- the locations of the start bits of the two occurrences of v in uv using $\log n(n-1)$ bits;
- the literal word uw, using exactly $l(uw)$ bits.

Altogether, this description requires $n - 2\log n + \log n(n-1)$ bits. Since $C(x|n,p)$ is the shortest such description, we have $C(x|n,p) < n$. This contradicts the assumption in the lemma. □

Lemma 6.9.2 *If a string has no repetition of length m, then it is uniquely determined by the set of its substrings of length $m+1$; that is, it is the unique string with no repetition of length m and with precisely this set of substrings of length $m+1$.*

Proof. Let S be the set of substrings of x of length $m+1$. Let $a, b \in \{0,1\}$, and $u, v, w \in \{0,1\}^*$. The prefix of x of length $m+1$ corresponds uniquely to the $ua \in S$ such that for no b is bu in S. For any prefix vw of x with $l(w) = m$, there is a unique b such that $wb \in S$. Hence, the unique prefix of length $l(vw)+1$ is vwb. The lemma follows by induction. □

We continue to prove the theorem. By Lemmas 6.9.1 and 6.9.2 we let $m = 3 \log n$, so that y is uniquely determined by its set of substrings of length m. For $i = 1, \ldots, k$, assume inductively that $x_1, \ldots, x_{i-1}$, each of length m, have been chosen so that the input prefix $y\#x_1@\cdots@x_{i-1}@$ does not yet belong to L, and T spends at least ϵn steps on each $x_j@$ block for $j < i$.

We claim that for each i $(1 \leq i \leq k)$, there is an x_i of length m that is not a reverse substring of y such that appending $x_i@$ to the input requires at least $t = \epsilon n$ additional steps by T, where $\epsilon > 0$ does not depend on n. Setting $k = n/m$, this proves the theorem.

Assume, by way of contradiction, that this is not the case. We devise a short description of all the length-m substrings of y, and hence, by Lemma 6.9.2, of y. Simulate T with input $y\#x_1@\cdots@x_{i-1}@$, and record the following information at the time t_0 when T reads the last @ sign:

- this discussion;
- the work tape contents within distance $t = \epsilon n$ of the tape heads;
- the specification of T, length n, current state of T, and locations of T's heads.

With this information, one can easily search for all x_i's such that $l(x_i) = m$ and ${x_i}^R$ is a substring of y as follows: Simulate T from time t_0, using the above information and with input suffix $x_i@$, for t steps. By assumption, if T accepts or uses more than t steps, then x_i is a reverse substring of y. If ϵ is sufficiently small and n is large, then all the above information adds up to fewer than n bits, a contradiction. □

Exercises

6.9.1. [28] A *k-head deterministic finite automaton*, abbreviated k-DFA, is similar to a deterministic finite automaton except that it has k, rather than one, one-way read-only input heads. In each step, depending on the current state and the k symbols read by the k heads, the machine changes its state and moves some of its heads one step to the right. It stops when all heads reach the end of input, and at this time it accepts the input if it is in a final state. Use the incompressibility method to show that for each $k \geq 1$ there is a language L that is accepted by a $(k+1)$-DFA but is not accepted by any k-DFA.

Comments. Hint: use the language

$$L_b = \{w_1\# \cdots \#w_b@w_b\# \cdots \#w_1 : w_i \in \{0,1\}^*\}$$

with $b = \binom{k}{2} + 1$. Intuitively, when w_i's are all random, for each pair of w_i's we must have two heads matching them concurrently. But a k-DFA can match only $\binom{k}{2}$ pairs. This result was first conjectured in 1965 by A. Rosenberg [*IBM J. Res. Develop.,* 10(1966), 388–394]. The case $k = 2$ was settled by H. Sudborough [*Inform. Contr.*, 30(1976), 1–20] and O.H. Ibarra and C.E. Kim [*Acta Informatica*, 4(1975), 193–200]. The case $k > 2$ took a decade to settle [A.C.C. Yao and R. Rivest, *J. Assoc. Comp. Mach.*, 25(1978), 337–340; C.G. Nelson, *Technical Report*, 14-76(1976), Harvard University]. These proofs use more complicated counting arguments. The proof by Kolmogorov complexity is folklore; it can be found in Section 6.4.3 of the first edition of this book.

6.9.2. [40/O45] Refer to Exercise 6.9.1 for the definition of k-DFAs, prove the following. Let $L = \{x\#y : x \text{ is a substring of } y\}$.

(a) No 2-DFA can do string-matching, that is, no 2-DFA accepts L.

(b) No 3-DFA accepts L.

(c) No k-DFA accepts L, for any integer k.

(d) [Open] No k-DFA with sensing heads accepts L, for any k, where the term *sensing* means that the heads can detect each other when they meet.

Comments. The results in this exercise were motivated by a conjecture of Z. Galil and J. Seiferas [*J. Comput. System Sci.*, 26:3(1983), 280–294] that no k-DFA can do string-matching, for any k. Galil and Seiferas proved that 6-head two-way DFA can do string-matching in linear time. Item (a) was first proved in [M. Li and Y. Yesha, *Inform. Process. Lett.*, 22(1986), 231–235]; item (b) in [M. Geréb-Graus and M. Li, *J. Comput. System Sci.*, 48(1994), 1–8]. Both of these papers provided useful tools for the final solution, item (c), by T. Jiang and M. Li [*Proc. 25th ACM Symp. Theory Comput.*, 1993, pp. 62–70].

6.9.3. [38] A k-head PDA (k-PDA) is similar to a pushdown automaton except that it has k input heads. Prove that $k + 1$ heads are better than k heads for PDAs. That is, prove that there is a language that is accepted by a $(k + 1)$-PDA but not by any k-PDA.

Comments. Conjectured by M.A. Harrison and O.H. Ibarra [*Inform. Contr.*, 13(1968), 433–470] in analogy to Exercise 6.9.1. Partial solutions were obtained by S. Miyano [*Acta Informatica*, 17(1982), 63–67; *J. Comput. System Sci.*, 27(1983), 116–124]; and M. Chrobak [*Theoret. Comput. Sci.*, 48(1986), 153–181]. The complete solution, using incompressibility, is in [M. Chrobak and M. Li, *J. Comput. System Sci.*, 37:2(1988), 144–155].

6.9.4. [35] (a) A k-pass DFA is just like a usual DFA except that the input head reads the input k times, from the first symbol to the last

symbol, moving right only during each pass. Use incompressibility to show that a k-pass DFA is exponentially more succinct than a $(k-1)$-pass DFA. In other words, for each k, there is a language L_k such that L_k can be accepted by a k-pass DFA with $O(kn)$ states, but the smallest $(k-1)$-pass DFA accepting L_k requires $\Omega(2^n)$ states.

(b) A sweeping two-way DFA is again just like a usual finite automaton except that its input head may move in two directions with the restriction that it can reverse direction only at the two ends of the input. If a sweeping two-way DFA makes $k-1$ reversals during its computation, we call it a k-sweep two-way DFA. Show that there is a language R_k that can be accepted by a $2k$-sweep two-way DFA with $p(k,n)$ states for some polynomial p, but the smallest $(2k-1)$-sweep two-way DFA accepting R_k requires an exponential number of states in terms of k and n.

Comments. W.J. Sakoda and M. Sipser studied sweeping automata in [*Proc. 10th ACM Symp. Theory Comput.*, 1978, pp. 275–286]. They called the k-pass DFA by the name 'k-component series FA.' Source: T. Jiang, e-mail, 1992.

6.9.5. [32] Consider a singly linked list L of n items, where the ith item has a pointer pointing to the $(i+1)$st item, with the last pointer being nil. Let $\epsilon > 0$, prove:

(a) Every sequence of $t(n) \geq n$ steps of going backward on L can be done in $O(t(n)n^\epsilon)$ steps, without modifying L or using extra memory other than $O(1)$ extra pointers or counters.

(b) Any program using $O(t(n)n^\epsilon)$ steps to go back $t(n)$ steps on L requires at least $k-1$ pointers.

Comments. Hint: Item (a) does not need Kolmogorov complexity. You can use $O(n)$ initial start time. For Item (b), if a region passed by does not get visited by a pointer during this process, then it can be compressed. Source: A.M. Ben-Amram and H. Petersen, *Proc. 31st ACM Symp. Theory Comput.*, pp. 780–786, 1999.

6.9.6. [31] Let I be an index structure supporting text search in $O(l(P))$-bit probes to find pattern P in text T as a substring.

(a) If each query requires the location of P, then the size of I is $\Omega(l(T))$.

(b) Even if each query asks only whether a substring P is in T, the size of I is still $\Omega(l(T))$.

Comments. Item (a) is by E.D. Demaine and A. Lopez-Ortiz, *J. Alg.*, 48:1(2003), 2–15. Item (b) is due to M. Li and P.M.B. Vitányi [Unpublished, 2006]. Hint for (b), use Lemma 6.9.2. An upper bound of Item (b) is $O(w \log N)$ bits for w query words, by M.L. Fredman, J. Komlós, and E. Szemerédi [*J. Assoc. Comp. Mach.* 31:3(1984), 538–544].

6.10 Turing Machine Time Complexity

The incompressibility method has been quite successfully applied in solving open problems. We give one such example proving a lower bound on the time required to simulate a multitape Turing machine by a 1-(work)tape Turing machine (with a one-way input tape).

A *k-tape Turing machine with one-way input*, as shown in Figure 6.4, has k work tapes and a one-way read-only input tape that contains the input. Initially, the input is written on the leftmost tape segment, one symbol per tape square, and the input head scans the leftmost input symbol. The end of the input is delimited by a distinguished end marker. Observe that this model with $k = 1$ is far more powerful than the single-tape Turing machine model of Figure 6.1, page 443, where the single tape serves both as input tape and work tape. For instance, a Turing machine with one work tape apart from the input tape can recognize $L = \{w\#w^R : w \in \{0,1\}^*\}$ in real time, in contrast to the $\Omega(n^2)$ lower bound required in Section 6.1.1 of Section 6.1. The additional marker # allows the positive result and does not change the lower bound result.

In the literature, an *online* model is also used. In this model, the input tape is one-way, and on every prefix of the input the Turing machine writes the output, accepting or rejecting the prefix, on a write-only one-way output tape. Proving lower bounds is easier with the online model than with the one-way input model we use in this section. We use the latter model and thus prove stronger results.

A basic question in Turing machine complexity is whether additional work tapes add power. It is known that one tape can online simulate n steps of k tapes in $O(n^2)$ steps. It has been a two-decade-long open question whether the known simulation is tight. Kolmogorov complexity has helped to settle this question by an $\Omega(n^2)$ lower bound; see Exercise 6.10.2.

The tight $\Omega(n^2)$ lower bound requires a lengthy proof. We provide a weaker form of the result whose simpler proof yet captures the central ideas of the full version.

Theorem 6.10.1 *It requires $\Omega(n^{3/2}/\log n)$ time to deterministically simulate a linear-time 2-tape Turing machine with one-way input by a 1-tape Turing machine with one-way input.*

Proof. We first prove a useful lemma. Let T be a 1-tape Turing machine with input tape head h_1 and work tape head h_2. Let s be a segment of T's input, and R a tape segment on its work tape. We say that T *maps s into R* if h_2 never leaves tape segment R while h_1 is reading s, and T maps s *onto* R if h_2 traverses the *entire* tape segment R while h_1 reads s. The *c.s.* at position p of the work tape is a sequence of pairs of form

$$(\text{state of } T, \text{position of } h_1),$$

which records the status of T when h_2 enters p each time.

The following lemma states that a tape segment bordered by short $c.s.$'s cannot receive a lot of information without losing some. We assume the following situation: Let the input string start with $x\#$, where $x = x_1x_2\ldots x_k$ with $l(x_i) = l(x)/k$ for all i. Let R be a segment of T's storage tape such that T maps all blocks in $S = \{x_{i_1}, \ldots, x_{i_l}\}$ into tape segment R, where $S \subseteq \{x_i : 1 \leq i \leq k\}$.

Lemma 6.10.1 **(Jamming lemma)** *The contents of the storage tape of T at the time when h_1 moves to the # marker can be reconstructed using only the sequence of blocks $\bar{S} = \{x_i : 1 \leq i \leq k\} - S$, the final contents of R, the two final c.s.'s on the left and right boundaries of R, a description of T, and a description of this discussion.*

Roughly speaking, if the number of missing bits $\sum_{j=1}^{l} l(x_{i_j})$ exceeds the number of added description bits (those for R and the two crossing sequences around R), then the jamming lemma implies that either $x = x_1\ldots x_k$ is not incompressible or some information about x has been lost.

Proof. Let the two positions at the left boundary and the right boundary of R be l_R and r_R, respectively. Subdivide the input tape into c-sized slots with $c = l(x)/k$. Put the blocks x_j of $\bar{S}$ in their correct positions on the input tape in the following way: In the simulation, h_1 reads the input from left to right without backing up. We have a list $\bar{S}$ of c-sized blocks available. We put the consecutive blocks of $\bar{S}$ on the c-sized slots on the input tape such that the slots, which are traversed by h_1 with h_2 all the time positioned on R, are left empty. This can be easily determined from the left and right crossing sequences of R.

Simulate T with h_2 staying to the left of R using the $c.s.$ at l_R to construct the work tape contents to the left of R. Also simulate T with h_2 staying to the right of R using the $c.s.$ at r_R to construct the work tape contents to the right of R. Such a simulation is standard.

We now have obtained the contents of T's work tape at the end of processing $x\#$, apart from the contents of R. The final contents of R are given and put in position. Together, we now have T's work tape contents at the time when h_1 reaches #.

Notice that although there are many unknown x_i's (in S), they are never read, since h_1 skips over them because h_2 never goes into R. □

To prove the theorem we use the witness language L defined by

$$L = \{x_1@x_2@\cdots@x_k\#y_1@\cdots@y_l\#0^i1^j : x_i = y_j\}. \tag{6.18}$$

Clearly, L can be accepted in linear time by a 2-tape machine. Assume, by way of contradiction, that a deterministic 1-tape machine T accepts

L in $T(n) < c^{-5}n^{3/2}/\log n$ time, for some fixed new constant c and n large enough. We derive a contradiction by showing that then some incompressible string must have too short a description.

Assume, without loss of generality, that T writes only 0's and 1's in its work squares and that $l(T) = O(1)$ is the number of states of T. Fix the new constant c and take the word length n as large as needed to derive the desired contradictions below and such that the formulas in the sequel are meaningful.

First, choose an incompressible string $x \in \{0,1\}^*$ of length $l(x) = n$ and $C(x) \geq n$. Let x consist of the concatenation of $k = \sqrt{n}$ substrings, $x_1, x_2, \ldots, x_k$, each substring $\sqrt{n}$ bits long. Let

$$x_1@x_2@\cdots@x_k\#$$

be the initial input segment on T's input tape. Let time $t_\#$ be the step at which h_1 reads #. If more than $k/2$ of the x_i's are mapped *onto* a contiguous tape segment of size at least n/c^3, then T requires $\Omega(n^{3/2}/\log n)$ time, which is a contradiction. Therefore, there is a set S consisting of $k/2$ blocks x_i such that for every $x_i \in S$ there is a tape segment of $\leq n/c^3$ contiguous tape squares into which x_i is mapped. In the remainder of the proof we restrict attention to the x_i's in this set S. Order the elements of S according to the order of the left boundaries of the tape segments *into* which they are mapped. Let x_m be the median.

The idea of the remainder of the proof is as follows: Intuitively, the only thing T can do before input head h_1 crosses # is somehow copy the x_i's onto its work tape, and afterward copy the y_j's onto the work tape. There must be a pair of these x_i and y_j that are separated by $\Omega(n)$ distance, since all these blocks together must occupy $\Omega(n)$ space. At this time, head h_1 still has to read the 0^i1^j part of the tape. Hence, we can force T to check whether $x_i = y_j$, which means that it has to spend about $\Omega(n^{3/2}/\log n)$ time. To convert this intuition into a rigorous proof we distinguish two cases:

In the first case we assume that many x_i's in S are mapped (jammed) into a small tape segment R. That is, when h_1 (the input tape head) is reading them, h_2 (the work tape head) is always in this small tape segment R. We show that then, contrary to assumption, x can be compressed (by the jamming lemma). Intuitively, some information must have been lost.

In the second case, we assume there is no such jammed tape segment and that the records of the $x_i \in S$ are spread evenly over the work tape. In that case, we will arrange the y_j's so that there exists a pair (x_i, y_j) such that $x_i = y_j$ and x_i and y_j are mapped into tape segments that are far apart, at distance $\Omega(n)$. Then we complete T's input with final

index 0^i1^j so as to force T to match x_i against y_j. As in Section 6.1.1, page 442, either T spends too much time or we can compress x again, yielding a second contradiction and proving, for large enough n,

$$T(n) \geq \frac{n^{3/2}}{c^5 \log n}.$$

Case 1 (Jammed) Assume there are k/c blocks $x_i \in S$ and a fixed tape segment R of length n/c^2 on the work tape such that T maps all of these x_i's into R. Let S' be the set of such blocks.

We will construct a short program that prints x. Consider the two tape segments of length $l(R)$ to the left and to the right of R on the work tape. Call them R_l and R_r, respectively. Choose positions p_l in R_l and p_r in R_r with the shortest *c.s.*'s in their respective tape segments. Both *c.s.*'s must be shorter than $\sqrt{n}/(c^2 \log n)$. Namely, if the shortest *c.s.* in either tape segment is at least $\sqrt{n}/(c^2 \log n)$ long, then T uses at least

$$\frac{\sqrt{n}}{c^2 \log n} \cdot \frac{n}{c^2}$$

steps, and there is nothing to prove. Let tape segment $R_l{}'$ ($R_r{}'$) be the portion of R_l (R_r) right (left) of p_l (p_r).

Now, using the description of

- this discussion (including the text of the program below) and simulator T in $O(1)$ bits;
- the values of n, k, c, and the positions of p_l, p_r in $O(\log n)$ bits;
- the literal concatenated list $\{x_1, \ldots, x_k\} - S'$, using $n - n/c$ bits;
- the state of T and the position of h_2 at time $t_\#$ in $O(\log n)$ bits;
- the two *c.s.*'s at positions p_r and p_l at time $t_\#$ in at most $2\sqrt{n}(l(T)+O(\log n))$ bits; and
- the contents at time $t_\#$ of tape segment $R_l{}'RR_r{}'$ in at most $3n/c^2 + O(\log n)$ bits;

we can construct a program to check whether a candidate string y equals x by running T as follows:

Check whether $l(y) = l(x)$. By the jamming lemma (using the above information related to T's processing of the initial input segment $x_1@\cdots@x_k\#$), reconstruct the contents of T's work tape at time $t_\#$, the time h_1 gets to the first # sign. Divide y into k equal pieces and form $y_1@\cdots@y_k$. Run T, initialized in the appropriate state, head positions, and work tape

contents (at time $t_{\#}$), as the starting configuration, on each input suffix of the form

$$y_1@\cdots@y_k\#0^i1^i.$$

By definition of L, the machine T can accept for all i iff $y = x$.

This description of x requires not more than

$$n - \frac{n}{c} + \frac{3n}{c^2} + O(\sqrt{n}\log n) + O(\log n) \leq \gamma n$$

bits, for some constant $0 < \gamma < 1$ and large enough c and n. However, this contradicts the incompressibility of x ($C(x) \geq n$).

Case 2 (Not Jammed) Assume that for each fixed tape segment R, with $l(R) = n/c^2$, there are at most k/c blocks $x_i \in S$ mapped into R.

Fix a tape segment of length n/c^2 into which the median x_m is mapped. Call this segment R_m. Then, at most k/c strings x_i in set S are mapped into R_m. Therefore, for large enough c (and $c > 3$), at least $k/6$ of the x_i's in S are mapped into the tape right of R_m. Let the set of those x_i's be $S_r = \{x_{i_1}, \ldots, x_{i_{k/6}}\} \subset S$. Similarly, let $S_l = \{x_{j_1}, \ldots, x_{j_{k/6}}\} \subset S$ consist of $k/6$ strings x_i that are mapped into the tape left of R_m. Without loss of generality, assume $i_1 < i_2 < \cdots < i_{k/6}$, and $j_1 < j_2 < \cdots < j_{k/6}$.

Set $y_1 = x_{i_1}$, $y_2 = x_{j_1}$, $y_3 = x_{i_2}$, $y_4 = x_{j_2}$, and so forth. In general, for all integers s, $1 \leq s \leq k/6$,

$$y_{2s} = x_{j_s} \quad \text{and} \quad y_{2s-1} = x_{i_s}. \tag{6.19}$$

Using this relationship, we now define an input prefix for T to be

$$x_1@\cdots@x_k\#y_1@\cdots@y_{k/3}\#. \tag{6.20}$$

There exists a pair y_{2i-1}, y_{2i} that is mapped into a segment of size less than $n/(4c^2)$. Otherwise, T uses at least

$$\frac{k}{6} \cdot \frac{n}{4c^2} = \frac{n^{3/2}}{24c^2}$$

steps, and there is nothing to prove. Now this pair y_{2i-1}, y_{2i} is mapped into a segment with distance at least n/c^3 either to x_{i_s} or to x_{j_s}. Without loss of generality, let y_{2s-1}, y_{2s} be mapped n/c^3 away from x_{i_s}. So y_{2s-1} and x_{i_s} are separated by a region R of size n/c^3. Attach suffix $0^{i_s}1^{2s-1}$ to the initial input segment of Equation 6.20 to complete the input to T to

$$x_1@\cdots@x_k\#y_1@\cdots@y_{k/3}\#1^{i_s}1^{2s-1}. \tag{6.21}$$

So at the time when T reads the second # sign, x_{i_s} is mapped into the tape left of R, and y_{2s-1}, which is equal to x_{i_s}, is mapped into the tape right of R.

Determine position p in R that has the shortest *c.s.* of T's computation on the input of Equation 6.21. If this *c.s.* is longer than $\sqrt{n}/(c^2 \log n)$, then T uses at least

$$\frac{n}{c^3} \cdot \frac{\sqrt{n}}{c^2 \log n}$$

steps, and there is nothing to prove. Therefore, assume that the shortest *c.s.* has length at most $\sqrt{n}/(c^2 \log n)$. Then again we can construct a short program P to accept *only* x by a cut-and-paste argument, and show that it yields too short a description of x. Using the description of

- this discussion (including the text of the program P below) and simulator T in $O(1)$ bits;
- the values of n, k, c, and the position p in $O(\log n)$ bits;
- $n - \sqrt{n}$ bits for $S - \{x_{i_s}\}$;
- $O(\log n)$ bits for the index i_s of x_{i_s} to place it correctly on the input tape; and
- $\leq \sqrt{n}/c$ bits to describe the *c.s.* of length $\sqrt{n}/(c^2 \log n)$ at p (assuming $c \gg l(T)$);

we can construct a program to reconstruct x as follows:

Construct the input of Equation 6.21 on T's input tape with the two blocks x_{i_s} and y_{2s-1} filled with blanks. Now we search for x_{i_s} as follows. For each candidate z with $l(z) = \sqrt{n}$ put z in y_{2s-1}'s position and do the following simulation:

Using the *c.s.* at point p, we run T such that h_2 always stays at the right of p (y_{2s-1}'s side). Whenever h_2 encounters p, we check whether the current status matches the corresponding ID in the *c.s.* If it does, then we use the next ID of the *c.s.* to continue. If in the course of this simulation process T rejects or there is a mismatch (that is, when h_2 gets to p, machine T is not in the same state or h_1's position is not as indicated in the *c.s.*), then $z \neq x_{i_s}$. If the crossing sequence at p of T's computation for candidate z matches the prescribed *c.s.*, then we know that T would accept the input of Equation 6.21 with y_{2s-1} replaced by z. Therefore, $z = x_{i_s}$.

The description of x requires not more than

$$n - \sqrt{n} + \frac{1}{c}\sqrt{n} + O(\log n) \leq n - \gamma\sqrt{n}$$

bits for some positive $\gamma > 0$ and large enough c and n. This contradicts the incompressibility of x ($C(x) \geq n$) again.

Case 1 and Case 2 complete the proof that $T(n) \geq c^{-5} n^{3/2} / \log n$. □

Exercises

6.10.1. [33] Consider the 1-tape Turing machine as in Section 6.1.1, page 442. Let the input be $n/\log n$ integers each of size $O(\log n)$, separated by # signs. The element-distinctness problem is to decide whether all these integers are distinct. Prove that the element-distinctness problem requires $\Omega(n^2/\log n)$ time on such a 1-tape Turing machine.

Comments. A similar bound also holds for 1-tape nondeterministic Turing machines. Source: [A. López-Ortiz, *Inform. Process. Lett.*, 51:6(1994), 311–314].

6.10.2. [42] Extend the proof of Theorem 6.10.1 to prove the following: Simulating a linear-time 2-tape deterministic Turing machine by a 1-tape deterministic Turing machine requires $\Omega(n^2)$ time. (Both machines of the one-way input model.)

Comments. Hint: set the block size for x_i to be a large constant, and modify the language to one that requires comparison of $\Omega(n)$ pairs of x_i's and y_j's. The lower bound is optimal, since it meets the $O(n^2)$ upper bound of [J. Hartmanis and R. Stearns, *Trans. Amer. Math. Soc.*, 117(1969), 285–306]. Source: W. Maass, *Trans. Amer. Math. Soc.*, 292(1985), 675–693; M. Li and P.M.B. Vitányi, *Inform. Comput.*, 78(1988), 56–85.

6.10.3. [38] A k-pushdown store machine is similar to a k-tape Turing machine with one-way input except that the k work tapes are replaced by k pushdown stores. Prove: simulating a linear-time 2-pushdown store deterministic machine with one-way input by a 1-tape nondeterministic Turing machine with one-way input requires $\Omega(n^{3/2}/\sqrt{\log n})$ time.

Comments. Source: M. Li and P.M.B. Vitányi, *Inform. Comput.*, 78(1988), 56–85. This bound is optimal, since it is known that simulating a linear-time 2-pushdown store deterministic machine with one-way input by a 1-tape nondeterministic Turing machine with one-way input can be done in $O(n^{3/2}\sqrt{\log n})$ time [M. Li, *J. Comput. System Sci.*, 7:1(1988), 101–116].

6.10.4. [44] Show that simulating a linear-time 2-tape deterministic Turing machine with one-way input by a 1-tape nondeterministic Turing machine with one-way input requires $\Omega(n^2/((\log n)^2 \log\log n))$ time.

Comments. Hint: let S be a sequence of numbers from $\{0, \ldots, k-1\}$, where $k = 2^l$ for some l. Assume that each number $b \in \{0, \ldots, k-1\}$ is

somewhere in S adjacent to the number $2b \pmod k$ and $2b+1 \pmod k$. Then for every partition of $\{0, \ldots, k-1\}$ into two sets G and R such that $d(G), d(R) > k/4$ there are at least $k/(c \log k)$ (for some fixed c) elements of G that occur somewhere in S adjacent to a number from R. Subsequently prove the lower bound using the language $L \subseteq \{0,1\}^*$ defined as follows. Let $u = u_1 \ldots u_k$, where the u_i's are of equal length. Form $uu = u_1 \ldots u_{2k}$ with $u_{k+i} = u_i$. Then inserting u_i between u_{2i-1} and u_{2i} for $1 \leq i \leq k$ results in a member in L. These are the only members of L. Source: W. Maass, *Trans. Amer. Math. Soc.*, 292(1985), 675–693. The language L defined in this hint will not allow us to obtain an $\Omega(n^2)$ lower bound. Define a graph $G = \langle Z_n, E_{ab} \rangle$, where $Z_n = \{0, 1, \ldots, n-1\}$, $E_{ab} = \{(i,j) : j \equiv (ai + b) \bmod n \text{ for } i \in Z_n\}$, and a and b are fixed positive integers. Then G has a separator, a set of nodes whose removal separates G into two disconnected, roughly equal-sized components of size $O(n/\sqrt{\log_a n})$. Using such a separator, L can be accepted in subquadratic time by a 1-tape online deterministic machine [M. Li, *J. Comput. System Sci.*, 7:1(1988), 101–116].

6.10.5. [46] Prove that simulating a linear-time 2-tape deterministic Turing machine with one-way input by a 1-tape nondeterministic Turing machine with one-way input requires $\Omega(n^2/\log^{(k)} n)$ time, for any k, where $\log^{(k)} = \log \log \cdots \log$ is the k-fold iterated logarithm. This improves the result in Exercise 6.10.4.

Comments. Source: Z. Galil, R. Kannan, and E. Szemerédi [*J. Comput. System Sci.*, 38(1989), 134–149; *Combinatorica*, 9(1989), 9–19].

6.10.6. [O47] Does simulating a linear-time 2-tape deterministic Turing machine with one-way input by a 1-tape nondeterministic Turing machine with one-way input require $\Omega(n^2)$ time?

6.10.7. [46] A k-queue machine is similar to a k-tape Turing machine with one-way input except with the k work tapes replaced by k work queues. A queue is a first-in last-out (FIFO) device. Prove (with one-way input understood):

(a) Simulating a linear-time 1-queue machine by a 1-tape Turing machine requires $\Omega(n^2)$ time.

(b) Simulating a linear-time 1-queue machine by a 1-tape nondeterministic Turing machine requires $\Omega(n^{4/3}/\log^{2/3} n)$ time.

(c) Simulating a linear-time 1-pushdown store machine (which accepts precisely CFLs) by a 1-queue machine, deterministically or nondeterministically, requires $\Omega(n^{4/3}/\log n)$ time.

Comments. Items (a) and (b) are from [M. Li and P.M.B. Vitányi, *Inform. Comput.,* 78(1988), 56–85]; Item (c) is from [M. Li, L. Longpré, and P.M.B. Vitányi, *SIAM J. Comput.*, 21:4(1992), 697–712]. The bound

in Item (a) is tight. The bound in Item (b) is not tight; the best upper bound is $O(n^{3/2}\sqrt{\log n})$, in [M. Li, *J. Comput. System Sci.*, 7:1(1988), 101–116]. The bound in Item (c) is not tight; the upper bound is known to be $O(n^2)$ (also to simulate a 2-pushdown store machine).

6.10.8. [43] Use the terminology of Exercise 6.10.7, with one-way input understood.

(a) Show that simulating a linear-time deterministic 2-queue machine by a deterministic 1-queue machine takes $\Omega(n^2)$ time.

(b) Show that simulating a linear-time deterministic 2-queue machine by a nondeterministic 1-queue machine takes $\Omega(n^2/(\log^2 n \log\log n))$ time.

(c) Show that simulating a linear-time deterministic 2-tape Turing machine by nondeterministic 1-queue machine takes $\Omega(n^2/\log^2 n \log\log n)$ time.

Comments. The upper bounds in all cases are $O(n^2)$ time. Source: M. Li, L. Longpré, and P.M.B. Vitányi, *SIAM J. Comput.*, 21:4(1992), 697–712. For additional results on simulating $(k+1)$-queue machines and 2-tape or multitape machines by k-queue machines see [M. Hühne, *Theoret. Comput. Sci.*, 113:1(1993), 75–91].

6.10.9. [38] Consider the stronger offline deterministic Turing machine model with a two-way read-only input tape. Given an $l \times l$ matrix A, with $l = \sqrt{n/\log n}$ and element size $O(\log n)$, arranged in row-major order on the two-way (1-dimensional) input tape,

(a) Show that one can transpose A (that is, write A^T on a work tape in row-major form) in $O(n \log n)$ time on such a Turing machine with two work tapes.

(b) Show that it requires $\Omega(n^{3/2}/\sqrt{\log n})$ time on such a Turing machine with one work tape to transpose A.

(c) From Items (a) and (b), obtain a lower bound on simulating two work tapes by one work tape for the above machines.

Comments. Source: M. Dietzfelbinger, W. Maass, and G. Schnitger, *Theoret. Comput. Sci.*, 82:1(1991), 113–129.

6.10.10. [37] We analyze the speed of copying strings for Turing machines with a two-way input tape and one or more work tapes.

(a) Show that such a Turing machine with one work tape can copy a string of length s, initially positioned on the work tape, to a work tape segment that is d tape cells removed from the original position in $O(d + sd/\log\min(n, d))$ steps. Here n denotes the length of the input.

(b) Show (by the incompressibility method) that the upper bound in Item (a) is optimal. For $d = \Omega(\log n)$, such a Turing machine with one

work tape requires $\Omega(sd/\log\min(n,d))$ steps to copy a string of length s across d tape cells.

(c) Use Item (a) to show that such Turing machines can simulate $f(n)$-time bounded multitape Turing machines in $O(f(n)^2/\log n)$ steps. This is faster by a multiplicative factor $\log n$ than the straightforward simulations.

Comments. Source: M. Dietzfelbinger, *Inform. Process. Lett.*, 33(1989/1990), 83–90.

6.10.11. [38] Show that it takes $\Theta(n^{5/4})$ time to transpose a Boolean matrix on a Turing machine with a two-way read-only input tape, a work tape, and a one-way write-only output tape. That is, the input is a $\sqrt{n}\times\sqrt{n}$ matrix A that is initially given on the input tape in row-major order. The Turing machine must output A in column-major order on the output tape.

Comments. Hint: for the upper bound, partition columns of A into $n^{1/4}$ groups each with $n^{1/4}$ consecutive columns. Process each group separately (you need to put each group into a smaller region first). The lower-bound proof is harder. Fix a random matrix A. Let the simulation time be T. Split T into $O(n^{3/4})$ printing intervals. Within each interval, $O(n^{1/4})$ entries of A are printed and half such intervals last fewer than $n^{1/2}$ steps. Also, split the work tape into (disjoint) intervals of size $O(n^{1/2})$, so that one quarter of the printing intervals do not overlap with two work tape intervals. Say, an input bit is mapped to a work tape interval if while the input head is reading that bit, the work tape head is in this interval. A work tape interval is *underinformed* if many bits in the printing interval it corresponds to are not mapped into this work tape interval before they are printed. Show that if there are many underinformed work tape intervals, A is compressible. Then show that if there are not many underinformed intervals, there must be many over burdened intervals, that is, more bits than the length of such intervals are mapped in to each interval. This also implies the compressibility of A.

Source: M. Dietzfelbinger, W. Maass, *Theoret. Comput. Sci.*, 108(1993), 271–290.

6.10.12. [O44] Obtain a tight bound for simulating two work tapes by one work tape for Turing machines with a two-way input tape.

Comments. W. Maass, G. Schnitger, E. Szemerédi, and G. Turan, [*Computational Complexity*, 3(1993), pp. 392–401] proved (not using Kolmogorov complexity) the following: Let $L = \{A\#B : A = B^t$ and $a_{ij} \neq 0$ only when $i,j \equiv 0 \bmod (\log m)$, where $m = 2^k$, for some k, is the size of matrices$\}$. Accepting L requires $\Omega(n\log n)$ time on a Turing machine with a two-way input tape and one work tape. Since L can

be accepted in $O(n)$ time by a similar machine with two work tapes, this result implies that two tapes are better than one for deterministic Turing machines with a two-way input tape. An upper bound to this question is given in Exercise 6.10.10, Item (c).

6.10.13. [40] Consider an online deterministic Turing machine with a one-way input tape, some work tapes/pushdown stores and a one-way output tape. The result of computation is written on the output tape. 'Online simulation' means that after reading a new input symbol the simulating machine must write down precisely the output of the simulated machine for the processed initial input segment before it goes on to read the next input symbol. Prove the following:

(a) It requires $\Omega(n(\log n)^{1/(k+1)})$ time to online simulate $k+1$ pushdown stores by k tapes.

(b) Online simulating one tape plus $k-1$ pushdown stores by k pushdown stores requires $\Omega(n(\log n)^{1/(k+1)})$ time.

(c) Each of the above lower bounds holds also for a probabilistic simulation where the probabilistic simulator flips a random coin to decide the next move. (No error is allowed. The simulation time is the average taken over all coin-tossing sequences.)

Comments. Item (a) is from [W.J. Paul, *Inform. Contr.*, 53(1982), 1–8]. Item (b) is due to P. Dŭriš, Z. Galil, W.J. Paul, and R. Reischuk [*Inform. Contr.*, 60(1984), 1–11]. Item (c) is from [R. Paturi, J. Simon, R. Newman-Wolfe, and J. Seiferas, *Inform. Comput.*, 88(1990), 88–104]. The last paper also includes proofs for Items (a) and (b).

6.10.14. [40] Consider the machine model in Exercise 6.10.13, except that the work tapes are two-dimensional. Such a machine works in real time if at each step it reads a new input symbol and is online. (Then it processes and decides each initial m-length segment in precisely m steps.) Show that for such machines, two work tapes with one head each cannot real-time simulate one work tape with two independent heads.

Comments. Source: W.J. Paul, *Theoret. Comput. Sci.*, 28(1984), 1–12.

6.10.15. [48] As in Exercise 6.10.14, consider the Turing machine model of Exercise 6.10.13 but this time with 1-dimensional tapes. Show that a Turing machine with two single-head one-dimensional tapes cannot recognize the set $\{x2x' : x \in \{0,1\}^*$ and x' is a prefix of $x\}$ in real time, although it can do so with three tapes, two two-dimensional tapes, or one two-head tape, or in linear time with just one tape.

Comments. This is considerably more difficult than the problem in Exercise 6.10.14. In particular, this settles the longstanding conjecture that a two-head Turing machine can recognize more languages in real time

if its heads are on the *same* one-dimensional tape than if they are on *separate* one-dimensional tapes. Source: partial results in [W.J. Paul, *Ibid.*; P.M.B. Vitányi, *J. Comput. System Sci.*, 29(1984), 303–311]. This forty-year open question was finally settled by T. Jiang, J.I. Seiferas, and P.M.B. Vitányi in [*J. Assoc. Comp. Mach.*, 44:2(1997), 237–256].

6.10.16. [38] A *tree work tape* is a complete, infinite, rooted binary tree used as storage medium (instead of a linear tape). A work tape head starts at the root and can in each step move to the direct ancestor of the currently scanned node (if it is not the root) or to either one of the direct descendants. A *multihead tree machine* is a Turing machine with a one-way linear input tape, one-way linear output tape, and several tree work tapes each with $k \geq 1$ heads. We assume that the finite control knows whether two work tape heads are on the same node or not. A *d-dimensional work tape* consists of nodes corresponding to d-tuples of integers, and a work tape head can in each step move from its current node to a node with each coordinate ± 1 of the current coordinates. Each work tape head starts at the origin, which is the d-tuple with all zeros. A *multihead d-dimensional machine* is like the multihead tree machine but with d-dimensional work tapes.

(a) Show that simulating a multihead tree machine online by a multihead d-dimensional machine requires time $\Omega(n^{1+1/d}/\log n)$ in the worst case. Hint: prove this for a tree machine with one tree tape with a single head that runs in real time.

(b) Prove the same lower bound as in Item (a), where the multihead d-dimensional machine is made more powerful by allowing the work tape heads also to move from their current node to the current node of any other work tape head in a single step.

Comments. Source: M.C. Loui, *SIAM J. Comput.*, 12(1983), 463–472. The lower bound in Item (a) is optimal, since it can be shown that every multihead tree machine of time complexity $t(n)$ can be simulated online by a multihead d-dimensional machine in time $O(t(n)^{1+1/d}/\log t(n))$. It is known that every log-cost RAM (Exercise 6.10.17) can be simulated online in real time by a tree machine with one multihead tree tape [W.J. Paul and R. Reischuk, *J. Comput. System Sci.,* 22(1981), 312–327]. Hence, we can simulate RAMs online by d-dimensional machines in time that is bounded above and below by the same bounds as the simulation of tree machines. See also [M.C. Loui, *J. Comput. System Sci.,* 28(1984), 359–378].

6.10.17. [37] A log-cost random access machine (log-cost RAM) has the following components: an infinite number of registers each capable of holding an integer and a finite sequence of labeled instructions including 'output,' 'branch,' 'load/store,' 'add/subtract between two registers.'

The time cost for execution of each instruction is the sum of the lengths of the integers involved.

(a) Every tree machine with several tree tapes, each with one head, of time complexity t can be simulated online by a log-cost RAM of time complexity $O(t \log t / \log\log t)$. Show that this is optimal.

(b) Show that online simulating a linear-time log-cost RAM by a d-dimensional Turing machine requires $\Omega(n^{1+1/d}/\log n(\log\log n)^{1+1/d})$.

Comments. Source: D.R. Luginbuhl, Ph.D. thesis, 1990, Univ. Illinois, Urbana-Champaign; M.C. Loui and D.R. Luginbuhl, [*SIAM J. Comput.* 21:5(1992), 959–971; *Math. Systems Theory*, 25:4(1992), 293–308].

6.10.18. [38] Consider the machine models in Exercise 6.10.13, Item (c). All machines below have one multidimensional tape with one head.

(a) Show that an l-dimensional machine running in time T can be simulated by a probabilistic k-dimensional machine running in time $O(T^r(\log T)^{1/k})$, where $r = 1 + 1/k - 1/l$.

(b) Show that a probabilistic k-dimensional machine requires time $\Omega(T^r)$ to simulate an l-dimensional machine running in time T, where $r = 1 + 1/k - 1/l$.

Comments. Source: [N. Pippenger, *Proc. 14th ACM Symp. Theory Comput.*, 1982, pp. 17–26]. Pippenger used Shannon's information measure to prove Item (b).

6.10.19. [30] Prove that if the number of states is fixed, then a 1-tape nondeterministic Turing machine with no separate input tape (with only one read/write two-way tape) can accept more sets within time bound $a_2 n^a$ than within $a_1 n^a$, for $0 < a_1 < a_2$ and $1 < a < 2$.

Comments. Source: K. Kobayashi, *Theoret. Comput. Sci.*, 40(1985), 175–193.

6.10.20. [30] A *parallel random access machine* (PRAM), also called a 'concurrent-read and concurrent-write priority PRAM,' consists of a finite number of processors, each with an infinite local memory and infinite computing power, indexed as $P(1), P(2), P(3), \ldots$, and an infinite number of shared memory cells $c(i)$, $i = 1, 2, \ldots$, each capable of holding any integer. Initially, the input is contained in the first n memory cells. The number of processors is polynomial in n. Each step of the computation consists of all processors in parallel executing three phases as follows. Each processor (i) reads from a shared memory cell; (ii) performs any deterministic computation; and (iii) may attempt writing into some shared memory cell.

At each step, every processor is in some *state*. The actions and the next state of each processor at each step depend on the current state and the

value read. In case of a *write conflict*, that is, more than one processor tries to write to the same memory cell, the processor with the minimum index succeeds in writing. By the end of computing, the shared memory contains the output. Prove that adding (or multiplying) n integers, each $\geq n^{\epsilon}$ bits for a fixed $\epsilon > 0$, requires $\Omega(\log n)$ parallel steps on a PRAM.

Comments. Hint: Cut a random string x into segments $x_1, \ldots, x_n$, the segments being used as inputs for the processors. Define an input to be 'not useful' if it does not 'influence' the final output of the sum (in some precise way). Then show that there is an input x_i that is not useful; hence we can compress x using the rest of the inputs and the output. Source: Independently proved by P. Beame [*Inform. Comput.*, 76(1988), 13–28] without using Kolmogorov complexity and by M. Li and Y. Yesha [*J. Assoc. Comp. Mach.*, 36:3(1989), 671–680]. Slightly weaker versions of Exercise 6.10.20 were proved by F. Meyer auf der Heide and A. Wigderson [*SIAM J. Comput.*, 16(1987), 100–107] using a Ramsey theorem, by A. Israeli and S. Moran [private communication, 1985], and by I. Parberry [Ph.D. thesis, 1984, Warwick University, UK]. In the last three proofs, one needs to assume that the integers have arbitrarily (or exponentially) many bits.

6.10.21. [15] Consider the following 'proof' for Exercise 6.10.20 without using Kolmogorov complexity: Assume that a PRAM M adds n numbers in $o(\log n)$ time. Take any input $x_1, \ldots, x_n$. Then there is an input x_k that is 'not useful' as in the hint in Exercise 6.10.20. If we change x_k to $x_k + 1$, then the output should still be the same, since x_k is not useful, a contradiction. What is wrong with this proof?

6.10.22. [26] A function $f(x_1, \ldots, x_n)$ is called *invertible* if for each i, argument x_i can be computed from $\{x_1, \ldots, x_n\} - \{x_i\}$ and $f(x_1, \ldots, x_n)$. Use the PRAM model with q processors defined in this section. Show that it requires $\Omega(\min\{\log(b(n)/\log q), \log n\})$ time to compute any invertible function $f(x_1, \ldots, x_n)$, where $l(x_i) \leq b(n)$, for all i, and $\log n = o(b(n))$.

Comments. Source: M. Li and Y. Yesha, *J. Assoc. Comp. Mach.*, 36:3(19 89), 671–680.

6.10.23. [36] *Computing the minimum index*: Modify the PRAM model as follows. We now have n processors $P(1), \ldots, P(n)$ and only one shared memory cell, $c(1)$. Each processor knows one input bit. If several processors attempt to write into $c(1)$ at the same time, then they must all write the same data; otherwise each write fails. This PRAM version requires $\Omega(\log n)$ time to find the smallest index i such that $P(i)$ has input bit 1. Can you give two proofs, one using incompressibility arguments and the other not?

Comments. The original proof without using Kolmogorov complexity is due to F. Fich, P. Ragde, and A. Wigderson [*SIAM J. Comput.*, 17:3(1988), 606–627].

6.11 Communication Complexity

Suppose Alice has input x, Bob has input y, and they want to compute a function $f(x,y)$ by communicating information and by performing local computation according to a fixed protocol. Assume that Alice outputs $f(x,y)$. Local computation costs are ignored; we are interested only in minimizing the number of bits communicated between Alice and Bob. Usually, one considers the worst-case or average-case over all inputs x,y of given length n. But in many situations, for example replicated file systems and cache coherence algorithms in multiprocessor systems and computer networks, the worst-case and average-case are not necessarily significant. From the individual communication complexities we can always obtain the worst-case complexity and the average-case complexity.

Definition 6.11.1 The *individual communication complexity* $CC(x,y|P)$ is defined as the number of bits Alice with input x and Bob with input y need to exchange, both using a communication protocol P. We assume that the protocol is deterministic, possibly partial, and it knows parameter n. A protocol is *total* if it gives a definite result for all inputs, and it is *partial* if it computes correctly on input (x,y) (on other inputs P may output incorrect results or not halt). Note that a protocol implicitly specifies the function being computed in an operational manner.

Let f be a function defined on pairs of strings of the same length. Assume that Alice has x, Bob has y, and Alice wants to compute $f(x,y)$. A (total) communication protocol P over domain X with range Z is a finite rooted binary tree, whose internal nodes are divided into two parts, A and B, called Alice's nodes and Bob's nodes. (They indicate the turn of move.) Each internal node v is labeled by a function $r_v : X \to \{0,1\}$ and each leaf v is labeled by a function $r_v : X \to Z$. A node reached by a protocol P on inputs x,y is the leaf reached by starting at the root of P and walking toward leaves, where in each encountered internal node we go left if $r_v(x) = 0$, and we go right otherwise. This leaf is called the conversation on x,y. Using P on input x and y, Alice computes $z \in Z$ if the leaf v reached on x and y satisfies $z = r_v(x)$. We say that a protocol computes a function $f : X \to Z$ if Alice computes $f(x,y)$ for all $x,y \in X$. The domain X of protocols considered is always equal to the set $\{0,1\}^n$ of binary strings of certain length n. As Z we will take either $\{0,1\}$ or $\{0,1\}^n$.

The length of communication $CC^P(x,y)$ of the protocol P on inputs x,y is the length of the path from the root of P to the leaf reached on x,y. By the complexity of a protocol P we mean $C(P|n)$. Formally, a partial protocol is a protocol, as defined above, but the functions r_v may be partial. The complexity $C(P|n)$ of a partial protocol P is defined as the minimal Kolmogorov

complexity of a program that given n, v determines whether v is a leaf or Alice's internal node or Bob's internal node, and given n, v, x computes $r_v(x)$. If a partial protocol happens to be total (all r_v are total functions) then the new definition of $C(P|n)$ coincides with the old one.

6.11.1 Identity Function

Let $I(x, y) = (x, y)$ be the *identity* function: Alice has to learn Bob's string. If Alice can compute I, then she can compute every computable function f. In the following theorem we consider protocols that compute I on all strings x, y of length n. By definition of the Kolmogorov complexity, the lower bound for the number of bits to be transmitted is $C(y|x, P)$. Since the number m of halting programs of length $n + O(1)$ satisfies $m \leq 2^{n+O(1)}$, we can determine the halting of all programs of length up to $n + O(1)$ if we are given m: Run all programs up to that length dovetail fashion; by the time m of them have halted we know that the remainder will never halt. In this way, we can determine the shortest program, given n, for every string y of length n, since $C(y) \leq n + O(1)$. Thus, Bob using a protocol P, containing m, can find the shortest program for y and send it to Alice, who computes y. Thus, $CC(x, y|P) \leq C(y|n) + O(1)$ with $C(P) \leq n + O(1)$. By Theorem 3.8.1 on page 242 we see that $C(P) \geq n - O(\log n)$ if it is supposed to work for every y of length n.

Theorem 6.11.1 *For every protocol P for the identity function I, and every x, y, we have $CC(x, y|P) \geq C(y|P) - O(1) \geq C(y|n) - C(P|n) - O(\log C(P|n))$.*

Proof. Let c be the conversation between Alice and Bob on inputs x, y. It suffices to show that given P, c we can find y. We call a set $R \subseteq A \times A$ a *rectangle* if whenever both (x_1, y_1) and (x_2, y_2) are in R, then so is (x_1, y_2). The definition of a communication protocol implies that the set of all pairs (x', y') such that the conversation between Alice and Bob on input (x', y') is equal to c is a 'rectangle', that is, has the form $X \times Y$, for some $X, Y \subset \{0, 1\}^n$. The set Y is a one-element set, since for every $y' \in Y$ Alice outputs y also on the input (x, y') (the output of Alice depends on c, P, x only). We can find Y given P, c, and since $Y = \{y\}$, we are done. □

Example 6.11.1 We look at the special case $x = y$. For every P, there are x, y with $CC(x, y|P) \geq C(y|x) + n - O(1)$. This holds in particular for $y = x$ with $C(y) \geq n$. Then $C(y|x) = O(1)$ and by Theorem 6.11.1 we have $CC(x, y|P) \geq C(y|P) - O(1) \geq C(y|x) + n - O(1)$. ◇

6.11.2 Inner Product Function

Initially, Alice has a string $x = x_1, \ldots, x_n$ and Bob has a string $y = y_1, \ldots, y_n$ with $x, y \in \{0,1\}^n$. Alice and Bob compute

$$f(x,y) \equiv \left(\sum_{i=1}^{n} x_i \cdot y_i \right) \bmod 2,$$

with Alice ending up with the result.

Lemma 6.11.1 Every protocol P computing the inner product function f requires at least $CC(x,y|P) \geq C(x,y|P) - n - O(1)$ bits of communication for every pair x, y.

Proof. Fix a communication protocol P that computes the inner product. Let Alice's and Bob's input be as above. Run the communication protocol P on x, y and let $c(x,y)$ be a record of the communication between Alice and Bob. Consider the set $S = S(x,y)$ defined by

$$S := \{(a,b) : c(a,b) = c(x,y), \quad \text{and Alice outputs } f(x,y) \text{ on conversation } c(x,y) \text{ and input } a\}.$$

We claim that $d(S) \leq 2^n$. To prove this, assume first that $f(x,y) = 0$. Let $X = \{a : (a,b) \in S\}$ be the first projection of S and let $Y = \{b : (a,b) \in S\}$ be the second projection of S. Since P computes f we know that $f(a,b) = 0$ for all $(a,b) \in S$. In other words, every element of X is orthogonal to every element in Y, and therefore $\mathrm{rank}(X) + \mathrm{rank}(Y) \leq n$. Thus,

$$d(S) = d(X) \cdot d(Y) \leq 2^{\mathrm{rank}(X)+\mathrm{rank}(Y)} \leq 2^n.$$

Assume now that $f(x,y) = 1$. Again $S = X \times Y$ for some X, Y and $f(a,b) = 1$ for all $(a,b) \in S$. Subtracting x from the first component of all pairs in S, we obtain a rectangle S' such that $f(a,b) = 0$ for all $(a,b) \in S'$. By the above argument, we have $d(S') \leq 2^n$. Since $d(S') = d(S)$ we are done. Given P, $c(x,y)$, $f(x,y)$, and the index of (x,y) in S we can compute (x,y). Padding the index of (x,y) up to length n, while n is known by the protocol, we observe that the index of (x,y) and $c(x,y)$ can be concatenated without delimiters. Consequently, $C(x,y|P) \leq l(c(x,y)) + n + O(1)$. □

Since there are 2^{2n} pairs of n-length strings, we can choose x, y with $C(x,y|P) \geq 2n$. Thus, the worst-case communication complexity for the function f is $n - c$. There are $2^{2n} - 2^{2n-c_1}$ pairs x, y with $C(x,y|P) \geq 2n - c_1$. Hence, the average-case communication complexity for the function f is $n - O(1)$.

Exercises

6.11.1. [24] Assume that a function $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ satisfies $C(f|n) \geq 2^{2n} - n$: the truth table describing the outcomes of f for the 2^n possible inputs x (the rows) and the 2^n possible inputs for y (the columns) has high Kolmogorov complexity. If we flip the truth table for a prospective f using a fair coin, then with probability at least $1-2^{-n}$ it will satisfy this. Show that every deterministic protocol P computing such a function f requires at least $CC(x, y|P) \geq \min\{C(x|P), C(y|P)\} - \log n - O(1)$.

Comments. Source: H.M. Buhrman, H. Klauck, N.K. Vereshchagin, P.M. B. Vitányi, *J. Comput. Syst. Sci.* 73(2007), 973–985.

6.11.2. [24] Let f be the equality function, with $f(x, y) = 1$ if $x = y$ and 0 otherwise. Show that for every deterministic protocol P computing f, we have $CC(x, x|P) \geq C(x|P) - O(1)$ for all x, y. On the other hand, there is a P of complexity $O(1)$ such that there are x, y $(x \neq y)$ with $C(x|P), C(y|P) \geq n - 1$ for which $CC(x, y|P) = 2$.

Comments. Source: H.M. Buhrman, H. Klauck, N.K. Vereshchagin, P.M. B. Vitányi, *Ibid.*

6.11.3. [35] Define the protocol-independent communication complexity $TCC(x, y|C(P) \leq i)$, of computing a function $f(x, y)$, as the minimum $CC(x, y|P)$ over all deterministic total protocols P computing $f(x, y)$ for all pairs (x, y) $(l(x) = l(y) = n)$ with $C(P) \leq i$. For example, $TCC(x, y|C(P) \leq n + O(1)) = 0$ for all computable functions f and all x, y.

(a) Show that for every computable function $f(x, y)$ its $TCC(x, y|C(P) \leq i)$ is always at most the $TCC(x, y|C(P) \leq i)$ of the identity function $I(x, y) = (x, y)$, for every x, y, i.

(b) Show that for every computable function f we have $TCC(x, y|C(P) \leq i) \geq C(f(x, y)|x) - i - O(\log i)$. For $f = I$ this gives $TCC(x, y|C(P) \leq i) \geq C(y|x) - i - O(\log i)$.

(c) Show that for the identity function I, restricting the protocols to one-way (Bob sends a single message to Alice only) does not significantly alter the protocol-independent communication complexity for total protocols: $TCC(x, y|C(P) \leq i+O(1), P \text{ is one-way}) \leq TCC(x, y|C(P) \leq i)$, where in the right-hand side P is allowed to be two-way.

Comments. Source: H.M. Buhrman, H. Klauck, N.K. Vereshchagin, P.M. B. Vitányi, *Ibid.*

6.11.4. [37] We continue Exercise 6.11.3. Let $h_x(i)$ be the structure function as in Definition 5.5.6 on page 405. Define, with P a protocol that computes the identity function I, the protocol-size function $p_y(j) = \min\{i : TCC(x, y|C(P) \leq i, P \text{ is one-way }) \leq j\}$. The function

$p_y(j)$ gives the minimal number of bits of a protocol of the total deterministic type considered that transmits $y \in \{0,1\}^*$ in at most j bits of communication. By Exercise 6.11.3, Item (c), total one-way protocols are as powerful as total two-way protocols of about the same complexity. Note that the one-way protocol does not depend on x.

(a) Show that $p_y(j) = h_y(j) + O(\log n)$ for all y and j.

(b) Use item (a) to show the following: (i) For every string y of length n we have $p_y(n) = O(1)$ and $0 \leq p_y(j) - p_y(k) \leq k - j + O(\log n)$, for every $j < k \leq n$. Conversely, if p is a function from $\{0, 1, \ldots, n\}$ to the natural numbers satisfying the conditions in (i), with the $O(1), O(\log n)$ replaced by 0, then there is a string y of length n such that $p_y(j) = p(j) + O(\log n + C(p))$, where $C(p)$ stands for the complexity of the set $\{(j, p(j)) : j \in \{0, \ldots, n\}\}$.

(c) Show that there exist *noncommunicable strings* in the following sense. Let $k < n$. Apply Item (b) to the function p defined as $p(j) = k$ for $j \leq n-k$ and $p(j) = n-j$ for $j \geq n-k$. By Item (b) there exists a string y of length n such that $p_y(0) = k + O(\log n)$ (thus $C(y) = k + O(\log n)$) and the protocol-independent communication complexity for the identity function I is $TCC(x, y|C(P) \leq i, P \text{ is one-way}) > n - i - O(\log n)$ for every $i < k - O(\log n)$.

Comments. Item (c) shows that Bob can hold a highly compressible string y, but cannot use that fact to reduce the communication complexity significantly below $l(y)$. Unless *all* information about y is hardwired into the protocol, the communication between Bob and Alice requires sending y almost completely literally. Indeed, For such y with, say, $C(y) = \log^{O(1)} n$, we have (irrespective of x) communication complexity that is *exponential* in the complexity of y for all protocols of complexity less than that of y. When the complexity of the protocol reaches the complexity of y, the communication complexity suddenly drops to 0. Source: H.M. Buhrman, H. Klauck, N.K. Vereshchagin, P.M.B. Vitányi, *Ibid.*

6.11.5. [33] Let the protocol-independent communication complexity $PCC(x, y|C(P) \leq i)$ stand for the minimum $CC^P(x, y)$ over all partial deterministic protocols P of complexity at most i computing f correctly on input (x, y) (on other inputs P may output incorrect results or not halt). Trivially, $PCC(x, y|C(P) \leq i) \leq TCC(x, y|C(P) \leq i)$ for every computable function.

(a) Show that for the identity function I we have $C(y|x) - i - O(\log i) \leq PCC(x, y|C(P) \leq i) \leq PCC(x, y|C(P) \leq i, P \text{ is one-way}) \leq C(y)$ for all x, y, i such that i is at least $\log C(y) + O(1)$. The addition 'one-way' means that Bob communicates with Alice but not vice versa.

(b) Prove that for the identity function I we have $PCC(x, y|C(P) = O(\log n), P \text{ is one-way}) \leq C(y|x) + O(\log n)$, for all x, y of length n.

Comments. Item (a) is obvious. Comparing Item (b) with Exercise 6.11.4, Item (c), we see that protocol-independent communication complexity for the identity function I of one-way partial deterministic protocols is strictly less than that of one-way total deterministic protocols (there are no noncommunicable objects for protocol-independent communication complexity of partial protocols). Moreover, by Exercise 6.11.3, Item (c), the protocol-independent communication complexity for the identity function I of one-way total deterministic protocols equals that of two-way ones. Hint for Item (b): use Muchnik's theorem, Theorem 8.3.7, on page 654. H.M. Buhrman, H. Klauck, N.K. Vereshchagin, P.M.B. Vitányi, *Ibid.*

6.11.6. [34] In Theorem 6.11.1 it was shown that for a deterministic protocol of say, complexity $O(1)$, to compute the identity function Alice and Bob need to exchange about $C(y)$ bits, even if the required information $C(y|x)$ is much less than $C(y)$. Show that for randomized protocols the communication complexity is close to $Cy|x)$.

Comments. Source: H.M. Buhrman, M. Koucký, N.K. Vereshchagin, Randomized individual communication complexity, Manuscript, CWI, 2006.

6.12 Circuit Complexity

A key lemma in the study of circuit complexity is the so-called Håstad's switching lemma. It is used to separate depth-k and depth-$(k+1)$ circuit classes, and to construct oracles relative to which the polynomial hierarchy is infinite and properly contained in PSPACE. The traditional proof of this lemma uses sophisticated probabilistic arguments. We describe a simple elementary proof using the incompressibility method.

According to Definition 5.3.3 on page 376, a k-DNF formula is a disjunction of conjunctions with each conjunct (or term) containing at most k literals. A k-CNF is a conjunction of disjunctions with each disjunct (or clause) containing at most k literals.

Definition 6.12.1 A *restriction* ρ is a function from a set of variables to $\{0, 1, \star\}$. Given a Boolean function f, $f|_\rho$ is the restriction of f in the natural way: x_i is free if $\rho(x_i) = \star$ and x_i takes on the value $\rho(x_i)$ otherwise. The *domain* of a restriction ρ, $\text{dom}(\rho)$, is the set of variables mapped to 0 or 1 by ρ.

We can also naturally view a restriction ρ as a *term* of f if $f|_\rho = 1$. A *minterm* is a restriction such that no proper subset of the variables set by the restriction forms a term. Let R_l be the set of restrictions on n variables that leave l variables free. Obviously, $d(R_l) = \binom{n}{l}2^{n-l}$.

Lemma 6.12.1 **(Switching lemma)** *Let f be a t-CNF on n variables, ρ a random restriction $\rho \in R_l$ and $\alpha = 12tl/n \leq 1$. Then the probability that $f|_\rho$ is an s-DNF is at least $1 - \alpha^s$.*

Proof. Fix a t-CNF f on n variables, and integers s and $l < n$. Note that $f|_\rho$ is l-DNF; we can assume $s \leq l$. In this proof, we will use conditional complexity $C(\cdot|\mathbf{x})$, where $\mathbf{x}$ denotes the list of fixed values of f, t, l, n, s and several (fixed) programs needed later.

Claim 6.12.1 For any $\rho \in R_l$ such that $f|_\rho$ is not s-DNF, ρ can be effectively described by some $\rho' \in R_{l-s}$, a string $\sigma \in \{0, 1, \star\}^{st}$ such that σ has s non$\star$ positions, and $\mathbf{x}$. That is, $C(\rho|\rho', \sigma, \mathbf{x}) = O(1)$.

Before proving Claim 6.12.1, we show that it implies the switching lemma. Fix a random restriction $\rho \in R_l$ with

$$C(\rho|\mathbf{x}) \geq \log(d(R_l)\alpha^s), \tag{6.22}$$

where $\alpha = 12tl/n \leq 1$. If we show that $f|_\rho$ is an s-DNF, then since there are at least $d(R_l)(1 - \alpha^s)$ ρ's in R_l satisfying Equation 6.22 by Theorem 2.2.1, this will imply the lemma.

Assume that $f|_\rho$ is not an s-DNF and $\rho' \in R_{l-s}$ as in Claim 6.12.1. Obviously $C(\rho'|\mathbf{x}) \leq \log d(R_{l-s})$. Since $l(\sigma) = st$ and σ has s non$\star$ positions, we have

$$C(\sigma|\mathbf{x}) \leq \log \binom{st}{s} + s \leq s \log et + s = s \log 2et,$$

by standard estimation (Stirling's approximation), where $e = 2.718\ldots$. By Claim 6.12.1, we have

$$C(\rho|\mathbf{x}) \leq C(\rho'|\mathbf{x}) + C(\sigma|\mathbf{x}) \leq \log d(R_{l-s}) + s \log 2et. \tag{6.23}$$

By Equations 6.22 and 6.23, we have

$$d(R_l)\alpha^s \leq d(R_{l-s})2^{s \log 2et}.$$

Substituting $\binom{n}{l}2^{n-l}$ for $d(R_l)$ and $\binom{n}{l-s}2^{n-l+s}$ for $d(R_{l-s})$, and using the fact $\binom{n}{l} / \binom{n}{l-s} \geq ((n - l + s)/l)^s$, we obtain

$$\frac{12tl}{n} \leq \frac{4etl}{n - l + s}.$$

But the above formula cannot hold simultaneously with $12tl/n \leq 1$, a contradiction. This proves the switching lemma.

Proof. (For Claim 6.12.1) Given a t-CNF f on n variables and a restriction $\rho \in R_l$ such that $f|_\rho$ is not s-DNF. Let

$$f = \bigwedge_{j=1}^{k} D_j, \tag{6.24}$$

where each D_j is a disjunct of size at most t. We also can write $f|_\rho$ as a DNF: $f|_\rho = \vee_j C_j$, where each C_j (the corresponding restriction) is a minterm of $f|_\rho$. Since $f|_\rho$ is not an s-DNF, there must be a minterm π that contains at least $s+1$ variables. We will extend ρ to ρ' using s of the variables of π.

First we split π into subrestrictions. Assume that $\pi_1, \ldots, \pi_{i-1}$ have already been defined and $\mathrm{dom}(\pi) - \mathrm{dom}(\pi_1 \cdots \pi_{i-1}) \neq \varnothing$. Choose the first disjunct D_j in Equation 6.24 that is not already 1 under restriction $\rho\pi_1 \cdots \pi_{i-1}$. Let S be the set of variables that appear both in D_j and in $\mathrm{dom}(\pi) - \mathrm{dom}(\pi_1 \cdots \pi_{i-1})$. Define π_i as

$$\pi_i(x) = \begin{cases} \pi(x) & \text{if } x \in S, \\ \star & \text{otherwise.} \end{cases}$$

Because π is a minterm, it must force each disjunct to 1, and no subrestriction of π (namely $\pi_1 \cdots \pi_{i-1}$) will. Thus the above process is always possible. Let k be the least integer such that $\pi_1 \cdots \pi_k$ sets at least s variables. Trim π_k so that $\pi_1 \cdots \pi_k$ sets exactly s variables.

Change π_i to $\tilde{\pi}_i$: for each variable $x \in \mathrm{dom}(\pi_i)$, if it appears in the corresponding D_j as x then $\tilde{\pi}_i(x) = 0$; if it appears in D_j as $\bar{x}$ then $\tilde{\pi}_i(x) = 1$. Thus $\pi_i \neq \tilde{\pi}_i$, since $\rho\pi_1 \cdots \pi_{i-1}\pi_i$ forces D_j to be 1 but $\rho\pi_1 \cdots \pi_{i-1}\tilde{\pi}_i$ does not. If x is the mth variable in D_j, the mth digit of $\sigma^{(i)}$ is

$$\sigma_m^{(i)} = \begin{cases} \pi_i(x) & \text{if } x \in \mathrm{dom}(\pi_i)\ (= \mathrm{dom}(\tilde{\pi}_i)), \\ \star & \text{otherwise.} \end{cases}$$

Since D_j is of size at most t, $l(\sigma^{(i)}) = l(D_j) \leq t$. Let

$$\rho' = \rho\tilde{\pi}_1 \cdots \tilde{\pi}_k, \quad \sigma = \sigma^{(1)} \cdots \sigma^{(k)} \star^{st-kt}.$$

Pad σ with $\star$'s so that $l(\sigma) = st$. Since $\pi_1 \cdots \pi_k$ sets exactly s variables, σ has s non$\star$ positions.

Now we show how to recover $\pi_1, \ldots, \pi_k$, hence ρ, from σ and ρ' (given $\mathbf{x}$). Assume that we have already recovered $\pi_1, \ldots, \pi_{i-1}$, from which we can infer $\rho\pi_1 \cdots \pi_{i-1}\tilde{\pi}_i \cdots \tilde{\pi}_k$, using ρ'. Recall that π_i was defined by choosing the first clause D_j not already forced to 1 by $\rho\pi_1 \cdots \pi_{i-1}$. Since $\tilde{\pi}_i$ does not force D_j to be 1 and $\tilde{\pi}_{i+1} \cdots \tilde{\pi}_k$ are defined on variables not

contained in D_j, we simply identify D_j from $f|_{\rho\pi_1\cdots\pi_{i-1}\bar{\pi}_i\cdots\bar{\pi}_k}$, as the first non-1 clause. Given D_j, recover π_i using $\sigma^{(i)}$. With $\rho\pi_1\cdots\pi_k$ and $\pi_1\cdots\pi_k$, we can recover ρ. This proves Claim 6.12.1, and hence the theorem. □ □

Let us summarize the central ideas in the above proof. When we choose a random ρ, the number of bits needed to specify each extra variable is roughly $O(\log n)$. However, the fact that $f|_\rho$ is not an s-DNF implies that it has a long minterm, and this allows us to construct a σ, together with ρ', specifying s extra variables at the expense of roughly $\log 2et$ bits per variable. So a large term is a kind of regularity a random restriction ρ does not produce.

Example 6.12.1 Lemma 6.12.1 is a powerful lemma in circuit complexity. Let's define a depth-k (unbounded fan-in Boolean) circuit as follows: The input to the circuit is $I = \{x_1, \ldots, x_n, \bar{x}_1, \ldots, \bar{x}_n\}$. The circuit has k alternating levels of AND and OR gates, each with unbounded fan-in. The kth (top) level contains just one gate, which gives the output of the circuit. Each gate in the ith level gets an arbitrary number of inputs from the outputs of the $(i-1)$st level, assuming that I is at the zeroth level. A *parity* function $f(x_1 \ldots x_n)$ equals 1 if and only if an odd number of x_i's are 1's. It is easy to show that a polynomial-size depth-2 circuit cannot compute parity. Assume that this is the case for $k-1$. For a depth-k circuit, if we apply a random restriction to it, then by Lemma 6.12.1, with high probability, we can switch the bottom two levels, say, from AND-OR to OR-AND. Then the second-level OR can merge with the third-level OR, hence reducing the circuit depth to $k-1$. Note that a restriction of a parity function remains a parity function. Making this kind of induction precise, one can prove the following: there is a constant $c > 0$ such that a depth-k circuit with at most $2^{c^{k/k-1}n^{1/k-1}}$ gates cannot compute parity. ◇

6.13 History and References

Apparently, W.J. Paul [*Proc. Int. Conf. Fund. Comput. Theory*, L. Budach, ed., 1979, pp. 325–334] is the pioneer of using the incompressibility method, and he proved several lower bounds with it. R.V. Freivalds ["On the running time of deterministic and nondeterministic Turing machines," *Latv. Mat. Ezhegodnik*, 23(1979) 158–165 (in Russian)] proved a lower bound on the time of Turing machine computations for a certain problem, implicitly using a Kolmogorov complexity argument in a veiled form of 'optimal enumerations,' justified by the invariance theorem, Theorem 2.1.1. He did not use the incompressibility method.

The initially most influential paper is probably the paper by W.J. Paul, J. Seiferas, and J. Simon, [*J. Comput. System Sci.*, 23:2(1981), 108–126]. This was partly because the paper by W.J. Paul, which contains the example of Section 6.1.1, was not widely circulated.

The aim of the paper by Paul, Seiferas, and Simon was "to promote the approach" of applying Kolmogorov complexity to obtain lower bounds. In that paper, using incompressibility arguments, the authors greatly simplified the proof of a difficult theorem proved by S.O. Aanderaa [pp. 75–96 in: *Complexity of Computation*, R. Karp, ed., Amer. Math. Soc., 1974], which proves that *real time* simulation of k tapes by $k-1$ tapes is impossible for deterministic Turing machines. Earlier, M.O. Rabin [*Israel J. Math.*, 1(1963), 203–211] proved the particular case $k = 2$ of this result. In 1982, W.J. Paul [*Inform. Contr.,* 53(1982), 1–8] further improved Aanderaa's result from real-time to nonlinear lower bounds by incompressibility arguments. In the same year, S. Reisch and G. Schnitger [*Proc. 23rd IEEE Found. Comput. Sci.*, 1982, pp. 45–52] published a paper giving three applications of incompressibility in areas other than Turing machine computational complexity. (The authors later lost contact with each other and they have never written up a journal version of this paper.) Subsequently, incompressibility arguments started to be applied to an ever-increasing variety of problems.

Lemma 6.1.1 in Section 6.1.1 was first proved by F.C. Hennie using a counting argument in [*Inform. Contr.*, 8:6(1965), 553–578]. The proof we give here is due to W.J. Paul. Section 6.1.2 is based on [R. Beigel, W. Gasarch, M. Li, and L. Zhang, *Theoret. Comput. Sci.*, 191(1998), 245–248]. The original probabilistic analysis is in [A.W. Burks, H.H. Goldstine, and J. von Neumann, "Preliminary discussion of the logical design of an electronic computing instrument," Institute for Advanced Studies, Report (1946). Reprinted in *John von Neumann Collected Works*, Vol. 5, 1961]. Improved probabilistic analysis can be found in [B.E. Briley, *IEEE Trans. Computers*, C-22:5(1973)] and [G. Schay, *Amer. Math. Monthly*, 102:8(1995), 725–730]. Background material on adder design can be found in [K. Hwang, *Computer Arithmetic: Principles, Architecture, and Design,* Wiley, New York, 1979]. Lemma 6.1.3 in Section 6.1.3 is due to J. Seiferas and Y. Yesha [Personal communication, 1986]. The idea of proving a lower time bound for palindrome recognition by a probabilistic Turing machine, as mentioned in the comment at the end of Section 6.1 and in Exercise 6.10.13, Item (c), is due to R. Paturi, J. Simon, R. Newman-Wolfe, and J. Seiferas, [*Inform. Comput.*, 88(1990), 88–104].

The discussion in Section 6.2 on the quantitative relation between high-probability properties of finite objects and individual randomness of finite objects is taken from [H.M. Buhrman, M. Li, J.T. Tromp, and P.M.B. Vitányi, *SIAM J. Comput.*, 29:2(1999), 590–599]. With respect

to infinite binary sequences the distinction between laws of probability (that hold with probability one) and individual random sequences is discussed in [M. van Lambalgen, *Random Sequences*, PhD thesis, Universiteit van Amsterdam, Amsterdam, 1987] and [V.V. Vyugin, *Theory Probab. Appl.*, 42:1(1996), 39–50].

Section 6.3, on combinatorics, follows [M. Li and P.M.B. Vitányi, *J. Comb. Theory, Ser. A*, 66:2(1994), 226–236]. The problems on tournaments in Section 6.3 are from [P. Erdős and J.H. Spencer, *Probabilistic Methods in Combinatorics*, Academic Press, 1974]. See also [N. Alon, J.H. Spencer, and P. Erdős, *The Probabilistic Method*, Wiley, 1992] for the probabilistic method. Section 6.3.3 was suggested by W. Gasarch; the lower bound on the Ramsey numbers was originally proved in [P. Erdős, *Bull. Amer. Math. Soc.*, 53(1947), 292–294]; see [P. Erdős and J.H. Spencer, *Ibid.*]. The lower bound for the coin-weighing problem in Theorem 6.3.4 was established, using probabilistic or information-theoretic methods, by P. Erdős and A. Rényi [*Publ. Hungar. Acad. Sci.*, 8(1963), 241–254], L. Moser [*Combinatorial Structures and Their Applications*, Gordon and Breach, 1970, pp. 283–384], and N. Pippenger [*J. Comb. Theory, Ser. A*, 23(1977), 105–115]. The last paper contains proofs, by entropy methods, of Theorem 6.3.4 on page 455 and Exercise 6.3.4 on page 457. Recently, entropy methods have also been used quite successfully in proving lower bounds on parallel sorting [J. Kahn and J. Kim, *Proc. 24th ACM Symp. Theory Comput.*, 1992, pp. 178–187], perfect hashing [I. Newman, P. Ragde, and A. Wigderson, *5th IEEE Conf. Structure in Complexity Theory*, 1990, pp. 78–87], and lower bounds on parallel computation [R. Boppana, *Proc. 21st ACM Symp. Theory Comput.*, 1989, pp. 320–326]. On Exercise 6.3.11 on page 460 Ramsey-type results, that were earlier obtained using Lovász's local lemma, are obtained by incompressibility.

Section 6.4 on graphs is primarily based on [H.M. Buhrman, M. Li, J.T. Tromp, and P.M.B. Vitányi, *SIAM J. Comput.*, 29:2(1999), 590–599]. For random graphs in a probabilistic sense see for example [B. Bollobás, *Random Graphs*, Academic Press, 1985]. The statistics of subgraphs of high-complexity graphs, Theorem 6.4.1, has a corresponding counterpart in quasirandom graphs, and a similar expression is satisfied almost surely by random graphs [N. Alon, J.H. Spencer, and P. Erdős, *The Probabilistic Method*, Wiley, 1992] pp. 125–140; see especially Property $P_1(s)$ on page 126. The latter property may be weaker in terms of quantification of 'almost surely' and the $o(\cdot)$ and $O(\cdot)$ estimates involved than the result we present here.

The results in Section 6.5 on routing tables are from [H.M. Buhrman, J.H. Hoepman, and P.M.B. Vitányi, *SIAM J. Comput.*, 28:4(1999), 1414–1432]. Related research (see exercises) appears in [E. Kranakis and D.

Krizanc [*Proc. 3nd Int. Colloq. Structure Inform. Communication Complexity*, Siena, Italy, 1996, pp. 119–124; *Proc. 13th Symp. Theoret. Aspects Comput. Sci.*, 1996, pp. 529–540], and E. Kranakis, D. Krizanc, and F. Luccio, *Proc. 13th Symp. Math. Found. Comput. Sci.*, 1995, pp. 392–401].

Heapsort was originally discovered by J.W.J. Williams [*Comm. Assoc. Comp. Mach.*, 7(1964), 347–348]. R.W. Floyd [*Comm. Assoc. Comp. Mach.*, 7(1964), 701] subsequently improved the algorithm. Researchers had previously tried to analyze the precise average-case complexity of Heapsort with no success. For example, the analysis typically works only for the first step; after one step, the heap changes and certain properties such as that all heaps are equally likely no longer hold. Section 6.6.1 is based on an explanation by I. Munro on a summer evening in 1992. The solution to the average-case complexity of Heapsort was first obtained by R. Schaffer and R. Sedgewick [*J. Algorithms*, 15(1993), 76–100]. The proof in the form given in Section 6.6.1 is due to I. Munro. Claim 6.6.1, that the Heapify procedure produces a random heap from a random input, was observed by T. Jiang, at a 1993 Dagstuhl seminar, and I. Munro.

Shellsort was discovered by D.L. Shell [*Comm. Assoc. Comp. Mach.*, 2:7(1959), 30–32)]. Section 6.6.2 on Shellsort is based on [T. Jiang, M. Li, and P.M.B. Vitányi [*Proc. Int. Colloq. Aut. Lang. Progr.*, Lect. Notes Comp. Sci., Vol 1644, Springer-Verlag, Berlin, 1999, 453–462; *J. Assoc. Comp. Mach.* 47:5(2000), 905–911], where the reader can also find papers on worst-case analysis of Shellsort not discussed here. Previously, D.E. Knuth [*The Art of Computer Programming, Vol. 3: Sorting and Searching*, Addison-Wesley, 1973, 1998] showed that the average running time for two-pass Shellsort is $\Theta(n^{5/3})$ for the best choice of increments; A.C.C. Yao [*J. Alg.*, 1(1980), 14–50] analyzed the three-pass case without giving a definite running time; Yao's analysis was extended by S. Janson and D.E. Knuth [*Random Struct. Alg.* 10(1997), 125–142] to an $O(n^{23/15})$ upper bound on the average running time for three-pass Shellsort.

Section 6.7 on longest common subsequences is from [T. Jiang and M. Li, *SIAM J. Comput.*, 24:5(1995), 1122–1139]. Section 6.8 on formal language theory follows [M. Li and P.M.B. Vitányi, *SIAM J. Comput.*, 24:2(1995), 398–410]. For the history of and an introduction to formal language theory, see [M.A. Harrison, *Introduction to Formal Language Theory*, Addison-Wesley, 1978; J.E. Hopcroft and J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, 1979].

The proof in Section 6.9 of the lower bound on the time required for linear context-free language recognition is due to J. Seiferas [*Inform. Contr.*, 69(1986), 255–260], simplifying the original proof of H. Gallaire

[*Inform. Contr.*, 15(1969), 288–295]. The latter paper improved a $\log n$ multiplicative factor weaker result by F.C. Hennie. Gallaire's proof uses a complicated counting argument and de Bruijn sequences. T. Kasami [*Inform. Contr.*, 10(1967), 209–214] proved that linear context-free languages can be online recognized in $O(n^2)$ by a one-work-tape Turing machine.

It is fair to say that the solutions to the conjectures on k-PDA in Exercise 6.9.3, string-matching in Exercise 6.9.2, Theorem 6.10.1, and many exercises in Section 6.10 would not have been possible, at least not proved in such a short period of time, without the use of incompressibility arguments. Recently, T. Jurdziński and K. Loryś [*Proc. 29th Int. Colloq. Aut., Lang., Prog.*, 2002, pp. 147–158] used the incompressibility method to prove a 1988 McNaughton–Narendran–Otto conjecture that the Church–Rosser languages do not contain the set of palindromes, hence not CFL $\bigcap$ coCFL. The results in Section 6.10 concerning whether an extra tape adds computational power in various Turing machine models and especially the 'two heads are better than two tapes' result in Exercise 6.10.15, could probably not be proven without the incompressibility method. These results were open for decades before they were solved using the incompressibility method and some other Kolmogorov complexity-related techniques, and no other proofs are known.

The results and methods of Section 6.10 on lower bounds for time complexity of Turing machines were instrumental in initiating large-scale use of the incompressibility method. It is well known and easy that if a k-tape Turing machine runs in $O(T(n))$ time, then it can be simulated by a 1-tape Turing machine in $O(T^2(n))$ time [J. Hartmanis and R. Stearns, *Trans. Amer. Math. Soc.*, 117(1969), 285–306] and by a 2-tape Turing machine in $O(T(n) \log T(n))$ time [F.C. Hennie and R. Stearns, *J. Assoc. Comp. Mach.*, 4(1966), 533–546]. For years, only several weak lower bounds were known with complicated proofs, such as M.O. Rabin's paper from 1963 and S.O. Aanderaa's paper of 1974 above. These papers consider the restricted online model with an extra output tape. For the more general model used in Theorem 6.10.1, P. Dŭriš, Z. Galil, W.J. Paul, and R. Reischuk [*Inform. Contr.,* 60(1984), 1–11] proved that it requires $\Omega(n \log n)$ time to simulate two tapes by one. Research advanced quickly only after the incompressibility argument was invented. W.J. Paul [*Inform. Contr.*, 53(1982), 1–8] proved Exercise 6.10.13, Item (a), on page 512, improving Aanderaa's result. Around 1983/1984, independently and in chronological order, Wolfgang Maass at UC Berkeley, one of us [ML] at Cornell, and the other one [PV] at CWI Amsterdam, obtained an $\Omega(n^2)$ lower bound on the time to simulate two tapes by one tape (deterministically), and thereby closed the gap between 1 tape versus $k \geq 2$ tapes (Exercise 6.10.2 on page 508). All three relied on Kolmogorov complexity, and actually proved more in various ways.

(One of us [PV], at first not realizing how to use incompressibility, reported in [P.M.B. Vitányi, *Theoret. Comput. Sci.,* 34(1984), 157–168] an $\Omega(n^{3/2})$ lower bound on the time to simulate a single pushdown store online by one *oblivious* tape unit. However, after being enlightened by J. Seiferas about how to use incompressibility with respect to another result, he realized how to apply it to the 1-tape versus 2-tape problem without the oblivious restriction [P.M.B. Vitányi, *Inform. Process. Lett.,* 21(1985), 87–91 and 147–152], and the optimal results cited below.) W. Maass also obtained a nearly optimal (almost square) lower bound for nondeterministic simulation (Exercise 6.10.4 on page 508) [W. Maass, *Trans. Amer. Math. Soc.,* 292(1985), 675–693]. Maass's lower bound on nondeterministic simulation was improved bc Z. Galil, R. Kannan, and E. Szemerédi [*Proc. 18th ACM Symp. Theory Comput.*, 1986, pp. 39–49] to $\Omega(n^2/\log^{(k)} n)$ by constructing a language whose computation graph does not have small separators (Exercise 6.10.5 on page 509). The exercises contain many more lower bounds which were proved in this direction. Section 6.10 is based on [M. Li and P.M.B. Vitányi, *Inform. Comput.,* 78(1988), 56–85], which also contains results on tapes versus stacks and queues. Many lower bounds for various models of computation, such as machines with extra two-way input tapes, machines with queues, random access machines, machines with many heads on a tape, machines with tree tapes, machines with k-dimensional tapes, and probabilistic machines, have since been proved using Kolmogorov complexity. We have tried to cover these results in the exercises, where also the references are given.

Communication complexity was invented by A.C.C. Yao, [*Proc. 11th ACM Symp. Theory Comput.*, 1979, 209–213]. The main reference is [E. Kushilevitz, N. Nisan, *Communication Complexity*, Cambridge Univ. Press, 1997]. These works consider the worst-case or average-case complexity. Section 6.11, on individual communication complexity, is based on [H.M. Buhrman, H. Klauck, N.K. Vereshchagin, P.M.B. Vitányi, *J. Comput. Syst. Sci.* 73(2007), 973–985].

The proof of Lemma 6.12.1 in Section 6.12 is based on a paper by L. Fortnow and S. Laplante [*Inform. Comput.*, 123(1995), 121–126], which in turn was based on a proof by A. Razborov [pp. 344–386 in *Feasible Mathematics II*, P. Clote, J. Remmel, eds., 1995]. This lemma was originally proved by J. Håstad [pp. 143–170 in *Randomness and Computation*, S. Micali, ed., JAI Press, 1989] for the purpose of simplifying and improving Yao's lower bound on unbounded circuits [A.C.C. Yao, *Proc. 26th IEEE Symp. Found. Comput. Sci.*, 1985, pp. 1–10]. Note that in Lemma 6.12.1, in order to simplify the proof, we have $\alpha = 12tl/n$ instead of Håstad's $\alpha = 5tl/n$ or Fortnow and Laplante's $\alpha = 5.44tl/n$. See also [M. Agrawal, E. Allender, and S. Rudich, *J. Comput. Syst. Sci.*, 57:2(1998), 127–143] for more circuit lower bounds by incompressibility.

There are applications of the incompressibility method and Kolmogorov complexity we do not cover. This is because there are by now simply too many of them. Also, some applications require lengthy discussions of computational models and preliminary facts; and some others are indirect applications. A.M. Ben-Amram and Z. Galil [*J. Assoc. Comp. Mach.*, 39:3(1992), 617–648] use Kolmogorov complexity to formalize the concept of incompressibility for general data types and prove a general lower bound for incompressible data types. J. Shallit and his coauthors in a series of papers study a variation of descriptional complexity, 'automaticity,' where the description device is restricted to finite automata; see [J. Shallit and Y. Breitbart, *Proc. 11th Symp. Theoret. Aspects Comput. Sci.*, 1994, pp. 619–630 and *J. Comput. System Sci.* 53:1(1996), 10–25]. U. Vazirani and V. Vazirani [*Theoret. Comput. Sci.*, 24(1983), 291–300] studied probabilistic polynomial-time reductions. It is possible to do their reduction by Kolmogorov complexity. Kolmogorov complexity has also been studied in relation to the tradeoff of table size and number of probes in hashing by H.G. Mairson [*Proc. 24th IEEE Found. Comput. Sci.*, 1983, pp. 40–47]. See also [K. Mehlhorn, *Proc. 23rd IEEE Found. Comput. Sci.*, 1982, 170–175]. D. Hammer and A.K. Shen [A strange application of Kolmogorov complexity, *Theor. Comput. Syst.*, 31:1(1998), 1–4] use complexity to derive a geometric relation, and a geometric relation to derive a property of complexity. Namely, from $2C(a,b,c) \leq C(a,b) + C(b,c) + C(c,a) + O(\log n)$ one can derive $\|V\|^2 \leq \|S_{xy}\| \cdot \|S_{yz}\| \cdot \|S_{zx}\|$. Here V is a set in three-dimensional space, S_{xy}, S_{yz}, S_{zx} are its two-dimensional projections, and $\|\cdot\|$ denotes volume. Moreover, from the well-known Cauchy–Schwarz inequality one can derive $2K(a,b,c) \leq K(a,b) + K(b,c) + K(c,a) + O(1)$. The incompressibility method has been applied to logical definability by M. Zimand [*Inform. Process. Lett.*, 57(1996), 59–64] and to finite-model theory and database query languages by J. Tyszkiewicz [*Inf. Comput.*, 135:2(1997), 113-135; *Proc. 8th Int. Conf. Database Theory*, *Lect. Notes Comput. Sci.*, Vol. 893, Springer-Verlag, 1995, pp. 97–110]. M. Zimand [*Ibid.*] studies a 'high-low Kolmogorov complexity law' equivalent to a 0-1 law in logic. See also [R. Book, *SIAM J. Comput.*, 23(1994), 1275–1282]. K.W. Regan [*Proc. 10th IEEE Conf. Structure in Complexity Theory*, 1995, pp. 50–64] uses Kolmorogov complexity to prove superlinear lower bounds for some problems in a type of hierarchical memory model that charges higher cost for nonlocal communication.

7

Resource-Bounded Complexity

Recursion theory has a resource-bounded version in computational complexity theory. Similarly, Kolmogorov complexity has resource-bounded Kolmogorov complexity. Several authors suggested early on the possibility of restricting the power of the device used to (de)compress strings. Says Kolmogorov in 1965:

"The concept discussed ... does not allow for the 'difficulty' of preparing a program p for passing from an object x to an object y. [... some] object permitting a very simple program, i.e., with very small complexity $K(x)$ can be restored by short programs only as the result of computations of a thoroughly unreal nature. [... this concerns] the relationship between the necessary complexity of a program and its permissible difficulty t. The complexity $K(x)$ that was obtained [before] is, in this case, the minimum of $K^t(x)$ on the removal of the constraints on t." [Kolmogorov]

Additional restrictions on resource bounds yield a rich mathematical theory and abundant applications. Many statements that used to be trivial or easily provable become nontrivial or very difficult questions with resource bounds. Even the definition of Kolmogorov complexity itself becomes nonstandard.

If we allow unlimited computational resources such as time and space, it does not matter whether we define the complexity of x as the length of the shortest program that *prints* x, or that *accepts only* x; both definitions turn out to be equivalent. However, it is not known whether the two definitions are still equivalent under polynomial-time restriction of the computational processes involved. We will specifically study two parameters of the 'permissible difficulty' of Kolmogorov, the *time* and *space* complexity.

M. Li and P.M.B. Vitányi, *An Introduction to Kolmogorov Complexity and Its Applications*,
DOI: 10.1007/978-0-387-49820-1_7, © Springer Science + Business Media, LLC 2008

7.1 Mathematical Theory

Consider Turing machines with a separate read-only input tape, a fixed number of work tapes on which the computation takes place, and a write-only output tape. The input is a finite binary string. Initially, it is placed on the input tape, delimited by distinguished end markers. A machine with k work tapes is called a *k-tape Turing machine*. Machines with $k \geq 1$ are generically called *multitape Turing machines*.

Let $\phi_1, \phi_2, \ldots$ be an effective enumeration of partial recursive functions. Let T_ϕ be a multitape Turing machine that computes ϕ. Let $n = l(x)$. If $T_\phi(y) = x$ in $t(n)$ steps (time) and $s(n)$ tape cells (space), then we also write $\phi^{t,s}(y) = x$. We identify the natural numbers $\mathcal{N}$ and the set of finite binary sequences $\{0,1\}^*$ as in Equation 1.3, page 12.

Definition 7.1.1 Let $x, y, p \in \mathcal{N}$. Every recursive function ϕ together with p, y such that $\phi(p, y) = x$ is a *description* of x. The *resource-bounded Kolmogorov complexity* $C_\phi^{t,s}$ of x with respect to ϕ and *conditional* to y, is defined by

$$C_\phi^{t,s}(x|y) = \min\{l(p) : \phi^{t,s}(p, y) = x\},$$

and $C_\phi^{t,s}(x|y) = \infty$ if there is no such p. Define the *unconditional* resource-bounded Kolmogorov complexity of x as $C_\phi^{t,s}(x) = C_\phi^{t,s}(x|\epsilon)$.

Obviously, for total recursive functions $t(n)$ and $s(n)$ the function $C_\phi^{t,s}$ is total recursive as well. We prove an *invariance theorem* for the resource-bounded Kolmogorov complexity. As usual, this invariance condition is the basis of the theories and applications to follow. The price we pay for adding resource bounds is that the invariance property becomes considerably weaker.

Theorem 7.1.1 *There exists a universal partial recursive function ϕ_0 such that for every other partial recursive function ϕ, there is a constant c such that*

$$C_{\phi_0}^{ct \log t, cs}(x|y) \leq C_\phi^{t,s}(x|y) + c,$$

for all x and y. The constant c depends on ϕ but not on x and y.

Proof. Consider a standard enumeration of multitape Turing machines $T_1, T_2, \ldots$. Let ϕ_i denote the partial recursive function computed by T_i. Let $n = l(y)$. Let $\langle y, p\rangle = 1^{l(y)}0yp$ be the standard linear time and space invertible pairing bijection. Let ϕ_0 be the partial recursive function computed by a universal Turing machine U with two worktapes such that U simulates all T_i's according to the well-known simulation of F.C. Hennie and R. Stearns [*J. Assoc. Comp. Mach.*, 13(1966), 533–546]. It follows from that simulation that for each i there is a constant c such that $U(\langle y, \langle i, p\rangle\rangle) = T_i(y, p)$. Suppose that T_i, started on input $\langle y, p\rangle$, computes x in $t(n)$ time and $s(n)$ space. Then the computation of U,

started on input $\langle y, \langle i, p\rangle\rangle$, takes at most time $ct(n)\log t(n)$ and space $cs(n)$. That is,

$$\phi_0^{ct\log t, cs}(\langle y, \langle i, p\rangle\rangle) = \phi_i^{t,s}(y, p).$$

Choosing c large enough to also exceed the length of the encoding of T_i in the input for U, say $c \geq 2i + 1$, proves the theorem. □

The invariance theorem allows us to select a reference universal Turing machine computing ϕ_0 as above and drop the subscript ϕ_0 in $C_{\phi_0}^{t,s}$ and write $C^{t,s}$, provided the statement we make is not sensitive to the additive constant term in the complexity of each string, the multiplicative logarithmic factor in the time complexity, and the multiplicative constant factor in the space complexity.

Definition 7.1.2 Let x be a string of length n and U compute ϕ_0 of Theorem 7.1.1. The t-time-bounded and s-space-bounded version of $C(x|y)$ is defined by

$$C^{t,s}(x|y) = \min\{l(p) : U(\langle y, p\rangle) = x \text{ in } t(n) \text{ steps and } s(n) \text{ space}\}.$$

Resource-bounded Kolmogorov complexity behaves differently from the unbounded case. The complexity of string x can be defined as the length of either the shortest program that *generates* x or the shortest program that *accepts only* x. These definitions are clearly equivalent when there are no resource bounds. It is unknown whether they are the same, for example, for the polynomial-time-bounded versions. There may be a short program that accepts precisely x in time $t(n)$, but one may have trouble finding an equally short program that prints x in time $t(n)$. In Definition 7.1.1 we have defined the resource-bounded complexity in terms of the shortest program that generates x. We also require the other type.

Definition 7.1.3 A *predicate* is a 0–1-valued function.

Let $\psi_1, \psi_2, \ldots$ be an effective enumeration of partial recursive predicates. Let T_ψ be the multitape Turing machine that computes ψ. The computation $T_\psi(\langle x, \langle p, y\rangle\rangle)$ outputs 0 or 1. If T_ψ uses at most $t(n)$ time and $s(n)$ space on every x of length n, for all auxiliaries p, y, then we write the computed function as $\psi^{t,s}(x, p, y)$.

Definition 7.1.4 Let $x, y, p \in \mathcal{N}$. Let ψ be a partial recursive predicate. The *resource-bounded accepting complexity CD* of x with respect to ψ and *conditional* to y, is defined as

$$CD_\psi^{t,s}(x|y) = \min\{l(p) : \forall v, \psi^{t,s}(v, p, y) = 1 \text{ iff } v = x\},$$

and $CD^{t,s}_{\psi}(x|y) = \infty$ if there is no such p. If $y = \epsilon$, then $CD^{t,s}_{\psi}(x) = CD^{t,s}_{\psi}(x|\epsilon)$ is the *unconditional CD*-complexity of x. (In *CD* the *D* stands for 'decision.')

The statement and proof of the invariance theorem for *CD*-complexity are omitted. They are completely analogous to Theorem 7.1.1. In cases in which this is justified by the invariance theorem, we drop the index ψ in $CD^{t,s}_{\psi}$.

Therefore, we distinguish between $C^{t,s}(x)$ as the length of the shortest program generating x of length n in $t(n)$ time and $s(n)$ space, and $CD^{t,s}(x)$ as the length of the shortest program accepting precisely x in $t(n)$ time and $s(n)$ space.

Definition 7.1.5 Let $S \subseteq \mathcal{N}$. We define $C^{t,s}(x|S)$ or $CD^{t,s}(x|S)$ similarly to the above, except that conditional S now means that the underlying Turing machine is equipped with a distinguished *oracle tape*. In the course of its computation the Turing machine can write questions of the form "is $z \in S$?" on the oracle tape. After the question is written, which takes at least $l(z)$ steps, the oracle gives the correct answer "yes" or "no" in one step.

In the notation $C^{t,s}$ and $CD^{t,s}$, we always use the first parameter t for time and second parameter s for space. Considering only time complexity, we write C^t and CD^t; considering only space complexity, we write C^s and CD^s. When neither t nor s is required, that is, when $t = \infty$ and $s = \infty$, then $C^{t,s}$ and $CD^{t,s}$ both converge to the original plain Kolmogorov complexity C.

Definition 7.1.6 Assume the notation above. We define the following descriptional complexity classes:

$$C_{\phi}[f(n), t(n), s(n)|y] \stackrel{\text{def}}{=} \{x : C^{t,s}_{\phi}(x|y) \leq f(n),\ n = l(x)\},$$
$$CD_{\psi}[f(n), t(n), s(n)|y] \stackrel{\text{def}}{=} \{x : CD^{t,s}_{\psi}(x|y) \leq f(n),\ n = l(x)\},$$

where the subscripts ϕ and ψ are dropped in case they are superfluous. If $y = \epsilon$, then we also drop the conditional $|\epsilon$.

Theorem 7.1.2 *Let p, q be polynomial-time bounds, and let A be an* NP*-complete set.*

(i) *For every p, there exist q such that $CD^q(x) \leq C^p(x) + O(1)$.*

(ii) *For every p, there exist q such that $C^q(x|A) \leq CD^p(x) + O(1)$.*

Proof. Item (i) is immediate. To prove Item (ii), use the NP oracle repeatedly to determine the successive bits of $x = x_1 \dots x_n$ as follows: Let T be a Turing machine that accepts only x and runs in time $p(n)$. Define a Turing machine T^A that generates x using an appropriate oracle A. Assume that T^A has already determined $x_1 \dots x_i$. At the next step, T^A asks the oracle A whether T accepts a string with prefix $x_1 \dots x_i 0$. If the answer is "yes," then T^A sets $x_{i+1} := 0$; otherwise it sets $x_{i+1} := 1$. The oracle can answer these questions, since the set

$$\{\langle T, y, 1^t, 1^n \rangle : \ T \text{ accepts } yz \text{ in time } t \text{ for some } z \text{ with } l(yz) = n\}$$

is in NP. The oracle machine generates all of x in this way, taking time at most polynomial in $p(n)$. □

The fact that it is not known how to improve Theorem 7.1.2 may suggest that $C^{t,s}$ and $CD^{t,s}$ are different measures, at least for polynomial-time bounds. This difference disappears, however, when we have enough time available. Namely, in exponential time the machine can search through all strings of length n.

Example 7.1.1 **(Resource-bounded prefix complexity)** The definition and the proof of the invariance theorem, Theorem 2.1.1 on page 117, make the *exact* plain Kolmogorov complexity of a string an intrinsic property of the string itself. For the resource-bounded version of the plain Kolmogorov complexity, we do not have an exact invariance theorem, but an *efficient* resource-bounded invariance theorem, Theorem 7.1.1. The resource-bounded prefix complexities $K^{t,s}$ and $KD^{t,s}$ can be defined similarly to $C^{t,s}$ and $CD^{t,s}$. However, in Chapter 3 there are two definitions given. The first is in terms of the standard enumeration of Turing machines, each of which is modified so that the machines in the modified enumeration compute all and only partial recursive prefix functions, using the algorithm of Definition 3.1.1 on page 200. This type of prefix machine we called 'partially recursive prefix function machines.' The second definition, Example 3.1.1 on page 201, is in terms of a new model of Turing machine with a separate one-way read-only input tape. This type of prefix machine we have called 'self-delimiting machines.' In the absence of a resource bound, the prefix complexities resulting from the two definitions coincide, and this single prefix complexity satisfies the invariance theorem, Theorem 3.1.1 on page 212. This makes the prefix complexity an intrinsic property of the individual string just as in the plain Kolmogorov complexity case. In the presence of time bounds the situation is not so simple. To prove the equivalent of Theorem 7.1.1 for self-delimiting machines is straightforward: There is an optimal self-delimiting machine, in the sense of Theorem 3.1.1, that simulates every self-delimiting machine in the sense of Theorem 7.1.1 such that for every

self-delimiting machine T there is a constant c such that

$$K^{c\cdot t\log t}(x) \leq K^t_T(x) + c, \tag{7.1}$$

for every string x and number of steps t. If we want to prove the equivalent of Theorem 7.1.1 for time-bounded partial recursive prefix functions, we first must convert each partial recursive function (rather, the Turing machine computing it) to a prefix function (rather, a prefix machine computing it) using the algorithm of Definition 3.1.2. This construction may dramatically increase the running time of the prefix machine over that of the original machine from which it derived. Hence, this approach for partial recursive prefix functions does not give an efficient resource-bounded invariance theorem. We may try to do so indirectly by simulating a partial recursive prefix machine by a self-delimiting machine and simulating the latter by a reference universal self-delimiting machine. Thus, the question arises whether one can show that every partially recursive prefix function can be efficiently simulated by a self-delimiting machine. In [D.W. Juedes, J.H. Lutz, *Theor. Comput. Systems*, 33(2000), 111–123] this question is answered in the negative: such an efficient simulation exists iff P = NP. Even though the self-delimiting machines compute exactly the partial recursive prefix functions, they are not efficient. There is such a function that cannot be computed efficiently by a self-delimiting machine unless P = NP. Nonetheless, we can show the following. Define K based on self-delimiting machines and $\hat{K}$ based on partial recursive prefix-function machines. For every partial recursive prefix machine M, there is a constant c such that

$$K^{c\cdot t(\log t)^2}(x) \leq \hat{K}^t_M(x) + O(\log l(x)), \tag{7.2}$$

for every string x and number of steps t. To see this, observe that by Theorem 7.1.1, for every partial recursive prefix machine M, there is a constant c_1 such that

$$C^{c_1 t\log t}(x) \leq C^t_M(x) + c_1 = \hat{K}^t_M(x) + c_1,$$

for every x and t. Also, by Equation 7.1,

$$K^{c_2 t\log t}(x) \leq C^t(x) + 2\log C^t(x) + c_2,$$

for some c_2, and every x and t. Combining the two observations and observing that $\log C^t(x) \leq \log l(x) + c_3$ for some c_3 and all x and t gives Equation 7.2. This is attributed to H.M. Buhrman in [D.W. Juedes, J.H. Lutz, *Ibid.*]. See also Exercise 7.1.4 on page 546. ◇

7.1.1 Computable Majorants

Here, consider only time-bounded plain Kolmogorov complexity and write C^t or CD^t. Similar results to those obtained here can be derived

for space-bounded complexities. Upper semicomputable functions (below) are the functions that can be approximated from above by recursive functions, Section 4.1. We repeat the definition in Example 4.1.1.

Definition 7.1.7 A *majorant* of Kolmogorov complexity is a upper semicomputable function ϕ such that $C(x) < \phi(x)+c$ for all x, where c is a constant depending on ϕ but not on x.

Example 7.1.2 Let t be a total recursive function. Then the function C^t is a total recursive majorant of C. We first prove that C^t is a majorant. For every x, we can run the universal machine on all programs of length at most $l(x) + c$ for $t(l(x))$ steps each. Then $C^t(x)$ is the length of the shortest program that halts with output x. Clearly, $C^t(x) \geq C(x) - O(1)$. This also shows that C^t is total recursive. ◇

If C^t is a total recursive majorant, then $C(x)$ and $C^t(x)$ may be exponentially different. That is, if we impose a recursive time limit, the length of the shortest description of some strings can sharply increase. We prove this fact in terms of characteristic sequences. Let A be a set, and $\chi = \chi_1\chi_2\ldots$ its characteristic sequence defined by $\chi_i = 1$ if $x_i \in A$ and $\chi_i = 0$ otherwise. By Barzdins's lemma, Theorem 2.7.2 on page 178, if A is recursively enumerable, then $C(\chi_{1:n}|n) \leq \log n + O(1)$. The following *blowup* theorem is one of the very first results in time-limited Kolmogorov complexity.

Theorem 7.1.3 *There is a recursively enumerable set A with characteristic sequence χ such that for all total recursive t and all n we have $C^t(\chi_{1:n}|n) \geq c_t n$, where $0 < c_t < 1$ is a constant independent of n (but dependent on t).*

Proof. Let $\mathbf{T} = T_1, T_2, \ldots$ be a standard enumeration of Turing machines. Let $\mathbf{M} = M_1, M_2, \ldots$ be another enumeration of Turing machines such that $T_k = M_i$ with $i = 2^{k-1} + j2^k$, for all $k \geq 1$ and $j \geq 0$. Stated differently, k is the largest integer such that 2^{k-1} divides i. That is, in the following sequence, if k is the index in position i, then M_i is T_k:

$$12131214121312151213121412131216 12\ldots .$$

Thus, T_1 occurs every two machines in the list, T_2 appears every four machines in the list, and T_k occurs every 2^k machines in the list.

Given a total recursive time bound t, there is some T_k that computes the value $t(n)$ from input n. We know that T_k occurs with intervals of 2^k consecutive machines in the $\mathbf{M}$ list, starting with the $(3 \cdot 2^{k-1})$th machine.

The following process enumerates an infinite sequence $\chi = \chi_1\chi_2\ldots$ by diagonalization. Later we show that there is a constant $c_t > 0$ such that $C^t(\chi_{1:n}|n) \geq c_t n$.

Initialize. Set $i = 1$ and $\chi_1 := 0$.

For all $i > 1$ dovetail the following computations: Set $n := 2^{i-1}$. {This step will enumerate the segment $\chi_{n+1:2n}$}

Set $n' := 2^{2^k} n$, where k is the index of M_i on the **T** list. This means that $M_i = T_k$ computes the partial recursive function, say, t. Simulate the computation of $M_i(n')$. If $M_i(n')$ terminates, then its output is time bound $t(n')$.

Simulate all binary programs of size up to $n-1$ for $t(n')$ steps on the reference universal Turing machine. Choose $\chi_{n+1:2n}$ such that it is different from the segment from position $n+1$ to $2n$ of the output of any of these simulations. Since there are only $2^n - 1$ programs of size at most $n-1$, and there are 2^n different candidates for $\chi_{n+1:2n}$, this is always possible. If $M_i(n')$ does not terminate, then define $\chi_{n+1:2n} := 0^n$.

The above procedure does not give an effective construction for χ. If $M_i(n')$ does not terminate, then we have no value of $t(n')$ to use; hence, we cannot decide that $\chi_{n+1:2n} = 0^n$ for $n = 2^{i-1}$.

Claim 7.1.1 Let $A \subseteq \mathcal{N}$ be defined by $x \in A$ iff $\chi_x = 1$. Then A is recursively enumerable.

Proof. For all x, simultaneously dovetail the following computation: First, determine i such that $n + 1 < x \leq 2n$ for $n = 2^{i-1}$. Enumerate $\chi_{n+1:2n}$ by simulating M_i. If $M_i(n')$ never terminates, then $\chi_{n+1:2n} = 0^n$. That is, both $x \notin A$, and our simulation of $M_i(n')$ doesn't halt. If $M_i(n')$ terminates, then we effectively construct $\chi_{n+1:2n}$. We enumerate x as an element of A iff $\chi_x = 1$. □

Claim 7.1.2 Let χ be as above. It is incompressible as stated in the theorem, where without loss of generality we take t to be a monotonic increasing total recursive function.

Proof. Let t be some monotonic increasing total recursive function. Then t is computed by some machine T_k. By enumeration of **M**, the machine T_k occurs for the first time as M_{k_0} with $k_0 = 2^{k-1}$. Subsequently, T_k returns in the list in fixed intervals of 2^k machines. Let i run over the infinite sequence of values

$$k_0, k_0 + 2^k, k_0 + 2 \times 2^k, k_0 + 3 \times 2^k, \ldots .$$

Consider the special sequence of values of n defined by $n = 2^{i-1}$. Denote this sequence by S. Since t is total, the machine T_k halts for all inputs.

Enumerating χ, for every $n \in S$ we used a copy of T_k for a terminating computation of segment $\chi_{n+1:2n}$. This segment was determined differently from all corresponding segments of outputs of programs of length at most $n-1$ and halting within time $t(n')$ on the universal Turing machine.

For every large enough $n \in S$, there is an $m \in S$ such that $n/2^{2^k} \ll m \leq \frac{1}{2}n$ and no program of length at most $m-1$ outputs an initial segment $\chi_{1:2m}$ in $t(m') = t(2^{2^k}m) \gg t(n)$ steps. (This is why we need to use $t(n')$ rather than $t(n)$ in the definition of χ.) Therefore,

$$C^{t(m')}(\chi_{1:2m}) \geq m. \tag{7.3}$$

By way of contradiction, assume that

$$C^{t(n)}(\chi_{1:n}|n) \leq n/2^{2^k+1}.$$

Then, given n, we can compute $\chi_{1:n}$ in time $t(n) \ll t(m')$ from a program of length at most $\frac{1}{2}m$. Subsequently, given an additional description of m in $O(\log m)$ bits, we can output the prefix $\chi_{1:2m}$ in linear time. Since we have altogether used fewer than $t(m')$ steps, and a program of length less than m, we contradict Equation 7.3. □

Set the constant $c_t := 1/2^{2^k+1}$ with k an index of a Turing machine computing t. The two claims prove the theorem. □

This lower bound is approximately optimal. Namely, let A be a recursively enumerable set. For the characteristic sequence χ of A and each constant $c > 0$, there exists a total recursive function t such that for infinitely many n, $C^t(\chi_{1:n}|n) \leq cn$. This result, as well as Theorem 7.1.3, and Barzdins's lemma, Theorem 2.7.2 on page 178, are due to J.M. Barzdins [*Soviet Math. Dokl.*, 9(1968), 1251-1254]. See also Exercise 7.1.6 on page 546.

The theorem can be generalized to computable majorants. There is a recursively enumerable set A, with characteristic sequence χ, such that for each computable majorant ϕ and each n, we have $\phi(\chi_{1:n}|n) \geq c_\phi n$, with c_ϕ a constant independent of n (but dependent on ϕ) (Exercise 7.1.5 on page 546).

If a set A is recursive, then its characteristic sequence χ is also recursive, that is, there is a program that outputs $\chi_{1:n}$ on input n. By Exercise 2.3.4 on page 131, the following statements are equivalent:

- χ is an infinite recursive sequence.
- There exists a constant c such that for all n, we have $C(\chi_{1:n}|n) \leq c$.
- There exists a constant c such that for all n, we have $C(\chi_{1:n}) \leq \log n + c$.

Adding a recursive time bound, we obtain a blowup theorem again. It is stated for C^t, but the same proof also works for K^t.

Theorem 7.1.4 *Let f and t be unbounded total recursive functions. There is a recursive sequence χ such that $C^t(\chi_{1:n}|n) \geq n - f(n)$ infinitely often.*

Proof. Without loss of generality, let $f(n) \leq n$ and $\lim_{n\to\infty} f(n) = \infty$. The sequence χ will be constructed by diagonalization. Define a function g inductively by $g(1) = 1$, and $g(n) = \min\{m : f(m) > g(n-1)\}$ for $n > 1$. The function g is total recursive, nondecreasing, and unbounded.

Initialize. Set $\chi_1 := 0$.

For $n := 2, 3, \ldots$ **do** Compute $\chi_{g(n-1)+1:g(n)}$ as follows: Simulate all programs of size less than $g(n) - g(n-1)$, each for $t(g(n))$ steps. Extend $\chi_{1:g(n-1)}$ to $\chi_{1:g(n)}$ so that $\chi_{1:g(n)}$ is not the initial segment of an output of any of the above simulations. **{**Since there are at most $2^{g(n)-g(n-1)} - 1$ programs of length less than $g(n) - g(n-1)$, extending χ in this way is always possible**}**

Thus, by construction, for almost all n we have $C^t(\chi_{1:g(n)}|g(n)) \geq g(n) - g(n-1)$. By definition of $g(n)$, we have $f(g(n)) \geq g(n-1)$. Then for infinitely many n, we have $C^t(\chi_{1:n}|n) \geq n - f(n)$. □

In Theorem 2.5.5, page 153, it was stated that almost all infinite sequences have maximal Kolmogorov complexity. More precisely, with probability one in the sense of the uniform measure, for every infinite sequence χ there is a constant c such that

$$C(\chi_{1:n}|n) \geq n - c$$

infinitely often. The above theorem shows that every recursive time bound blows up the Kolmogorov complexity of some recursive sequences to nearly maximal. But, similar to the complexity oscillations in pure Kolmogorov complexity for random sequences (Theorems 2.5.1 and 2.5.4, page 143 and page 152, respectively), recursive sequences cannot achieve maximal Kolmogorov complexity, even when they are restricted by a linear recursive time bound. In this sense, Theorem 7.1.4 is optimal.

Theorem 7.1.5 *Let $t(n) = \Omega(n)$ be an unbounded total recursive function and let χ be a recursive sequence. There is an unbounded total function f such that for all n, we have*

$$C^t(\chi_{1:n}|n) \leq n - f(n).$$

Proof. Since χ is recursive, there is a Turing machine T that outputs χ_n on input n. We can trivially design a short program q for the universal Turing machine U such that $U(\langle q, n\rangle) = \chi_{1:n}$. The computation of U on input $\langle q, n\rangle$ simulates T on input 0 until T halts, then on input 1 until T halts, and so on, for a *total* of n steps. Let m be the last input for which T halted in this simulation. Then U has already obtained $\chi_{1:m}$. Therefore, q needs only to additionally store $\chi_{m+1:n}$ of length $n-m+1$ to be able to reconstruct $\chi_{1:n}$ totally. This means that $l(q) = n - m + c$ for some constant c (the index of T). Moreover, the computation of $\chi_{1:n}$ by U runs in linear time. Since $m \to \infty$ as $n \to \infty$, the theorem follows. □

An infinite sequence χ may have maximal Kolmogorov complexity in the sense that for some c, for all n,

$$K(\chi_{1:n}|n) \geq n - c,$$

where K is the prefix complexity; see Theorem 3.9.1 on page 247. But Theorem 7.1.5 and its proof also hold for prefix complexity. Hence, the displayed equation doesn't hold for all n with c a constant, χ a recursive infinite sequence, and K replaced by K^t with t a linear recursive time bound.

Example 7.1.3 We have considered sequences that are incompressible with respect to time-limited Kolmogorov complexity. Sequences that are incompressible with respect to pure Kolmogorov complexity are shown to be random in Theorem 2.5.5, page 153, and Theorem 3.6.1, page 222. In Section 2.5, sequential (Martin-Löf) tests were used to define randomness of a sequence. An infinite sequence is random iff it can withstand all sequential tests (Definition 2.5.2). A sequential test is defined by a recursive function (Definition 2.5.1).

Random numbers cannot be generated by arithmetic means. In many applications, such as cryptography, it is sufficient to generate numbers that appear to be random if we do not inspect them too closely. Obviously, the notion of sequential tests can be scaled down by permitting only time-bounded (or space-bounded) tests. In particular, one is often interested in sequences that are just random enough to pass sequential tests that run in polynomial time.

Let $\{0,1\}^\infty$ be the set of infinite binary sequences with the uniform (Lebesgue) measure. This measure assigns probability $2^{-l(x)}$ to the set of infinite sequences starting with x.

Definition 7.1.8 A sequential test δ is called a *sequential ptime* (*pspace*) *test* if δ is polynomial-time (polynomial-space) computable. An infinite sequence ω is *ptime* (*pspace*) *pseudorandom* if for every sequential ptime (pspace) test δ, and every polynomial p, it satisfies $\delta(\omega_{1:p(n)}) < n$ for all but finitely many n.

The condition on ω that $\delta(\omega_{1:p(n)}) < n$ guarantees that no evidence of nonrandomness of ω at significance level 2^{-n} can be found by δ unless an initial segment of length greater than $p(n)$ is examined. For the terminology of testing, consult Sections 2.4 and 2.5. Contrary to the case of unbounded sequential tests, it appears to be the case that there is no universal ptime sequential test. This is because a universal sequential ptime test must simulate all other sequential ptime tests, and it is unknown how to make such a universal test run in polynomial time. By diagonalization, on the other hand, one can show that there exist pspace pseudorandom sequences that are double-exponential-space computable. We give a criterion in terms of space-bounded complexity for a sequence to be pspace random.

Lemma 7.1.1 *Let C^s denote the s-space bounded generating complexity and* $\mathrm{poly}(f(n))$ *mean polynomial in $f(n)$. If $C^q(\omega_{1:n}) > n - \mathrm{poly}(\log n)$ for all polynomials q and all but finitely many n, then ω is pspace pseudorandom.*

Every pspace pseudorandom sequence is also a ptime pseudorandom sequence, simply because a ptime-sequential test cannot use more than polynomial space. See also (Exercise 7.1.11 on page 548; Ker-I Ko, *Theoret. Comput. Sci.*, 48(1986), 9–33). ◇

7.1.2 Resource-Bounded Hierarchies

Given a fixed reference universal machine, the three parameters in

$$C[f(n), t(n), s(n)] = \{x : C^{t,s}(x) \leq f(n),\ n = l(x)\}$$

characterize classes of strings of different resource-bounded Kolmogorov complexity. A shortest program p of length $C^{t,s}(x)$, from which we can compute x with $l(x) = n$ in $t(n)$ time and $s(n)$ space simultaneously, is called a *seed*. Intuitively, with a longer seed, more time, or more space, we should be able to obtain some string that is not obtainable with these parameters. Conversely, with a shorter seed, less time, or less space, we may not be able to obtain strings that are obtainable with the current parameters.

As usual, everything needs to be related to one fixed reference universal Turing machine. For very reasonable f, s, n, x with $l(x) = n$, there is always a universal Turing machine U such that $x \in C_U[f(n), n, s(n)]$. This is similar to the case of nonresource-bounded Kolmogorov complexity.

The linear-space and linear-time speedup theorems, which were useful tools in proving Turing machine space and time hierarchies, do not apply here because compressing time or space results in the increase of program length.

Theorem 7.1.6 *Let f and g be unbounded total recursive functions such that $f(n) + g(n) \leq n$, and for every k we can compute the least n such that $f(n) = k$ in space $s(n) \geq \log n$ and time $t(n) - n$. Then for large n,*

$$C[f(n) + g(n), t(n), s(n)] - C[f(n), \infty, \infty] \neq \varnothing.$$

Proof. It suffices to prove that for the fixed reference universal Turing machine U, there is a constant c depending only on U such that

$$C[f(n) + c, t(n), s(n)] - C[f(n), \infty, \infty] \neq \varnothing.$$

Now let n and constant c be large enough that the following formulas are true. Fix a string x of length $l(x) = f(n) + \frac{1}{2}c$ such that $C(x) \geq l(x)$. We will show that

$$x0^{n-l(x)} \in C[f(n) + c, t(n), s(n)] - C[f(n), \infty, \infty].$$

Note that $l(x0^{n-l(x)}) = n$. Let $p = qx$ be a program that computes $x0^{n-l(x)}$ as follows:

- Compute the first n such that $f(n) + c \geq l(p)$ in $t(n) - n$ time and $s(n)$ space.
- Print x followed by $0^{n-l(x)}$.

The length of the program is $l(p) \leq f(n) + c$. Hence, $C^{t,s}(x0^{n-l(x)}) \leq f(n) + c$. Therefore, $x0^{n-l(x)} \in C[f(n) + c, t(n), s(n)]$.

We show that $x0^{n-l(x)} \notin C[f(n), \infty, \infty]$. Suppose the contrary. But then we can reconstruct x as follows: Generate $x0^{n-l(x)}$ using $f(n)$ bits. In order to retrieve x, we need to remove the suffix of length $n - l(x)$. We know that $l(x) := f(n) + \frac{1}{2}c$. The length of a self-delimiting program to compute c is at most $2 \log c$. We know $f(n)$ because we remembered this value as the length of the initial program. We can compute n from $f(n)$ in some constant d bits by the assumption in the theorem. Delete the suffix $0^{n-l(x)}$ to retrieve x. Altogether, this description of x takes $f(n) + 2 \log c + d$ bits. For c large enough we obtain $C(x) < f(n) + \frac{1}{2}c = l(x)$, a contradiction. □

Example 7.1.4 Let $f(n) = \sqrt{n}$, which is computable in $O(n)$ time, and $g(n) = \log \log n$, which is also computable in $O(n)$ time. By the above theorem, for the reference universal Turing machine U, for n large enough,

$$C_U[\sqrt{n} + \log \log n, cn, \infty] - C_U[\sqrt{n}, \infty, \infty] \neq \varnothing.$$

Since $C_U[\sqrt{n}, cn, \infty]$ is by definition contained in both these classes, its containment in $C_U[\sqrt{n} + \log \log n, cn, \infty]$ is proper. ◇

Theorem 7.1.7 *Let $s'(n) \geq 2n + s(n) + c$ and let $s(n)$ be nondecreasing and computable in space $s'(n)$. Let $f(n) \leq n$ be an unbounded nondecreasing function computable in space $s'(n) - \log n$. For n large enough, we have*

$$C[f(n), \infty, s'(n)] - C[n-1, \infty, s(n)] \neq \varnothing.$$

Proof. The idea is to find the first string not in $C[n-1, \infty, s(n)]$. The following program p for the reference universal Turing machine U prints a string $x \in C[f(n), \infty, s'(n)] - C[n-1, \infty, s(n)]$. Program p first computes the first n such that $f(n) > l(p)$, using $s'(n)$ space, including $\log n$ space to count to n. Then p marks off $s(n)$ tape cells. Subsequently, p simulates all $s(n)$ space-bounded computations of U, with inputs of sizes up to $n-1$. If x is the lexicographically first string of length n not generated by any of these $s(n)$ space-bounded computations, then p prints x. We can find this x because we can simply repeat all $s(n)$ space-bounded computations for each next candidate in the lexicographic enumeration of all strings of length n. Since there are 2^n candidates, and only $2^n - 1$ inputs of sizes up to $n-1$, such an x exists. With input p, reference machine U uses $s'(n) \geq 2n + s(n) + c$ space. Here c is a constant, n tape cells are needed to generate all strings of length n, another n tape cells are needed to generate all inputs for the simulated computation, and $s(n)$ space is needed to carry out the simulation.

Program p has length less than $f(n)$. It uses $s'(n)$ space and generates a string x not in $C[n-1, \infty, s(n)]$. □

Example 7.1.5 Theorems 7.1.6 and 7.1.7 are still true with C replaced by CD everywhere. For example, for large enough n, we have $CD[\log n, \infty, n^2] \subset CD[\log n, \infty, n^2 \log n]$ and $CD[\log n, \infty, n^2] \subset CD[2 \log n, \infty, n^2]$. ◇

A straightforward generalization to time-bounded Kolmogorov complexity leaves an exponential gap between two time classes. We leave this to the Exercises section. The nontrivial time counterpart of the above theorem is still an open problem. The above proof method fails, since time is not reusable. The time- or space-constructibility assumptions in Theorems 7.1.6 and 7.1.7 are necessary. Indeed, as in computational complexity theory, we can prove gap theorems.

Theorem 7.1.8 *Given a total recursive function $g(n) \geq n$ there exists a total recursive function $s(n)$ such that for every $f(n) < n$, for large enough n,*

$$C[f(n), \infty, s(n)] = C[f(n), \infty, g(s(n))].$$

Proof. Consider the reference universal Turing machine U. Given n, we define $s(n)$ to be the first i such that no program of size up to $n-1$

(starting with empty input) halts using between $i+1$ and $g(i)$ space on U. This $s(n)$ can always be recursively found because there are only finitely many programs of size up to $n-1$ and we can simulate them all recursively in dovetailing style.

Let $x \in C[f(n), \infty, g(s(n))]$ and let p be a program of length $f(n)$ printing x using at most $g(s(n))$ space. Because $f(n) < n$ and from our construction, we know that p does not use an amount of space between $s(n)+1$ and $g(s(n))$. Therefore, p uses at most $s(n)$ space, and hence $x \in C[f(n), \infty, s(n)]$. □

Let Q be a collection of subsets in $\{0,1\}^*$. A set is *Q-immune* if it is infinite and does not have infinite subsets belonging to Q. It is easy to see that for every unbounded total recursive $f(n) < n$, the set $\{0,1\}^* - C[f(n), \infty, \infty]$ is recursively enumerable-immune. For suppose $\{0,1\}^* - C[f(n), \infty, \infty]$ contains an infinite recursively enumerable subset accepted by a Turing machine T. Then we can construct the following machine T': The machine T' enumerates the subset by simulating T. While enumerating, it eventually will find a string x such that $f(l(x)) > l(T')$. Since x is enumerated by a program T' that is shorter than $f(n)$ with $n = l(x)$, we have $x \in C[f(n), \infty, \infty]$. But the subset enumerated by T' was in the complement of $C[f(n), \infty, \infty]$, which gives the required contradiction. We extend this idea in the following example.

Example 7.1.6 Let $\lim_{n\to\infty} s(n)/s'(n) \to 0$. Let U be the reference universal Turing machine. Let $s'(n) \geq n$ be a nondecreasing function. Let $f(n)$ be an unbounded nondecreasing function computable in space $s(n)$ by U. We prove that $\{0,1\}^* - C[f(n), \infty, s'(n)]$ is DSPACE$[s(n)]$-immune.

Assume, by way of contradiction, that $\{0,1\}^* - C[f(n), \infty, s'(n)]$ contains an infinite subset $A \in$ DSPACE$[s(n)]$. Let T_A be a Turing machine that accepts A using $O(s(n))$ space. We exhibit a program p for the reference universal Turing machine U that prints a long string in A:

Step 1. Find the first i such that $f(i) > l(p)$.

Step 2. Find the first $x \in A$ **{**by simulating T_A**}** such that $l(x) \geq i$. Print x. **{**This step uses $O(s(n))$ space, where $n = l(x)$**}**

For n large enough, by choice of p, the reference universal Turing machine prints x in space $s'(n)$ with the program p, where $l(p)$ is at most $f(n)$. Therefore, $x \in C[f(n), \infty, s'(n)]$, a contradiction. ◇

The sets $C[c \log n, t, s]$ will be especially useful in certain applications. Such sets belong to the wider class of sparse sets. A set is sparse if it has only a polynomial number of elements for each length.

Exercises

7.1.1. [20] Define $K^{t,s}$ and $KD^{t,s}$ based on self-delimiting machines, Example 7.1.1, and prove the invariance theorems for them.

7.1.2. [20] Prove an invariance theorem for $CD^{t,s}$.

7.1.3. [25] Prove the results of Theorem 7.1.2 for KD^s, the space-bounded prefix complexity.

7.1.4. [O46] We resume the discussion in Example 7.1.1.

(a) Is there an efficient invariance theorem for prefix Kolmogorov complexity based on the partial recursive prefix function definition, Definition 3.1.2 on page200? Efficiency here means simulation of a machine in the enumeration of the partial recursive prefix functions by an optimal machine in, say, $O(t \log t)$ steps for t steps.

(b) Is there an efficient universal partially recursive function?

(c) Are the two notions of prefix Kolmogorov complexity efficiently equivalent?

Comments. Source: D.W. Juedes, J.H. Lutz, *Theor. Comput. Systems*, 33(2000), 111–123.

7.1.5. [39] Prove that there is a recursively enumerable set A with characteristic sequence χ such that for every total recursive majorant ϕ of C and every n, we have $\phi(\chi_{1:n}|n) \geq c_\phi n$, where c_ϕ is a constant independent of n (but dependent on ϕ).

Comments. Use the proof of Theorem 7.1.3 and also Example 4.1.1. Source: A.K. Zvonkin and L.A. Levin, *Russ. Math. Surveys*, 25:6(1970), 83–124], attributed to J.M. Barzdins and N.V. Petri. See also Example 4.1.1.

7.1.6. [35] Show that there is a recursively enumerable set A with characteristic sequence χ such that for all total recursive functions t, and for all $0 < c < 1$, there exist infinitely many n such that $C^t(\chi_{1:n}) > cn$.

Comments. Compare this exercise with Theorem 7.1.3. R.P. Daley [*Inform. Contr.*, 23(1973), 301–312] proved a more general form of this result using the uniform complexity of Exercise 2.3.2, page 130.

7.1.7. [40] Uniform complexity was defined in Exercise 2.3.2, page 130. Here we define the time-bounded uniform complexity. For an infinite string ω and the reference universal Turing machine U,

$$C^t(\omega_{1:n}; n) = \min\{l(p) : \forall i \leq n[U^t(p, i) = \omega_{1:i}]\}.$$

Define

$$CU[f, t, \infty] = \{\omega : \forall^\infty n[C^t(\omega_{1:n}; n) \leq f(n)]\}.$$

Let ω be a Mises–Wald–Church random sequence, with total recursive admissible place-selection rules instead of the standard definition with partial recursive admissible place-selection rules. Show that for every unbounded, nondecreasing, total recursive time bound t we have $\omega \notin CU[\frac{1}{2}n, t, \infty]$. This holds a fortiori for the set of Mises–Wald–Church random sequences using partial recursive place-selection rules.

Comments. This exercise shows that there is a Mises–Wald–Church random sequence, as defined in Definition 1.9 on page 53, with limiting frequency $\frac{1}{2}$, that has very low uniform Kolmogorov complexity, but high time-bounded uniform Kolmogorov complexity. In Exercise 2.5.13, Item (a), on page 161, we have proved that there is a Mises–Wald–Church random sequence ω, with partial recursive place-selection rules as in Definition 1.9 and limiting frequency $\frac{1}{2}$, such that $C(\omega_{1:n}; n) \leq f(n) \log n + O(1)$ for every unbounded, nondecreasing, total recursive function f. In Exercise 2.5.13, Item (c), on page 161, we have proved that there is a Mises–Wald–Church random sequence ω, with total recursive place-selection rules and limiting frequency $\frac{1}{2}$, such that $C(\omega_{1:n}; n) \leq f(n) + O(1)$ for every unbounded, nondecreasing, total recursive function f. Now we have shown that if there is a recursive time bound, then all Mises–Wald–Church random sequences (even with respect to only total recursive place-selection rules) indeed look quite random. One should be able to prove similar theorems for C and K versions of this exercise and Exercise 7.1.8. Source: R.P. Daley, *Inform. Contr.*, 23(1973), 301–312. The fact that some Mises–Wald–Church random sequences have low Kolmogorov complexity is from [R.P. Daley, *Math. Systems Theory*, 9(1975), 83–94].

7.1.8. [40] Use the uniform complexity of Exercise 7.1.7. We consider a time–information tradeoff theorem for resource-bounded uniform complexity. Let $f_i = \lceil n^{1/i} \rceil$, for $i = 1, 2, \ldots$. Construct a recursive infinite sequence ω and a set of total recursive, nondecreasing, unbounded functions $\{t_i\}$, where the t_i's are recursively enumerated as $t_1, t_2, \ldots$, such that the following hold:

(a) For all $i > 1$, we have $\omega \notin CU[f_i, t_i, \infty]$, where $CU[\cdot]$ is defined in Exercise 7.1.7.

(b) For all i, $\omega \in CU[f_i, t_{i+1}, \infty]$.

Comments. Source: R.P. Daley, *J. Assoc. Comp. Mach.*, 20:4(1973), 687–695. Daley proved a more general statement with a general characterization of $\{f_i\}$.

7.1.9. [29] Call χ polynomial-time computable if for some polynomial p and Turing machine T, the machine T on input n outputs $\chi_{1:n}$ in time $p(n)$. Prove that χ is polynomial-time computable if and only if for some

polynomial p and constant c for all n we have $C^p(\chi_{1:n}; n) \leq c$, where $C(\cdot;\cdot)$ is the uniform complexity of Exercise 7.1.7.

Comments. This is a polynomial-time version of the results about recursive infinite sequences in Exercise 2.3.4. Source: Ker-I Ko, *Theoret. Comput. Sci.*, 48(1986), 9–33.

7.1.10. [30] Let $f(n)$ be computable in polynomial time and satisfy the condition $\sum 2^{-f(n)} = \infty$. Then for every infinite sequence ω, there is a polynomial p such that for infinitely many n, we have $C^p(\omega_{1:n}|n) \leq n - f(n)$.

Comments. This is a polynomial-time version of Theorem 2.5.1 on page 143. Source: Ker-I Ko, *Theoret. Comput. Sci.*, 48(1986), 9–33.

7.1.11. [38] Use the terminology of Example 7.1.3. Prove that there exists a pspace-pseudorandom infinite sequence ω such that $\omega_{1:n}$ is computable in 2^{2^n} space.

Comments. Hint: prove this in two steps. First, prove that there is an infinite sequence ω, computable in double exponential space, such that for every polynomial p the p-space bounded *uniform* Kolmogorov complexity satisfies $C^{\infty,p}[\omega_{1:n}; n] \geq n - 3\log n$, for infinitely many n. Second, prove that every ω satisfying the above condition is pspace pseudorandom. Third, conclude that ω is also ptime pseudorandom. Source: Ker-I Ko, *Theoret. Comput. Sci.*, 48(1986), 9–33.

7.1.12. • [27/O48] In Section 3.9.1, we proved various versions of a symmetry of information theorem, with no resource bounds. For example, up to an $O(\log l(xy))$ additive term, $C(x, y) = C(x) + C(y|x)$. Here $C(x, y) = C(\langle x, y\rangle)$, where $\langle\cdot,\cdot\rangle$ is the standard pairing function.

(a) The symmetry of information holds for space-bounded complexity. Let $s(n) \geq n$ be a nondecreasing function and $f(n)$ be a nondecreasing function computable in $s(n)$ space. Show that for all x, y such that $l(y) = f(l(x))$, to within an additive term of $O(\log l(xy))$,

$$C^{\infty,O(s(n))}(x, y|l(xy)) = C^{\infty,O(s(n))}(x|l(x)) + C^{\infty,O(s(n))}(y|x).$$

(b) Use the assumptions in Item (a), except that now $f(n)$ is computable in exponential time. Let $E = \bigcup_c \mathrm{DTIME}(2^{cn})$. Show that to within an additive term of $O(\log l(xy))$,

$$C^{E,\infty}(x, y|l(xy)) = C^{E,\infty}(x|l(x)) + C^{E,\infty}(y|x).$$

(c) (Open) Use the assumptions of Item (a) except that now $f(n)$ is computable in nondecreasing polynomial time. Prove, or disprove,

$$C^{\mathrm{poly},\infty}(x, y|l(xy)) = C^{\mathrm{poly},\infty}(x|l(x)) + C^{\mathrm{poly},\infty}(y|x).$$

Comments. Here and in Exercise 7.1.13 we consider the question of *resource-bounded symmetry of information*. The main problem in making the proof of Item (b) work for Item (c) is a construction involving the enumeration of a set: one needs to find in polynomial time the ith element in an exponential-size P set. In general, is there a similar symmetry of information theorem for subexponential time-bounded Kolmogorov complexity? Source: L. Longpré and S. Mocas, *Inform. Process. Lett.*, 46:2(1993), 95–100. They proved that if a certain version of (c) holds, then one-way functions do not exist.

7.1.13. [36] We investigate the open problem of Exercise 7.1.12, Item (c), further. We call this problem 'polynomial-time symmetry of information.' Let $p(n)$ be a polynomial. We say that a subset of $\{0,1\}^*$ contains *almost all strings* (of $\{0,1\}^*$) if for each n it contains a fraction of at least $1 - 1/p(n)$ of $\{0,1\}^{\leq n}$. We call a string w a *polynomial-time description* of x if the universal reference machine U computes x from w in polynomial time.

(a) If the polynomial-time symmetry of information holds, then there is a polynomial-time algorithm that computes the shortest p-time description of a string for almost all strings.

(b) If the polynomial-time symmetry of information holds, then every polynomial-time computable function is probabilistic polynomial-time invertible for almost all strings in its domain.

(c) If P = NP (which means that every polynomial-time computable function is polynomial-time invertible), then the polynomial-time symmetry of information holds.

Comments. The prime question is whether symmetry of information holds in a polynomial-time-bounded environment. Intuitively, this problem is related to the complexity of inverting a polynomial-time computable function. The above evidence supports this intuition. Source: L. Longpré and O. Watanabe, *Inform. Computation*, 121:1(1995), 14–22.

7.1.14. [27/O35] Let $T(n), f(n)$ be functions both computable in time $T(n)$. Prove that $C[f(n), T(n), \infty] \subset C[f(n), c2^{f(n)}T(n), \infty]$, for some constant c. Open problem: can we establish a tighter version of this time hierarchy without an exponential-time gap in the hierarchy?

Comments. Source: L. Longpré, Ph.D. thesis, Cornell University, 1986.

7.1.15. [25] Show that for some c, $C[n, \infty, c] \bigcap C[\log n, \infty, c \log n] \neq \varnothing$ for large n. Can you formulate other tradeoff results (between complexity, time, and space)?

7.1.16. [25] Kolmogorov complexity arguments may be used to replace diagonalization in computational complexity. Prove the following using Kolmogorov complexity:

(a) If $\lim_{n\to\infty} s(n)/s'(n) \to 0$, and $s'(n) \geq \log n$ computable in space $s'(n)$, then DSPACE$[s'(n)]$ – DSPACE$[s(n)] \neq \varnothing$.

(b) If $\lim_{n\to\infty} s(n)/s'(n) \to 0$, with $s'(n)$ computable in space $s'(n)$ and $s'(n) \geq 3n$, then there is a language $L \in$ DSPACE$[s'(n)]$ such that L is DSPACE$[s(n)]$-immune.

(c) Let $r(n)$ be a total recursive function. There exists a recursive language L such that for every Turing machine T_i accepting L in space $S_i(n)$, there exists a Turing machine T_j accepting L in space $S_j(n)$ such that $r(S_j(n)) \leq S_i(n)$, for almost all n.

(d) Exhibit a Turing machine that accepts an infinite set containing no infinite regular sets.

Comments. Source: suggested by B.K. Natarajan. In Item (a) we consider the *DSPACE hierarchy.* The original space hierarchy theorem was studied by R. Stearns, J. Hartmanis, and P. Lewis II [*6th IEEE Symp. Switching Circuit Theory and Logical Design,* 1965, pp. 179–190]. Item (c) is the *Blum speedup theorem* from [M. Blum, *J. Assoc. Comp. Mach.*, 14:2(1967), 322–336].

7.2 Language Compression

If A is a recursive set and x is lexicographically the ith element in A, then obviously $C(x) \leq \log i + c_A$ for some constant c_A depending only on A. If further, the membership of A can be determined in polynomial-time $p(n)$, that is, A is in P, and $A^{=n} = \{x \in A : l(x) = n\}$, then we are inclined to conjecture,

> There exists a constant c_A such that for all $x \in A^{=n}$,
> we have $C^p(x|n) \leq \log d(A^{=n}) + c_A$.

But since a Turing machine cannot search through all 2^n strings in polynomial-time $p(n)$, the argument that worked in the recursive case does not apply in the polynomial-time setting. This conjecture has interesting consequences in language compression. Results in this section all stem from this question. Although whether the conjecture is true is still an important open problem in time-bounded Kolmogorov complexity, we try to provide some partial answers. We deal only with time-bounded Kolmogorov complexity C^t and CD^t.

Definition 7.2.1 (i) Let $\Sigma = \{0, 1\}$. A function $f\colon \Sigma^* \to \Sigma^*$ is a *compression* of language $L \subseteq \Sigma^*$ if f is one-to-one on L and for all except finitely many $x \in L$, we have $l(f(x)) < l(x)$.

(ii) For a function f, the inverse of f is $f^{-1}\colon f(L) \to L$ such that for every $x \in L$, we have $f^{-1}(f(x)) = x$. A language L is *compressible* in time

$T(n)$ if there is a compression function f for L that can be computed in time $T(n)$, and its inverse f^{-1} can also be computed in time $T(n)$.

(iii) A compression function f *optimally compresses* a language L if for every $x \in L$ of length n, $l(f(x)) \leq \lceil \log \sum_{i=0}^{n} d(L^{=i}) \rceil$.

(iv) A natural and optimal form of compression is *ranking*. The ranking function $r_L \colon L \to \mathcal{N}$ maps $x \in L$ to its index in a lexicographic ordering of L.

Obviously, language compression is closely related to the Kolmogorov complexity of the elements in the language. Efficient language compression is closely related to the time-bounded Kolmogorov complexity of the elements of the language. Using a ranking function on a recursive set, we achieve the optimal Kolmogorov complexity for most elements in the set. In this sense, ranking optimally compresses a recursive set. When there is no resource bound, this compression is trivial. Our purpose is to study the polynomial-time setting of the problem. This is far from trivial.

7.2.1 Decision Compression

Given a polynomial-time computable set A and $x \in A^{=n}$, can we compress x by representing x in $\log d(A^{=n}) + O(1)$ bits? In order to apply the nonresource-bounded Kolmogorov complexity in previous chapters, we have often implicitly relied on Theorem 2.1.3 on page 111. The theorem states that given a recursively enumerable set A, for every $x \in A^{=n}$ we have $C(x|n) \leq \log d(A^{=n}) + O(1)$. The short program for x is simply its index in the enumeration of elements in $A^{=n}$. Similarly, we also have $CD(x|n) \leq \log d(A^{=n}) + O(1)$.

Even for A in P, a program that generates x in this way must search through all 2^n strings over Σ^n and test whether they belong to $A^{=n}$. Such a process takes at least exponential time. We are interested in the compression achievable in polynomial time. Remarkably, for CD, near optimal compression is achievable in polynomial time.

Theorem 7.2.1 *Let A be a set, given as an oracle. Then, there is a constant c and a polynomial p such that for every string $x \in A^{=n}$,*

$$CD^p(x|A^{=n}) \leq 2 \log d(A^{=n}) + 2 \log n + c.$$

Proof. Let A be a set and $d = d(A^{=n})$.

Lemma 7.2.1 *Let $S = \{x_1, \ldots, x_d\} \subseteq \{0, \ldots, 2^n - 1\}$. For every $x_i \in S$ there exists a prime $p_i \leq 2dn$ such that for every $j \neq i$ we have $x_i \not\equiv x_j \bmod p_i$.*

Proof. Let $N = 2^n$. Consider prime numbers between c and $2c$. For each pair $x_i, x_j \in S$ there are at most $\log_c N = \log N / \log c$ different prime numbers p such that $c \leq p \leq 2c$ and $x_i \equiv x_j \bmod p$. Fix index i. Since there are only $d-1$ pairs of strings in S containing x_i, it follows that there exists a prime number p_i among $(d-1)\log N/\log c + 1$ prime numbers between c and $2c$ such that for every $j \neq i$ we have $x_i \not\equiv x_j \bmod p_i$. The prime number theorem, Exercise 1.5.8 on page 17, implies that there are at least $c/\log c$ prime numbers between c and $2c$. Therefore, taking $c > (d-1)\log N$ yields that $p_i \leq 2d \log N = 2dn$. □

Let A play the role of S in Lemma 7.2.1. For $x \in A$, apply Lemma 7.2.1 to get p_x. The *CD* program for x works as follows:

Step 1. Input y.

Step 2. **If** $y \notin A^{=n}$ by oracle query to A **then** reject y
else if $y \equiv x \bmod p_x$ **then** accept y **else** reject y.

The size of the above program is $l(p_x) + l(x \bmod p_x) + O(1)$. This is $2\log d(A^{=n}) + 2\log n + O(1)$. Notice that by padding, we can make the descriptions of p_x and $x \bmod p_x$ equally long, precisely $\log d + \log n$ bits. Concatenating the two descriptions, we can parse them in the middle, saving an $O(\log\log n)$-bit overhead for a delimiter to separate them. Clearly, the program runs in polynomial time, and accepts only x. □

Corollary 7.2.1 *Let A be a set in* P. *There is some polynomial p such that for every string $x \in A^{=n}$ it holds that*

$$CD^p(x|n) \leq 2\log d(A^{=n}) + 2\log n + O(1).$$

Can we improve Theorem 7.2.1 and Corollary 7.2.1? Exercise 7.2.3 shows that Theorem 7.2.1 is tight for some set A. Exercise 7.2.4 improves Corollary 7.2.1 for most strings in A. The next theorem states that the compression can be made almost optimal at the penalty of adding more information to the conditional. That is, we are given a magical string s depending only on $A^{=n}$ for free.

Theorem 7.2.2 *Let A be a set. There is a polynomial $p(n)$ such that for every large n, if $x \in A^{=n}$, then*

$$CD^p(x|A^{=n}, s) \leq \log d(A^{=n}) + \log\log d(A^{=n}) + O(1),$$

where $A^{=n}$ is given as an oracle, and the string s depends only on $A^{=n}$ and has length about $n \log d(A^{=n})$.

In this theorem, as in Theorem 7.2.1, we do not need to assume that A is accepted in polynomial time. If, for example, for some c and for each n, the set $A^{=n}$ is accepted by a circuit of size n^c, then the oracle $A^{=n}$ in the condition can be replaced by a finite string, of polynomial length, describing the circuit. In the proof, if we need to decide whether an element x is in $A^{=n}$, then we just simulate the circuit in polynomial time. The same applies to Theorem 7.2.4.

Proof. In order to prove the above theorem, we need a coding lemma. Let $k = d(A^{=n})$ and $m = 1 + \lceil \log k \rceil$. Let $h : \Sigma^n \to \Sigma^m$ be a linear transformation given by a randomly chosen $m \times n$ binary matrix $R = \{r_{ij}\}$. That is, for each $x \in \Sigma^n$, we have that Rx is a string $y \in \Sigma^m$ with $y_i \equiv \left(\sum_j r_{ij} \cdot x_j\right) \bmod 2$.

Let H be a collection of such functions. Let $B, C \subseteq \Sigma^n$ and $x \in \Sigma^n$. The mapping h *separates* x *within* B if for every $y \in B$, different from x, it maps $h(y) \neq h(x)$. The mapping h *separates* C *within* B if it separates each $x \in C$ within B. The set of mappings H *separates* C *within* B if for each $x \in C$ some $h \in H$ separates x within B. In order to give each element in B a (logarithmic) short code, we randomly hash elements of B into short codes. If collisions can be avoided, then elements of B can be described by short programs. We now state and prove the promised coding lemma.

Lemma 7.2.2 **(Coding lemma)** *Let $B \subseteq \Sigma^n$, where $d(B) = k$. Let $m = 1 + \lceil \log k \rceil$. There is a collection H of m random linear transformations $\Sigma^n \to \Sigma^m$ such that H separates B within B.*

Proof. Fix a random string z of length nm^2 such that $C(z|B, P, m, n) \geq l(z)$, where P is the program for describing z (independent of the actual z) in the following discussion. Cut z into m equal pieces. Use the nm bits from each piece to form an $n \times m$ binary matrix in the obvious way. Thus, we have constructed a set H of m random matrices. We claim that H separates B within B.

Assume the contrary. That is, for some $x \in B$, no $h \in H$ separates x within B. Then there exist $y_1, \ldots, y_m \in B$ such that $h_i(x) = h_i(y_i)$ $(1 \leq i \leq m)$. Hence, $h_i(x - y_i) = 0$. (Here $x - y_i$ means mod 2 element-wise subtraction.)

Since $x - y_i \neq 0$, the first column of h_i corresponding to a 1 in the vector $x - y_i$ can be expressed by the rest of the columns using $x - y_i$. Now we can describe z by a fixed program P that uses the following data:

- the index of x in B, using $\lceil \log k \rceil$ bits;
- the indices of $y_1, \ldots, y_m$, in at most $m\lceil \log k \rceil$ bits;

- the matrices $h_1, \ldots, h_m$ each minus its redundant column, in $m^2 n - m^2$ bits.

From this description of the nonredundant columns of the h_i's, and x and the y_i's, the program P given B (as an oracle) will reconstruct h_i. The total length of the description is only

$$m^2 n - m^2 + \lceil \log k \rceil + m(\lceil \log k \rceil) \leq m^2 n - 1.$$

Hence, $C(z|B, P, m, n) < l(z)$, a contradiction. □

We continue the proof of the theorem. Let H be a collection of functions as given by Lemma 7.2.2. Let s be an encoding of H. For every $y \in A^{=n}$ there is some $h_i \in H$ that separates y within $A^{=n}$. Given oracle $A^{=n}$ and s, a short program with input $ih_i(y)$ can accept precisely y as follows:

Check in lexicographic order for every candidate x whether $x \in A^{=n}$. If $x \in A^{=n}$, then decode $ih_i(y)$ into i and $h_i(y)$, and find h_i using i and s. Compute $h_i(x)$. Accept x iff $h_i(x) = h_i(y)$, using the fact that h_i separates y within $A^{=n}$.

Therefore, y can be described by i followed by $h_i(y)$, where i uses precisely $\lceil \log m \rceil$ bits by padding and $h_i(y)$ requires m bits. All of this can be done in polynomial time, say $p(n)$ time. Hence,

$$\begin{aligned} CD^p(y|A^{=n}, s) &\leq m + \log m + O(1) \\ &= \log d(A^{=n}) + \log\log d(A^{=n}) + O(1). \end{aligned}$$

□

Example 7.2.1 We provide another application of the coding lemma, Lemma 7.2.2. A PTM (*probabilistic Turing machine*) is a Turing machine that can flip a fair coin to determine its next move. A PTM is also called a random algorithm. The class BPP is the set of all decision problems solvable by a polynomial-time PTM such that the "yes/no" answer always has probability at least $\frac{1}{2} + \delta$ of being correct for some fixed $\delta > 0$. One can make the error probability less than $1/2^n$ by running such an algorithm a polynomial number of times and taking the majority answer.

Theorem 7.2.3 $\mathrm{BPP} \subseteq \Sigma_2^p \bigcap \Pi_2^p$.

Proof. Let $B \in \mathrm{BPP}$ be accepted by a PTM T using $m = n^k$ coin flips (random bits), with error probability $\epsilon \leq 1/2^n$ on inputs of length n. Let $E_x \subset \Sigma^m$ be the collection of coin-flip sequences of length m on which T rejects x. If $x \in B$, then we must have $d(E_x) \leq 2^{m-n}$, since otherwise we would have $\epsilon > 1/2^n$. Setting $l = 1 + m - n$, the

coding lemma, Lemma 7.2.2, states that there is a collection H of l linear transformations from Σ^m to Σ^l separating E_x within E_x. If $x \notin B$, then $d(E_x) > 2^{m-1}$, since otherwise $\epsilon > 1/2^n$ again. But then, by the pigeonhole principle, no such collection H as above exists. Hence, $x \in B$ if and only if such an H exists. The latter can be expressed as

$$(\exists H)(\forall e \in E_x)(\exists h \in H)(\forall e' \in E_x)[e \neq e' \Rightarrow h(e) \neq h(e')].$$

The second existential quantifier has polynomial range, and hence can be absorbed into the polynomial-time computation. So, BPP $\subseteq \Sigma_2^p$. Since BPP is closed under complement, BPP $\subseteq \Pi_2^p$ and BPP $\subseteq \Sigma_2^p \bigcap \Pi_2^p$. □

Denote the set of languages accepted in random polynomial time by R. Let $\mathrm{R}_2 = \mathrm{R}^{\mathrm{NP}}$ be the collection of languages accepted in random polynomial time with an oracle for an NP-complete set. The above proof also yields BPP $\subseteq \mathrm{R}_2 \bigcap \mathrm{CoR}_2$. ◇

7.2.2 Description Compression

We discuss the compression issues with respect to C^t. Let A be a recursive set and let $d(A^{=n}) \leq p(n)$ for some polynomial p for all n. Then by Theorem 2.1.3 on page 111, there exists a constant c such that for every $x \in A^{=n}$,

$$C(x|n) \leq c \log n + c. \tag{7.4}$$

The next theorem states that this is true up to a $\log \log d(A^{=n})$ additive term in a polynomial-time setting with the help of a Σ_2^p oracle.

Theorem 7.2.4 *There is a polynomial $p(n)$ such that for every set A and every large n, if $x \in A^{=n}$, then*

$$C^p(x|A^{=n}, s, \mathrm{NP}^A) \leq \log d(A^{=n}) + \log \log d(A^{=n}) + O(1), \tag{7.5}$$
$$C^p(x|A^{=n}, \Sigma_2^{p,A}) \leq \log d(A^{=n}) + \log \log d(A^{=n}) + O(1), \tag{7.6}$$

where $A^{=n}$ (an NP^A-complete set) and some $\Sigma_2^{p,A}$-complete set are given as oracles, and s is a string of length about $n \log d(A^{=n})$.

Proof. Equation 7.5 follows from Theorems 7.1.2 and 7.2.2. To prove Equation 7.6, we use the $\Sigma_2^{p,A}$ oracle B to find s, the encoding of H in the proof of Theorem 7.2.2. By that proof there exists an H such that for all $x \in A^{=n}$, some h_i in H separates x within $A^{=n}$ (H separates $A^{=n}$ within $A^{=n}$). The idea is to reconstruct s by asking questions of B. Since H is not unique, we construct s bit by bit. Assume that we have constructed the prefix $s_1 \ldots s_i$ of s. We extend this by one more bit s_{i+1}, where $s_{i+1} = 1$ if oracle B answers "yes" to the question "is there an H with prefix $s_1 \ldots s_i s_{i+1}$ such that for all $x \in A^{=n}$, some $h_i \in H$ separates x within $A^{=n}$?" □

We can get rid of the oracle at the cost of obtaining much weaker inequalities than Equation 7.4 or Theorem 7.2.4. For a given language L, define the density function of L to be $\mu_L(n) = d(L^{=n})/2^n$.

Theorem 7.2.5 *If L is acceptable in polynomial time, $k > 3$, and $\mu_L \leq n^{-k}$, then L can be compressed in probabilistic polynomial time. The compression function maps strings of length n to strings of length at most $n - (k-3)\log n + c$.*

Proof. Complicated and omitted. See [A. Goldberg and M. Sipser, *SIAM J. Comput.*, 20(1991), 524–536]. □

Theorem 7.2.5 is far from optimal in two ways:

- If a language L is very sparse, say $\mu_L \leq 2^{-n/2}$, then one expects to compress $\frac{1}{2}n$ bits instead of only $O(\log n)$ bits given by the theorem. Can the $O(\log n)$ term be improved?
- The current compression algorithm is probabilistic; can this be made deterministic?

In computational complexity, oracles sometimes help us to understand the possibility of proving a new theorem. The following result shows that if S, the language to be compressed, is not restricted to be polynomial-time acceptable, and the membership query of S is given by an oracle, then compression by only a logarithmic term is optimal. The point here is that polynomial-time decidability of language membership is a stronger requirement than membership decidability by just oracle questions. The technique used in the following proof will be used again in the proof of Theorem 7.3.3.

Definition 7.2.2 A set A is called a *sparse set* if for some constant c, for all n, the cardinality $d(A^{=n})$ is at most $n^c + c$.

Theorem 7.2.6 *There is a sparse language L such that if L is compressed by a probabilistic polynomial-time machine with an oracle for L, then the compression function maps strings of length n to strings of length $n - O(\log n)$.*

Proof. Let L be a language that contains exactly one string x for each length n, which is a tower of 2's:

$$n = 2^{2^{\cdots^2}}.$$

The language L contains no strings other than those just described. Each string $x \in L$ is maximally complex:

$$C(x) \geq l(x).$$

Let T be a probabilistic machine with an oracle for L that runs in time n^k. By way of contradiction, let f be the compression function such that for large n and $x \in L$ of length n, and constant c_1 to be chosen later, $l(f(x)) \leq n - c_1 - (k + c_1) \log n$. It is enough to show that T cannot restore x from $f(x)$ with probability at least $\frac{1}{2}$.

Assume that T on input $f(x)$ outputs x with probability $\frac{1}{2}$. Hence, on half of the 2^{n^k} coin-toss sequences of length n^k, T should output the correct x. Let R be the set of such coin-toss sequences that lead to correct x. Choose $r \in R$ such that

$$C(r|x) \geq l(r) - 1.$$

Since $C(r) \leq l(r) + O(1)$ by the last two displayed equations, we have by the symmetry of information theorem, Theorem 2.8.2 on page 190, that for some constant c_2,

$$C(x|r) \geq n - c_2 \log n. \tag{7.7}$$

In order to give a short description of x, relative to r, we will describe in $(k+2) \log n + c_1$ bits all information needed to answer the oracle queries of T on input $f(x)$.

The language L is so sparse that on input of length of n (which is a tower of 2's), T has no time to write down a string in L of length greater than n because every such string has length at least 2^n. That is, for every query "$x' \in L$?" if $l(x') > n$, then the answer is "no." In addition, we can write down all the strings in L of lengths less than n in fewer than $2 \log n$ bits. This enables us to answer queries of lengths less than n.

Since T, with respect to the coin-toss sequence r, outputs x, we can redescribe x by simulating T on input $f(x)$, using the random sequence r and the following information:

- This discussion, in at most c_1 bits.
- $2 \log n$ bits to encode strings in L of length less than n.
- If T queries "$x \in L$?" then we use $k \log n$ bits to indicate the first step when T makes such query. Before this step, if T queries about a string of length n, then the answer is "no."

Therefore,

$$C(x|r) \leq c_1 + 2 \log n + k \log n + l(f(x)).$$

Choosing $c_1 > c_2 + 2$, the right side of the inequality is less than $n - c_2 \log n$, contradicting Equation 7.7. □

There is a language L, of density $\mu_L < 2^{-n/2}$, that cannot be compressed by any deterministic polynomial-time machine that uses an oracle for L. Further, the following results can be shown by diagonalization: (i) There is an exponential-time language that cannot be compressed in deterministic polynomial time. (ii) There is a double-exponential-time language that cannot be compressed in probabilistic polynomial time. See A. Goldberg and M. Sipser, *SIAM J. Comput.*, 20(1991), 524–536.

7.2.3 Ranking

Ranking is a special and optimal case of compression. Recall that the ranking function r_L maps the strings in L to their indices in the lexicographic ordering of L. If $r_L : L \to \mathcal{N}$ is polynomial-time computable, then so is $r_L^{-1} : \mathcal{N} \to L$ (Exercise 7.2.7 on page 560). We are interested only in polynomial-time computable ranking functions. In the previous sections we met several hard-to-compress languages. But there are also quite natural language classes that are easy to compress.

A one-way log-space Turing machine is a deterministic Turing machine with a separate one-way input tape on which the input is delimited between distinguished end markers, and a fixed number of separate work tapes. During its computation on an input of length n, the machine uses only $O(\log n)$ space on its work tapes.

Theorem 7.2.7 *If a language L is accepted by a one-way log-space Turing machine, then r_L can be computed in polynomial time.*

Proof. Let T be a one-way log-space Turing machine that accepts L. We want to compute $r_L(x)$ for each string x of length n. Write $y \leq x$ if y precedes x in the length-increasing lexicographic order. Let $L_x = \{y \in L : y \leq x\}$. Trivially, $r_L(x) = d(L_x)$. By storing x in the internal memory of T, we obtain a machine T_x accepting L_x. That is, T_x simulates T on each input and compares the input with x at the same time. The machine T_x accepts iff T accepts the input and the input is less than or equal to x.

T_x has size polynomial in $n = l(x)$ and uses $\log n$ space. We can construct a directed computation graph $G = (V, E)$ for T_x as follows: Let an ID of T_x be a triple,

(state, input head position, work tape content).

The set of nodes V contains all possible IDs of T_x. The set of edges E contains directed edges that represent moves of T_x. The sets V and E are polynomially bounded in n because T_x has a polynomial number of states, n input positions, and $\log n$ work tape cells. Since T_x is deterministic and runs in bounded time, G is loop-free.

In order to compute r_L, we need only to calculate $d(L_x)$ by counting the number of accepting paths in G from the initial ID to the final ID. This

latter task can be easily done by dynamic programming, since G does not contain loops. □

Exercises

7.2.1. [20] (a) Show that a set S is sparse iff for all $x \in S$, $CD^p(x|S) \leq O(\log l(x))$ for some polynomial p.

(b) Show that set $S \in \mathrm{P}$ is sparse iff for all $x \in S$, $CD^p(x) \leq O(\log l(x))$ for some polynomial p.

Comments. Use Theorem 7.2.1. Source: H.M. Buhrman and L. Fortnow, *Proc. 14th Symp. Theoret. Aspects Comput. Sci.*, Lect. Notes Comput. Sci., Springer-Verlag, Berlin, 1997; H.M. Buhrman, L. Fortnow, and S. Laplante, *SIAM J. Comput.*, 31:3(2001), 887–905. The authors also define a nondeterministic version of *CD* and prove several results.

7.2.2. [30] (a) Show that for $0 \leq x < y < 2^n$ there are at most n primes p such that $x \equiv y \bmod p$.

(b) Prove that there is some polynomial p such that for all formulas $\phi(x_1, \ldots, x_n) \in \mathrm{SAT}$ and all r such that $l(r) = p(l(\phi))$ and $C(r) \geq l(r)$, there is some satisfying assignment a of $(x_1, \ldots, x_n)$ such that $CD^p(a|\phi, r) \leq O(\log n)$.

Comments. Hint: see the proof of Theorem 7.2.1. Source: H.M. Buhrman and L. Fortnow, *Ibid.*, where a connection is given between this exercise and [L. Valiant and V. Vazirani, *Theoret. Comput. Sci.*, 47(1986) 85–93].

7.2.3. [35] For every polynomial p and sufficiently large n, there exists a set of strings $A \subseteq \{0,1\}^*$ such that $A^{=n}$ contains more than $2^{n/50}$ strings and there is an $x \in A^{=n}$ with

$$CD^p(x|A^{=n}) \geq 2 \log d(A^{=n}) - O(1).$$

Comments. This and the next exercise answer the open question posed in Exercise 7.2.3 in the second edition of this book. Source: H. Buhrman, S. Laplante, and P. Miltersen, *Proc. 15th IEEE Conf. Comput. Complexity*, 2000, pp. 126–130.

7.2.4. [38] For every set A in P, for all constants $\alpha, \epsilon > 0$, there is a polynomial p such that for all n and for all but an ϵ fraction of the $x \in A^{=n}$,

$$CD^p(x) \leq \min\{\log d(A^{=n}) + \log^{O(1)}(n), (1+\alpha) \log d(A^{=n}) + O(\log n)\}.$$

Comments. Source: H.M. Buhrman, L. Fortnow, and S. Laplante, *SIAM J. Comput.* 31:3(2001), 887–905.

7.2.5. [30] There is an infinite set A such that for every polynomial p, $CD^p(x|A) \geq l(x)/5$ for almost all $x \in A$.

Comments. A corollary is that A has no spare subsets in P^A. Source: L. Fortnow and M. Kummer, *Theoret. Comp. Sci. A*, 161(1996), 123–140.

7.2.6. [28] Let f be a function on the natural numbers. Let $\Sigma = \{0, 1\}$. A set A belongs to the class P/f if there exist another set $B \in \mathrm{P}$ and a function $h\colon \mathcal{N} \to \Sigma^*$ such that for all n, we have $l(h(n)) \leq f(n)$; and for all x, we have $x \in A$ iff $\langle x, h(l(x))\rangle \in B$. Define $\mathrm{P/poly} = \bigcup_{c>0} \mathrm{P}/n^c$. Prove that $\mathrm{BPP} \subseteq \mathrm{P/poly}$, using Kolmogorov complexity.

Comments. Let T be a probabilistic machine such that the error probability is $1/2^{n^2}$ and each path is of length n^k. A Kolmogorov random string of length n^k will always give a correct path. Source: W. Gasarch, e-mail, July 16, 1991. The class P/f was defined by R.M. Karp and R.J. Lipton in [*L'Enseignement Mathématique*, 28(1982), 191–209].

7.2.7. [28] Let r_L be the ranking function of L. Show that if r_L is polynomial-time computable, then so is r_L^{-1}.

Comments. Source: A. Goldberg and M. Sipser, *SIAM J. Comput.*, 20(1991), 524–536.

7.2.8. [30] A problem is in #P if there is a nondeterministic Turing machine such that for each input, the number of distinct accepting paths of the Turing machine is precisely the number of solutions for the problem for this input. #P-complete problems are defined (analogously to NP-complete problems) as follows: the problem is in #P, and every #P problem can be reduced to it by a polynomial-time deterministic Turing machine computation. As an example, counting the number of satisfying truth assignments for SAT is #P-complete. Let $\mathbf{C}$ be a class of languages. Say $\mathbf{C}$ is P-rankable if for all $L \in \mathbf{C}$, the ranking function r_L is polynomial-time computable. Prove:

(a) P is P-rankable iff NP is P-rankable.

(b) P is P-rankable iff $\mathrm{P} = \mathrm{P}^{\#\mathrm{P}}$.

(c) PSPACE is P-rankable iff P = PSPACE.

(d) P/poly is not P-rankable, where the class P/poly is defined in Exercise 7.2.6.

(e) Languages accepted by two-way deterministic pushdown automata are P-rankable iff P = #P.

(f) Languages accepted by one-way multihead DFAs are P-rankable iff P = #P.

(g) Languages accepted by one-way log-space-bounded nondeterministic Turing machines are P-rankable iff P = #P.

Comments. Source: Items (a)–(d) are from [L. Hemachandra and S. Rudich, *J. Comput. System Sci.*, 41:2(1990), 251–271]. Items (e)–(g) are from [D.T. Huynh, *Math. Systems Theory*, 23(1990), 1–19]. The #P class was introduced by L. Valiant [*Theoret. Comput. Sci.*, 8(1979), 189–201].

7.2.9. [40] Is it possible that #P problems have solutions of low time-bounded Kolmogorov complexity relative to the input? Prove that if there is a polynomial-time Turing machine that on input that is a Boolean formula f, prints out a polynomial-sized list of numbers among which one number is the number of solutions of f (though we may not know which one), then $\mathrm{P} = \mathrm{P}^{\#\mathrm{P}}$. This gives strong evidence that in general, the number of solutions of a Boolean formula has high time-bounded Kolmogorov complexity relative to the formula.

Comments. Source: J.-Y. Cai and L. Hemachandra, *Inform. Process. Lett.*, 38(1991), 215–219. Hint: use A. Shamir's polynomial interpolation technique in [*Proc. 31st IEEE Found. Comput. Sci.*, 1990, pp. 11–15].

7.2.10. [35] Let $\chi_L = \chi_1\chi_2\ldots$ be the characteristic sequence for language $L \subseteq \{0,1\}^*$ such that $\chi_i = 1$ iff the lexicographically ith word w_i is in L. As before, $L^{<n}$ is the set of strings in L with length less than n. Define the time-space-bounded Kolmogorov complexity of L as $C^{t,s}(\chi_{L^{<n}})$. Let $t(n) = 2^{n^{O(1)}}$ and $s(n,\epsilon) = c^{n^\epsilon}$. Prove by diagonalization:

(a) There is language $L \in \mathrm{DTIME}[2^{2^{O(n)}}]$ which is such that the $t(n)$ time-bounded Kolmogorov complexity of L is exponential almost everywhere (note: $\chi_{L^{<n}}$ has exponential length). That is, for all but finitely many n,

$$C^{t(n),\infty}(\chi_{L^{<n}}|n) > s(n,\epsilon), \quad \text{for some } c > 1, \epsilon > 0.$$

(b) Use Item (a) to show that if L is $\mathrm{DTIME}[2^{2^{O(n)}}]$-hard under polynomial-time Turing reduction, then the $t(n)$ time-bounded Kolmogorov complexity of L is exponential almost everywhere. That is, for all but finitely many n,

$$C^{t(n),\infty}(\chi_{L^{<n}}|n) > s(n,\epsilon), \quad \text{for some } c > 1, \epsilon > 0.$$

(c) There is a language $L \in \mathrm{SPACE}[2^{O(n)}]$ such that for all but finitely many n, we have $C^{\infty,2^n}(\chi_{L^{<n}}|n) > 2^{n-2}$.

(d) Use Item (c) to show that if L is $\mathrm{SPACE}[2^{O(n)}]$-hard under polynomial-time Turing reduction, then there exists a constant $\epsilon > 0$ such that for all but finitely many n, we have $C^{\infty,s(n,\epsilon)}(\chi_{L^{<n}}|n) > s(n,\epsilon)$.

(e) There is a language $L \in \mathrm{SPACE}[2^{O(n)}]$ such that for large enough n, $C^{\infty,2^n}(\chi_{L^{<n}}) > 2^n - n$.

(f) Consider the P/poly class defined in Exercise 7.2.6. It is not known whether DTIME[$t(n)$] is contained in P/poly. A nonsparse language L is said to be P/poly *immune* if it has only sparse subsets in P/poly. A language L is said to be P/poly *bi-immune* if L and its complement are both P/poly immune. Use Item (e) and the language L in that item to show that there exists a language $L \in \text{SPACE}[2^{O(n)}]$ that is P/poly bi-immune.

Comments. Source: D.T. Huynh, in [*Proc. 1st Conf. Structure Complexity Theory,* 1986, pp. 184–195; *Theoret. Comput. Sci.*, 96(1992), 305–324; *Inform. Comput.*, 90(1991), 67–85; *Inform. Process. Lett.*, 37(1991), 165–169].

7.3 Computational Complexity

Time- and space-bounded Kolmogorov complexities are natural tools in the study of time- and space-bounded computations and the study of the structures of the corresponding complexity classes. As in Chapter 6, they provide powerful techniques and help to make intuitive arguments rigorous.

7.3.1 Constructing Oracles

With a simple input, a polynomial-time bounded Turing machine cannot compute too complicated strings. Therefore, it cannot ask complicated questions to an oracle. Kolmogorov complexity enables us to formalize this intuition.

Definition 7.3.1 A function $f : \{0,1\}^* \to \{0,1\}^*$ is *honest* if there exists a k such that for every $x \in \{0,1\}^*$,

$$l(f(x)) \leq l(x)^k + k \quad \text{and } l(x) \leq l(f(x))^k + k.$$

That is, f neither shrinks nor stretches x more than polynomially in its length. The following result is used to prove Theorems 7.3.1 and 7.3.2.

Lemma 7.3.1 **(Honesty lemma)** *Let f be an honest function computable in polynomial time. For all $t \geq 1$, and all but finitely many n,*

$$f(C[\log\log n, n^t, \infty]) \subseteq C[\log n, n^{\log\log n}, \infty].$$

Proof. Let $n = l(x)$. By assumption, there is a k such that Definition 7.3.1 holds. Let machine T compute f and run in time polynomial in n. For each x in $C[\log\log n, n^t, \infty]$ there exists a y, with $l(y) \leq \log\log n$, such that some T' computes x from y and runs in time polynomial in n. So $f(x)$ is computable from y, T, and T'. This description has fewer than

$\log n$ bits for large n. Moreover, the computation takes polynomial time, which is less than $n^{\log\log n}$ for large n. So, for all but finitely many n,

$$f(C[\log\log n, n^t, \infty]) \subseteq C[\log n, n^{\log\log n}, \infty].$$

□

We present two examples applying this lemma. Baker, Gill, and Solovay showed how to create an oracle A as in the following theorem. We present an alternative proof by the incompressibility argument.

Theorem 7.3.1 *There is a recursive oracle A such that $\mathrm{P}^A \neq \mathrm{NP}^A$.*

Proof. By diagonalization. Define f inductively by $f(1) = 2$ and $f(k) = 2^{f(k-1)}$ for $k > 1$. Choose $B \subseteq \{1^{f(k)} : k \geq 1\}$ with $B \in \mathrm{DTIME}[n^{\log n}] - \mathrm{P}$. Use B to construct A as follows: For every k such that $1^{f(k)} \in B$, put the first string of length $f(k)$ from

$$C[\log n, n^{\log n}, \infty] - C[\log n, n^{\log\log n}, \infty]$$

in A. This set is nonempty by Exercise 7.1.14. Clearly, A is recursive and $B \in \mathrm{NP}^A$.

Suppose we present a polynomial-time bounded Turing machine with an input $1^{f(k)}$. This input has length $n = f(k)$. The machine can compute only strings not in A or of length less than $\log n$ in A by Lemma 7.3.1. Therefore, a P^A-oracle machine can in polynomial time query only about strings not in A or of length less than $\log n$. The former strings are not in A anyway. The membership of the latter strings in A can be determined in polynomial time. Therefore, the oracle is useless. Consequently, if B is in P^A, then also $B \in \mathrm{P}$. But this contradicts our assumption that $B \notin \mathrm{P}$. □

In recursion theory, all recursively enumerable-complete sets are recursively isomorphic. We define a similar concept in complexity theory.

Definition 7.3.2 Two sets are said to be *P-isomorphic* if there is a polynomial-time computable bijection between the two sets.

The Berman–Hartmanis conjecture states that all NP-complete sets are P-isomorphic. At this time of writing, all known natural NP-complete sets are P-isomorphic. The following theorem attempts to tackle the conjecture. The SAT decision problem was defined in Definition 1.7.9 and Definition 5.3.3.

Theorem 7.3.2 *If there exists a set $L \in \mathrm{P}$ such that $L \subset \mathrm{SAT}$ and*

$$C[\log n, n^{\log n}, \infty] \bigcap \mathrm{SAT} \subseteq L,$$

then $\mathrm{SAT} - L$ is an NP-complete set that is not P-isomorphic to SAT.

Proof. To see that $\mathrm{SAT} - L$ is $\leq_m^P$-complete for NP we reduce SAT to $\mathrm{SAT} - L$ by reducing all elements in L to a fixed element in $\mathrm{SAT} - L$ and all other strings to themselves. This is possible since $L \in \mathrm{P}$.

At the same time, SAT and $\mathrm{SAT} - L$ cannot be P-isomorphic, since an isomorphism h is an honest function in P and therefore cannot map the simple strings in L onto more complex strings in $\mathrm{SAT} - L$. That is, by the honesty lemma, Lemma 7.3.1, for all but finitely many n,

$$h(\mathrm{SAT} \bigcap C[\log\log n, n^3, \infty]) \subseteq C[\log n, n^{\log n}, \infty].$$

Since $\mathrm{SAT} \bigcap C[\log\log n, n^3, \infty] \neq \varnothing$, it follows that

$$h(\mathrm{SAT} \bigcap C[\log\log n, n^3, \infty]) \not\subseteq \mathrm{SAT} - L.$$

That is, SAT is not P-isomorphic to $\mathrm{SAT} - L$. □

It turns out that resource-bounded Kolmogorov complexity provides rigor to intuition for oracle constructions. We give another slightly more involved example.

Definition 7.3.3 A set A is *exponentially low* if $\mathrm{E}^A = \mathrm{E}$, where $\mathrm{E} = \bigcup_{c \in \mathcal{N}} \mathrm{DTIME}[2^{cn}]$. Obviously, for every $A \in \mathrm{P}$, the set A is exponentially low.

Theorem 7.3.3 *There is an exponentially low sparse set A that is not in P.*

Proof. Let $B = C[\frac{1}{2}n, 2^{3n}, \infty]$. The set $\overline{B}$ is the complement of B. Let $A = \{x : x \text{ is a lexicographically least element of } \overline{B} \text{ of length } 2^{2^{\cdots^2}} \text{ (tower of } m \text{ stacked 2's)}, m > 0\}$. Here m is not meant to be constant. Trivially, $A \in \mathrm{E}$. Furthermore, A is not in P. To prove this, assume that T accepts A in polynomial time. Then we can simulate T to find an x of length n such that $n > 2(l(T) + \log n)$ and $x \in A$, all in less than 2^{3n} time using fewer than $l(T) + \log n$ bits. Hence, $x \in B$, a contradiction.

We also need to show that $\mathrm{E}^A = \mathrm{E}$. We do this by simulating an E^A oracle machine T^A by an E machine. Let the machine T^A run in 2^{cn} time. An idea from the proof of Theorem 7.2.6 is useful:

Claim 7.3.1 Let $y \in A$ and $l(y) \geq c'n$ for a constant $c' > 3c + 3$. Then, machine T^A cannot ask a query "$y \in A$?"

Proof. Assume, by way of contradiction, that T^A can do so. Suppose further that y is the first string in A of length greater than $c'n$ queried by T^A on input x of length n. We will show that $y \in B$, and hence $y \notin A \subseteq \overline{B}$, a contradiction. In order to show $y \in B$, we use the fact that A is so sparse that we can actually describe the list of all strings in A of lengths less than $l(y)$ in fewer than $2\log(c'n)$ bits.

Assume that T^A with input x of length n queries y at step $t < 2^{cn}$. We can now reconstruct y by simulating T^A on input x, and stop T^A at time t. If T^A queries A before step t, by assumption on the computation time to produce a string in A, either the query is shorter than $l(y)$, or the queried string is not in A. For short queries, we can supply the answer from an exhaustive description of $A^{<c'n}$ of fewer than $2\log(c'n)$ bits. At time t, we can recover y from the query tape. In this way, we can describe y by the following items:

- $O(1)$ bits to describe this discussion and T^A;
- n bits to describe x;
- $2\log(c'n)$ bits to describe $A^{<c'n}$;
- cn bits to describe t.

In total, we use fewer than $\frac{1}{2}c'n \leq \frac{1}{2}l(y)$ bits. The time required for the simulation is at most $2^{cn} < 2^{c'n} \leq 2^{l(y)}$. Hence,

$$y \in B = C\left[\frac{1}{2}l(y), 2^{3l(y)}, \infty\right].$$

This contradicts the assumption on A. Hence, T^A queries no string in A of length greater than or equal to $c'n$. □

Now we can simulate T^A without using an oracle. Whenever we meet an oracle query about a string that is longer than $c'n$, we answer "no" for A. Whenever we meet an oracle query about a string y such that $l(y) \leq c'n$, we simply perform an exhaustive search to decide whether $y \in A$ using exponential time. In this way, we simulate an E^A machine by an E machine without using oracle A. Hence, $\mathrm{E}^A = \mathrm{E}$. □

7.3.2 P-Printability

We want to characterize the sets $C[k\log n, n^k, \infty]$, for constant k. These sets are said to have small time-bounded Kolmogorov complexity.

Definition 7.3.4 A set L is *polynomial-time printable* (P-printable) if for some integer k all the elements of L up to size n can be printed by a Turing machine in time $n^k + k$. Clearly every P-printable set is a sparse set in P. Let $y \leq x$

denote that y precedes x in the length-increasing lexicographic order. The ranking function, as defined in Definition 7.2.1, for a language L, denoted by r_L, is defined as $r_L(x) = d(\{y \in L : y \leq x\})$. A *tally set* is a set over a one-letter alphabet.

Theorem 7.3.4 *Let $L \subseteq \{0,1\}^*$. The following statements are equivalent:*

(i) *L is* P*-printable;*

(ii) *L is sparse and has a polynomial-time computable ranking function;*

(iii) *L is* P*-isomorphic to some tally set in* P*; and*

(iv) *$L \subseteq C[k \log n, n^k, \infty]$ for some constant k and $L \in$ P.*

Proof. (i) $\rightarrow$ (ii) is immediate.

(ii) $\rightarrow$ (iii). Let L have a polynomial-time computable ranking function r_1 and $d(L^{\leq n}) < p(n)$ for some (nondecreasing) polynomial $p(n)$. Then $r_2(x) = 1x - r_1(x)$ is a ranking function for the complement of L, where $1x$ is treated as a binary number. The set

$$T = \{0^{np(n)+i} : r_1(1^{n-1}) < i \leq r_1(1^n)\}$$

is a tally set in P. Let r_3 be a ranking function for $\{0\}^* - T$. By Exercise 7.2.7, all such ranking functions have inverses that are computable in time polynomial in the length of their output. It is now easy to see that the function that maps x of length n to $0^{np(n)+r_1(x)}$ if $x \in L$, and to $r_3^{-1}(r_2(x))$ if $x \notin L$, is a P-isomorphism that maps L one-to-one onto T.

(iii) $\rightarrow$ (iv). By assumption, there is a P-isomorphism f, both f and f^{-1} computable in time n^c for some constant c, that maps L one-to-one onto a tally set $T \subseteq \{0\}^*$, where $T \in$ P. Trivially, $L \in$ P.

Since f is computable in time n^c, we have $l(f(x)) \leq n^c$ for $n = l(x)$. Hence, the binary representation of $f(x)$ has length at most $c \log n$. Then, x can be represented by $f(x)$ in binary using only $c \log n$ bits. To compute x from $f(x)$, we simply compute $f^{-1}(0^{f(x)})$, which takes polynomial time by assumption. Thus, $L \subseteq C[k \log n, n^k, \infty]$ for some constant k.

(iv) $\rightarrow$ (i). Assume that $L \in$ P and that for some k,

$$L \subseteq C_U \left[k \log n, n^k, \infty\right].$$

We show how to print elements of L up to size n in polynomial time. Given n, simulate the reference universal machine U, using each string up to length $k \log n$ as input, for at most n^k steps. For each output x, print x if the computation halted in $l^k(x)$ steps, $l(x) = n$, and $x \in L$. Clearly, this can be done in time polynomial in n. □

Corollary 7.3.1 Let $L \subseteq \{0,1\}^*$. Then, $L \subseteq C\left[k \log n, n^k, \infty\right]$ for some constant k if and only if A is P-isomorphic to a tally set.

The paper [L. Fortnow, J. Goldsmith, M.A. Levy, and S. Mahaney, *SIAM J. Comput.*, 28:1(1998), 237–151] treats logpspace-printable sets, with connections to space-bounded Kolmogorov complexity, similar to the connections between P-printable and time-bounded Kolmogorov complexity treated above.

7.3.3 Derandomization

A major question in computational complexity is whether randomized algorithms have more power than their deterministic counterparts. For example, is BPP = P? Intuitively, if we have an incompressible string x, then we should be able to deterministically use x as the random bits required during a BPP computation, and thus derandomize the computation. This is because if a BPP computation makes only a small percentage of errors, then the sequences that lead to erroneous answers are rare, hence not incompressible. We present an example to illustrate this idea.

Theorem 7.3.5 *Let* $R = \{x : C(x) \geq l(x) - \log^2 l(x)\}$. *Then,* $\mathrm{BPP}^R = \mathrm{P}^R$.

Proof. We need to show only that $\mathrm{BPP}^R \subseteq \mathrm{P}^R$. First, we show how to obtain a string in R. We construct such a string x of length m inductively, starting with $x = \epsilon$. Assume that $C(x) \geq l(x) - \log^2 l(x)$. For every string y of length $\log m$, we use R to check whether $C(xy) \geq l(xy) - \log^2 l(xy)$. We are guaranteed to find a y for which the latter condition holds, since there is a y such that $C(y|x) \geq l(y)$. For every such y, by the symmetry of information theorem, Theorem 2.8.2 on page 190, with c the constant in the latter theorem, we have

$$\begin{aligned} C(xy) &\geq C(x) + C(y|x) - c \log C(xy) \\ &\geq C(x) + C(y|x) - c \log l(xy) \\ &\geq l(x) - \log^2 l(x) + l(y) - c \log l(xy) \\ &\geq l(xy) - \log^2 l(xy) \end{aligned}$$

for x and y long enough relative to c. Set $x := xy$, and repeat until $l(x) \geq m$.

Consider a BPP computation of n steps. It uses at most n random bits. For every input of length n, repeat the BPP computation n^2 times and take the majority answer. By the Chernoff bounds, Lemma 1.10.1 on page 61, the probability that this strategy makes a mistake is at most $2e^{-O(n^2)}$. Thus, only $2^M 2^{-O(n^2)}$ sequences cause errors, where $n^2 \leq M \leq n^3$ is the number of random bits used. The Kolmogorov complexity of such sequences is at most $M - O(n^2)$. Therefore, if we use x from R,

with $l(x) = M$, the BPP computation will give a correct answer for every input. □

Theorem 7.3.5 can be improved by replacing R by $R' = \{x : C(x) \geq l(x)/i\}$, where i is a positive integer, Exercise 7.3.13 on page 570. But this requires tools from pseudorandom number generator theory that are beyond the scope of the main text of this book.

Exercises

7.3.1. [28/O43] Let SAT be the set of satisfiable Boolean formulas. By Definition 7.2.2, a set A is sparse if there is a constant c such that for all n we have $d(A^{=n}) \leq n^c + c$. In Section 1.7.4 we defined that a set B is *polynomial-time Turing reducible* to set C, denoted by $B \leq_T^P C$, if there is a polynomial-time oracle Turing machine that accepts B, using oracle C. One might suspect that formulas with low Kolmogorov complexity are easy to solve. Prove the following:

(a) If $A \in$ NP and A is sparse, then $A \leq_T^P C[\log n, n^2, \infty] \bigcap$ SAT. Thus, SAT $\bigcap C[\log n, n^2, \infty]$ is a complete set for all other sparse sets in NP under polynomial-time Turing reduction.

(b) SAT $\bigcap C[\log n, n^2, \infty] \in$ P iff there are no sparse sets in NP $-$ P.

(c) (Open) SAT $\bigcap C[\log n, n^2, \infty] \in$ P?

Comments. Source: this exercise and the next seven exercises are taken from J. Hartmanis, *Proc. 24th IEEE Found. Comput. Sci.*, 1983, pp. 439–445.

7.3.2. [29] Let $g(n) \leq n$ be an unbounded, monotonically increasing function and let $G(n)$ be such that for every k,

$$\lim_{n\to\infty} n^k/G(n) = 0.$$

Show that $C[g(n), G(n), \infty] \bigcap$ SAT $\subseteq A_0 \subset$ SAT and $A_0 \in$ P implies that SAT is not P-isomorphic to SAT $- A_0$. Also show that SAT $- A_0$ is NP-complete.

7.3.3. [26] If $C[\log n, n^{\log n}, \infty] \bigcap$ SAT $\subseteq A_0 \subseteq$ SAT and $A_0 \in$ P, then E = NE.

7.3.4. [40] Prove the following:

(a) There is a sparse set in NP $-$ P iff NE $\neq$ E.

(b) Define Δ_2^{E} analogous to Δ_2^p (Definition 1.7.10 on page 40). If NE $= \Delta_2^{\mathrm{E}}$, and every sparse set in NP is polynomial-time many-to-one reducible (Definition 1.7.8 on page 39) to SAT $\bigcap C[\log n, n^2, \infty]$, then NE = E. Therefore, all sparse sets in NP are in P.

Comments. For Item (b), use the Berman–Mahaney tree-labeling technique in [S. Mahaney, *J. Comput. System Sci.*, 25(1982), 130–143].

7.3.5. [30] Show that P = NP iff for all oracles $A \subseteq C[\log n, \infty, n^2]$ we have $\mathrm{P}^A = \mathrm{NP}^A$.

7.3.6. [29] Use Kolmogorov complexity to show that there exists a recursive oracle A such that NP^A has P^A-immune sets.

7.3.7. [33] Show that the set $\{0,1\}^* - C[\log n, \infty, 2^n]$ is DSPACE$[2^{cn}]$-immune for every $c < 1$.

Comments. Compare this exercise with Theorem 2.7.1.

7.3.8. [34] We construct a sparse random oracle set A as follows: For every n, $n = 1, 2, \ldots$, toss a fair coin. If the result is 'tails,' then we do not include any string of length n in A; if the result is 'heads,' then we toss the coin n times and place the resulting binary string (the ith bit is 1 iff the ith toss is heads) in A. Prove that $\Pr(\mathrm{NP}^A \neq \mathrm{P}^A) = 1$.

7.3.9. [32] The method using resource-bounded Kolmogorov complexity to construct oracles in Section 7.3.1 can be used to obtain many more oracles. Define

$$\mathrm{EXPTIME} = \bigcup_{c \in \mathcal{N}} \mathrm{DTIME}[2^{n^c}].$$

Notice that EXPTIME is different from the class E. Let NSPACE stand for the nondeterministic version of PSPACE. Let us consider oracle Turing machines with an unbounded oracle tape. Prove the following:

(a) There is an oracle A such that $\mathrm{NSPACE}^A \not\subset \mathrm{EXPTIME}^A$.

(b) There is an oracle B such that $\mathrm{PSPACE}^B \subset \mathrm{EXPTIME}^B$, where the containment is proper.

(c) There is an oracle C such that $\mathrm{EXPTIME}^C \not\subset \mathrm{NSPACE}^C$.

(d) There is an oracle D with $\mathrm{PSPACE}^D \subset \mathrm{NSPACE}^D = \mathrm{EXPTIME}^D$, where the containment is proper.

Comments. Item (d) implies that Savitch's theorem [W.J. Savitch, *J. Comput. System Sci.* 4:2(1972), 177–192] does not relativize with an unbounded oracle tape. Source: R. Gavaldà, L. Torenvliet, O. Watanabe, and J.L. Balcázar, *Proc. 15th Math. Found. Comput. Sci. Conf.*, 1991, pp. 269–276. For a comprehensive study in this direction, see [R. Gavaldà, Ph.D. thesis, Universitat Politécnica de Catalunya, 1992].

7.3.10. [39] Show that if A is a set whose characteristic sequence is a random infinite binary sequence in the sense of Martin-Löf (Section 2.5), then $\mathrm{P}^A \neq \mathrm{NP}^A$.

Comments. This result is presented in a more general setting using results of Section 2.5 by R.V. Book, J.H. Lutz, and K.W. Wagner in [*Math. Systems Theory*, 27(1994), 201–209]. Such a set A can be used to establish other known probability-one oracle separations such as $\mathrm{PH}^A \neq \mathrm{PSPACE}^A$, where PH is the polynomial hierarchy. See also [R.V. Book and O. Watanabe, *Inform. Comput.*, 125(1996), 70–76]. Source: C.H. Bennett, J. Gill, *SIAM J. Comput.*, 10:1(1981), 96–113.

7.3.11. [33] Show that for all $t \geq 2$, the set $C[t \log n, n^t, \infty]$ is P-isomorphic to $\{0\}^*$.

Comments. Source: [E. Allender and O. Watanabe, *Inform. Comput.*, 86 (1990), 160–178]. In this paper, the authors also use sets $C[t \log n, n^t, \infty]$ to study the equivalent classes of tally sets under various types of reductions. Compare with [S. Tang and R.V. Book, *Theoret. Comput. Sci.*, 81:1(1991), 35–47; R. Gavaldà and O. Watanabe, *SIAM J. Comput.*, 22(6) (1993), 1257–1275; H. Buhrman, E. Hemaspaandra, and L. Longpré, *SIAM J. Comput.*, 24:4(1995), 673–681].

7.3.12. [32] We say that A is *truth-table* reducible to B if there are functions $g_1, \ldots, g_m$ and a Boolean function f where y_i is true iff $g_i(a) \in B$, and $f(y_1, \ldots, y_m)$ is true iff $a \in A$. We consider only polynomial-time truth-table reductions where f and the g_i's are computable in polynomial time. It is clear that if A many-to-one reduces to B, then A also truth-table reduces to B; and if A truth-table reduces to B, then A also Turing reduces to B. Use time-space-bounded Kolmogorov complexity to construct a set D that is complete for E under polynomial Turing reduction but not complete for E under polynomial truth-table reduction.

Comments. Source: O. Watanabe, *Theoret. Comput. Sci.*, 54(1987), 249–265. Improved in [B. Fu, *SIAM J. Comput.*, 24:5(1995), 1082–1090].

7.3.13. (a) [35] Improve Theorem 7.3.5 by proving $\mathrm{BPP}^{R'} = \mathrm{P}^{R'}$, where $R' = \{x : C(x) \geq l(x)/i\}$ for i a positive integer.

(b) [33] Show that $\mathrm{PSPACE} \subseteq \mathrm{P}^R$ and $\mathrm{NEXP} \subseteq \mathrm{NP}^R$, with R as in Theorem 7.3.5.

(c) [O38]) Is there a larger complexity class $\mathrm{DTIME}(t)$, $\mathrm{NTIME}(t)$, or $\mathrm{DSPACE}(s)$, for some t or s, that is contained in P^R?

Comments. Hint: For Item (a), use the pseudorandom generators of R. Impagliazzo and A. Wigderson, *Proc. 38th IEEE Symp. Found. Comp. Sci.*, 1997, 220–229. Source: E. Allender, H.M. Buhrman, M. Koucký, D. van Melkebeek, and D. Ronneburger, *Proc. 43rd IEEE Symp. Found. Comp. Sci.*, 2002, 669–678. Item (c) is from E. Allender, email of March 18, 2006.

7.4 Instance Complexity

In computational complexity, it is traditional to study the intractability of a decision problem by treating a set collectively. With Kolmogorov complexity, it is possible also to study the complexity of individuals in a set, and this is called *instance complexity*.

Consider a partial program p that gives answers 1 (accept), 0 (reject), and $\perp$ (don't know). For a set A we will write $A(x)$ as the characteristic function of A, with $A(x) = 1$ iff $x \in A$. We say that a function p is *consistent* with a set A if $p(x) = A(x)$ when $p(x) \neq \perp$. The computing time of machine T on input y, using program p, is denoted by $\text{time}_T(p, y)$. In the definition below, A is an arbitrary set, possibly nonrecursively enumerable, and t is an arbitrary function, possibly nonrecursive.

Definition 7.4.1 Let T be a Turing machine. Given a set A and time bound t, define the (t-bounded) *instance complexity* of x with respect to T and A as

$$\text{ic}_T^t(x : A) = \min\{l(p) : T(p, x) \neq \perp, \text{ and } \forall y\ T(p, y) \neq \perp \text{ implies } \text{time}_T(p, y) \leq t(l(y)),\ T(p, y) = A(y)\},$$

and is ∞ if no such p exists.

By this definition, also $x \notin A$ can have instance complexity with respect to A. Intuitively, the instance complexity of a string x, with respect to a set A, measures the length of the shortest program that correctly decides whether x is in A. This program doesn't need to decide about other strings as long as it does not contradict A when it makes a decision. The goal is to identify the hard instances that make a language hard. We state an invariance theorem for instance complexity.

Theorem 7.4.1 **(Invariance of instance complexity)** *There exists a universal Turing machine U such that for every Turing machine T there is a constant c such that for all A and t and x,*

$$\text{ic}_U^{t'}(x : A) \leq \text{ic}_T^t(x : A) + c,$$

where $t'(n) = ct(n) \log t(n) + c$.

The proof of this invariance theorem is similar to that of time-bounded Kolmogorov complexity, and is left to the reader. Using the above invariance theorem, we fix a reference universal machine and drop the index U in $\text{ic}_U^t(x : A)$. We define $\text{time}_p(x) = \text{time}_U(p, x)$.

Example 7.4.1 The relationship between time-bounded Kolmogorov complexity and instance complexity is a fundamental question. Obviously, CD^t is a special case of ic^t complexity because

$$CD^t(x) = \text{ic}^t(x : \{x\}).$$

If c is a large enough constant and $t'(n) = ct(n)\log t(n) + c$, it is not difficult to prove the following for every set A (Exercise 7.4.1):

$$\mathrm{ic}^{t'}(x : A) \leq C^t(x) + c, \text{ and}$$
$$\mathrm{ic}^{t'}(x : A) \leq CD^t(x) + c.$$

It is clear that for some sets A, such as $\{0,1\}^*$, the above two inequalities are strict. This leads us to the following *instance complexity conjecture*: Let a function $t(n)$ be computable in time $t(n)$, function t' be as above, and the set A be recursive. If $A \notin \mathrm{DTIME}[t(n)]$, then there is a constant c and infinitely many x such that

$$\mathrm{ic}^t(x : A) \geq C^{t'}(x) - c.$$

Exercises 7.4.5, 7.4.7, and 7.4.8, contain solutions to various special cases of this conjecture. ◇

Instance complexity provides pleasant and simple characterizations for many fundamental complexity-theoretic properties.

Lemma 7.4.1 *A set A is in* P *if and only if there exist a polynomial t and a constant c such that for all x, we have* $\mathrm{ic}^t(x : A) \leq c$.

Proof. (Only if) If A is in P, then for every x the polynomial-time program that decides A is trivially a consistent program for A and decides whether $x \in A$. This gives constant instance complexity for every string x.

(If) By assumption, there is a constant c, such that for every x, we have $\mathrm{ic}^t(x : A) \leq c$. Let B be the set of all programs, consistent with A, of size at most c, with running time bounded by $t(n)$. Decide whether $x \in A$ by simulating all the programs in B and accept x if and only if some program in B accepts x. □

Definition 7.4.2 Let A be a recursive set. An infinite set C (not necessarily a subset of A) is called a *polynomial complexity core* for A if for every total program p that decides A and polynomial t, we have $\mathrm{time}_p(x) > t(l(x))$ for all but finitely many x in C.

Lemma 7.4.2 *A set C is a polynomial complexity core for A if and only if for every polynomial t and constant c we have* $\mathrm{ic}^t(x : A) > c$ *for all but finitely many x in C.*

Proof. (If) Assume that there are infinitely many x in C such that $\mathrm{ic}^t(x : A) \leq c$ for some polynomial t and constant c. Let B be the set of

programs, running in time t, that are consistent with A and of length less than c. Then there are infinitely many elements x in C that we can accept in polynomial time by simulating all programs in B simultaneously in dovetailing style. Thus, C cannot be a polynomial complexity core for A.

(Only if) Assume that C is not a polynomial complexity core for A. Then there is a total program p for A and a polynomial t such that for infinitely many x in C we have $\text{time}_p(x) \leq t(l(x))$. The program p is consistent with A and it accepts infinitely many x in time $t(l(x))$. We modify p by adding a step counter to check for $t(n)$ to it. When the computation exceeds $t(n)$ steps, we halt p. The modified p is consistent with A, runs in polynomial time, and decides correctly infinitely many x. Thus, for infinitely many x, we have $\text{ic}^{t'}(x : A) \leq c$. □

Among the strings of various instance complexities, one class is particularly interesting. This is the class of strings of logarithmic instance complexity computable in polynomial time.

Definition 7.4.3 IC[log, poly] is the class of sets A for which there exists a polynomial t and a constant c such that for all x, $\text{ic}^t(x : A) \leq c \log l(x) + c$.

The SAT decision problem, Definition 1.7.9 on page 40, is the following: Given a Boolean formula $\phi(x_1, \ldots, x_k)$ in conjunctive normal form of binary description length n, we want to decide whether there is a truth assignment to the variables $x_1, \ldots, x_k$ that makes the formula true. We always assume that k and n are polynomially related. The set SAT is the set of ϕ's for which there is a truth assignment that makes ϕ true.

Theorem 7.4.2 *If* SAT $\in$ IC[log, poly], *then* NP = P.

Proof. Assume that SAT $\in$ IC[log, poly]. Then for some polynomial t and constant c, we have

$$\text{ic}^t(\phi : \text{SAT}) \leq c \log l(\phi) + c, \tag{7.8}$$

for all ϕ. We treat each input as a conjunctive normal form ϕ. If the input is of wrong format, then it can be rejected right away. Thus, for every ϕ, there is a program p of length $c \log l(\phi) + c$ such that p decides whether $\phi \in$ SAT correctly and p is consistent with SAT for all other inputs. We show that if Equation 7.8 is true, then SAT can be decided in polynomial time.

Given input $\phi(x_1, \ldots, x_k)$, let $l(\phi(x_1, \ldots, x_k)) = n$. First, set $A := \{p : l(p) \leq c \log n + c\}$. Clearly, $d(A)$ is of polynomial size. By Equation 7.8, for every input of length less than or equal to n, there is a program in A running in polynomial time $t(n)$ that decides correctly whether this particular input is in SAT, and on other inputs it makes decisions consistent

with SAT. In particular, there is a program p_0 running in polynomial time $t(n)$ that decides correctly whether $\phi(x_1, \ldots, x_k) \in \text{SAT}$. Moreover, p_0 is consistent with SAT on all other inputs. The program p_0 may just output $\perp$ signs for the latter two inputs.

In the procedure below we will, in a polynomial number of steps, either decide $\phi(x_1, \ldots, x_k) \in \text{SAT}$ correctly or delete at least one program p, with $p \neq p_0$ and p inconsistent with SAT, from A. Repeating this procedure $d(A) - 1$ times, we will either decide $\phi(x_1, \ldots, x_k)$ correctly or find p_0 as the last remaining element by elimination. Since $d(A)$ is polynomial, the entire process takes polynomially many steps. The procedure is as follows:

Step 1. For all $p \in A$, simulate p for $t(n)$ steps on input $\phi(x_1, \ldots, x_k)$.

- **If** no program p rejects $\phi(x_1, \ldots, x_k)$ **{**implying that p_0 has accepted $\phi(x_1, \ldots, x_k)$; some programs may output $\perp$**}** **then** we also accept $\phi(x_1, \ldots, x_k)$ and **exit** the procedure.
- **If** no p accepts $\phi(x_1, \ldots, x_k)$ **{**implying that p_0 must have rejected $\phi(x_1, \ldots, x_k)$**}** **then** reject $\phi(x_1, \ldots, x_k)$ and **exit** the procedure.

Step 2. If some program in A accepts $\phi(x_1, \ldots, x_k)$ and some other program in A rejects $\phi(x_1, \ldots, x_k)$ **then for** $i := 1, \ldots, k$ **do**

Suppose the binary values $b_1, \ldots, b_{i-1}$ are determined in the previous loops and $p \in A$ accepts $\phi = \phi(b_1, \ldots, b_{i-1}, x_i, \ldots, x_k)$.
Simulate all programs in A with inputs

$$\phi_0 = \phi(b_1, \ldots, b_{i-1}, 0, x_{i+1}, \ldots, x_k),$$
$$\phi_1 = \phi(b_1, \ldots, b_{i-1}, 1, x_{i+1}, \ldots, x_k).$$

{Formulas such as ϕ, ϕ_0, and ϕ_1 are all different instances**}**
If neither of the above inputs is accepted by some program in A **{**this means $\phi(b_1, \ldots, b_{i-1}, x_i, \ldots, x_k) \notin \text{SAT}$, because there are programs $q_0, q_1 \in A$ that are consistent with SAT and decide correctly whether $\phi_0, \phi_1 \in \text{SAT}$; consequently p is not consistent with SAT and $p \neq p_0$**}** **then** delete p from A and **exit** the procedure.
If one of the inputs, say with $x_i = 0$, is accepted by some program in A **then** set $b_i := 0$.

The process either exits in Step 1 deciding ϕ, or exits in Step 2 with a program p inconsistent with SAT deleted from A, or ends with a truth assignment $(b_1, \ldots, b_k)$. In the latter case, we can check (in polynomial time) whether the truth assignment $(b_1, \ldots, b_k)$ really satisfies

$\phi(x_1, \ldots, x_k)$. If it does, then we accept $\phi(x_1, \ldots, x_k)$. Otherwise, we delete p from A, since it is inconsistent with SAT. (In this way, we never delete programs consistent with SAT.) Therefore, in Step 2, we finish either with a satisfying assignment or by deleting an inconsistent program from A.

Repeating the procedure at most $d(A) - 1$ times, we are guaranteed to decide ϕ, or to find a satisfying assignment, or we have only p_0 left in A at the last step. □

Exercises

7.4.1. [25] Let $t(n)$ be computable in time $t(n)$, $t'(n) = ct(n) \log t(n) + c$, for some constant c. For set A, string x, and some constant c, prove:

(a) $\mathrm{ic}^{t'}(x : A) \leq C^t(x) + c$, and

(b) $\mathrm{ic}^{t'}(x : A) \leq CD^t(x) + c$.

Comments. Item (a) and the next three exercises are from [P. Orponen, K. Ko, U. Schöning, and O. Watanabe, *J. Assoc. Comp. Mach.*, 41(1994), 96–121]. Item (b) was suggested by L. Fortnow.

7.4.2. [30] The proof of Theorem 7.4.2 depends on the so-called self-reducibility of the SAT problem. A set is *self-reducible* if the membership question for an element can be reduced in polynomial time to the membership question for a number of *shorter* elements. For example, SAT is self-reducible, since an arbitrary Boolean formula $\phi(x_1, x_2, \ldots, x_n)$ is satisfiable if and only if at least one of the two shorter Boolean formulas $\phi(0, x_2, \ldots, x_n)$ or $\phi(1, x_2, \ldots, x_n)$ is satisfiable. Prove the following generalization of Theorem 7.4.2: If a set is self-reducible and it is in IC[log, poly], then it is also in P.

Comments. See the survey by D. Joseph and P. Young in [*Complexity Theory Retrospective*, A. Selman, ed., Springer-Verlag, 1990, pp. 82–107] for more information on self-reducibility.

7.4.3. [32] The class P/poly is defined in Exercise 7.2.6 on page 560. Use that exercise to analogously define the class $\mathrm{P/log} = \bigcup_{c>0} \mathrm{P}/c \log n$. Show that P/log is properly contained in IC[log, poly], which is in its turn properly contained in P/poly.

7.4.4. [31] There are sets with hard instance complexity everywhere. In particular, prove that there is a set A computable in $2^{O(n)}$ time such that for some constant c and for all x,

$$\mathrm{ic}^{\exp}(x : A) \geq C^{\exp'}(x) - 2 \log C^{\exp'}(x) - c,$$

where $\exp(n) = 2^n$ and $\exp'(n) = O(n2^{2n})$.

7.4.5. [38/O43] Consider Exercise 7.4.4 on page 575 and Lemma 7.4.2. Let us say that a set A has *p-hard instances* if for every polynomial t there exists a polynomial t' and a constant c such that for infinitely many x we have $\mathrm{ic}^t(x : A) \geq C^{t'}(x) - c$.

(a) (Open) Prove or disprove: Every recursive set $A \notin \mathrm{P}$ has p-hard instances. This may be regarded as a polynomial version of the instance complexity conjecture in Example 7.4.1.

(b) Every recursive tally set $A \notin \mathrm{P}$ has p-hard instances.

(c) Prove the following claim: Let a recursive set A be NP-hard with respect to the polynomial-time Turing reduction in which the length of a query is not shorter than a fixed polynomial of the length of the input. Then A has p-hard instances unless $A \in \mathrm{P}$. In particular, this holds for all natural NP-hard problems.

(d) (Open) We can also state a *CD* version of Item (a). Prove or disprove: For every recursive set $A \notin \mathrm{P}$ and every polynomial t there is a polynomial t' and a constant c such that for infinitely many x, $\mathrm{ic}^t(x : A) \geq CD^{t'}(x) - c$.

Comments. Source: L. Fortnow and M. Kummer, *Theoret. Comput. Sci. A*, 161(1996), 123–140. They have also shown that the instance complexity conjecture, Item (a), and Item (d), all fail relative to some oracles. Item (a) also holds relative to some oracle.

7.4.6. [35] Let t be a recursive time bound. There is a recursive set A such that $f(x) = \mathrm{ic}^t(x : A)$ is not recursive.

Comments. This result, due to L. Fortnow and M. Kummer [*Ibid.*], was originally conjectured by P. Orponen, K. Ko, U. Schöning, and O. Watanabe, *Ibid.*

7.4.7. [33] In Definition 7.4.1 on page 571, when we allow t to be an arbitrary finite time, we will remove t from ic^t and simply write ic.

(a) Let $R = \{x : C(x) \geq l(x)\}$. We know that R is infinite and that it contains at least one string of each length. Show that there is a constant c such that for every x in R we have $\mathrm{ic}(x : R) \geq C(x) - c$. That is, R contains only hard instances.

(b) Strings with high Kolmogorov complexity are individually hard to recognize by bounded computations. We say that a set C is dense if there is an ϵ such that $d(C \bigcap \Sigma^n) \geq 2^{\epsilon n}$. Let set A be complete for the class $\bigcup_{c \geq 0} \mathrm{DTIME}[2^{cn}]$ with respect to polynomial-time reductions. Show that there is a dense subset $C \subseteq A$ such that for every nondecreasing polynomial $t(n) \geq n \log n$, for each $x \in C$, $\mathrm{ic}^t(x : A) \geq C^t(x) - c$, for some constant c.

(c) For every recursively enumerable set A with an infinite complement, there exists a constant c such that for infinitely many $x \notin A$ we have $\mathrm{ic}(x : A) \geq C(x) - c$.

(d) Show that there is a recursively enumerable set A with $\mathrm{ic}(x : A) \geq l(x)$ for infinitely many $x \in A$.

Comments. Hint for Item (a). Let $L(p)$ be the set of strings accepted by p in the ic sense. Observe that there is a constant d such that for every total p for which $L(p) \subseteq R$, and for every $x \in L(p)$, we have $l(x) \leq l(p) + d$. (This is similar to Corollary 2.7.2 on page 176.) Now the result follows easily using the definition of ic. Item (b) may be regarded as another special case of the instance complexity conjecture in Example 7.4.1. Items (a)–(c) are from [H.M. Buhrman and P. Orponen, *J. Comput. System Sci.*, 53:2(1996), 261–266]. Item (d) and an extensive study of related topics can be found in [M. Kummer, *SIAM J. Comput.*, 25:6(1996), 1123–1143].

7.4.8. [39] Use the definition of ic in Exercise 7.4.7.

(a) Show that if $\mathrm{ic}(x : A) \leq \log C(x) - 1$ for all but finitely many x, then A is recursive.

(b) There is a nonrecursive recursively enumerable set A and a constant c such that $\mathrm{ic}(x : A) \leq \log C(x) + c$ for all but finitely many x.

Comments. Item (b) resolves an open question in the first edition of this book: it refutes the unbounded version of the *instance complexity conjecture* in Example 7.4.1. Originally proposed by P. Orponen, K. Ko, U. Schöning, and O. Watanabe, *Ibid.* The solution is due to M. Kummer [*Ibid.*], where Item (a) is attributed to J.T. Tromp.

7.5 Kt and Universal Search

It is meaningful to consider the *age* of strings. Loosely stated, the age of a string corresponds to the time we need to generate that string starting with a program of constant size. In this sense, the age of a random string x should be at least $2^{l(x)}$, considering the number of steps needed to generate x by enumerating all strings in length-increasing lexicographic order. This is also the expected time for a constant-size probabilistic program to generate x by fair coin flips. For random x, this yields $\mathrm{age}(x) = \Omega(2^{l(x)})$. The case of nonrandom strings is more interesting. Generate programs length-increasing lexicographically and simulate them dovetailing style. This leads to a definition of the *age* as

$$\mathrm{age}(x) = \min_p \{2^{l(p)} t : U(p) = x \text{ in } t \text{ steps}\}.$$

In this way, age is dominated by the total time needed for a string to appear out of nothing, enumerated by a constant-size program. Taking the

logarithm, we arrive at Levin's Kt complexity: $Kt(x) = \log \text{age}(x)$. Let us now define Kt complexity formally. We use the monotone machines of Definition 4.5.2 on page 298, or the prefix machines of Example 3.1.1 on page 201. The two models are completely equivalent for use in Definition 7.5.1; each model consists of a Turing machine with a one-way read-only input tape, a one-way write-only output tape, and a work tape. Initially, the input tape contains a one-way infinite binary sequence.

Definition 7.5.1 Let the universal monotonic machine U scan an initial input segment p before it prints x (without necessarily halting after it does so). Let $t(p, x)$ be the number of steps taken until x is printed. Then Kt is defined by

$$Kt_U(x) = \min_p\{l(p) + \log t(p, x)\}.$$

Since an invariance theorem such as Theorem 2.1.1, page 105, can be proved in the standard way, we will drop the subscript U and write Kt. For $x \in \{0,1\}^*$, if $x \in K[m - \log t, t, \infty]$, with m minimal, then x has Kt complexity $Kt(x) = m$. Namely, x can be computed (and hence enumerated) by a self-delimiting program of length $m - \log t$ in t steps.

7.5.1 Universal Optimal Search

There is a *universal search* method that will solve all problems of a certain class of inverting problems in time that is optimal but for a multiplicative constant. Despite its simplicity, the idea of universal optimal search is a powerful one. Let $T_1, T_2, \ldots$ be a standard enumeration of prefix machines, and let $\phi_1, \phi_2, \ldots$ be the corresponding enumeration of partial recursive functions. If ϕ is a partial recursive function and $\phi(y) = x$, then y is a *ϕ-witness* for x.

Definition 7.5.2 An algorithm A *inverts problem* ϕ if given some x in the range of ϕ, algorithm A computes a ϕ-witness y for x and checks that $\phi(y) = x$. Algorithm A diverges outside the range of ϕ.

Example 7.5.1 Many computational problems consist in finding feasible algorithms to invert functions. Given a composite natural number, we are required to find a factorization. Once a splitting (partial factorization) is found, we check whether it is correct. A solution to this inversion problem does not solve the corresponding decision problem of whether a natural number is prime.

Given a satisfiable Boolean formula, we want to find a truth assignment that satisfies it, Definition 1.7.9. Once an assignment is found, we check whether it makes the formula true. A solution to this inversion problem does not imply a solution to the corresponding decision problem SAT of whether a given Boolean formula is satisfiable.

To decide whether we can node color a given graph on n nodes with k colors such that no edge connects two nodes of the same color is an NP-complete problem. We know that we can color the graph with n colors. Hence, there is a least number of colors χ ($1 \leq \chi \leq n$) to color the graph. This χ is the chromatic number of the graph. Suppose we have an algorithm that finds a witness k-coloring if there is one in time $t(n)$. We can run this algorithm for $1 \leq k \leq n$ dovetail style. The dovetailed algorithm finds k_0 together with a k_0-coloring in $nt(n)$ steps. This solves the k-coloring problem for all k. The drawback is that we need to know $t(n)$ to make the actual decision. A polynomial-time solution to the inversion problem of finding a graph k-coloring, together with its running time t, also gives a polynomial-time solution for the corresponding NP-complete decision problem. ◇

To solve inverting problems naively usually requires us to search through exponentially many candidate witnesses. But we can also take the universal optimal search algorithm below.

Lemma 7.5.1 *Let ϕ be an inverting problem. If there is an algorithm A that inverts ϕ in time $t(n)$, then the SIMPLE algorithm below inverts ϕ in time $ct(n)$, where c is a constant depending only on A.*

Proof. We describe Algorithm SIMPLE. Run all machines T_i one step at a time according to the following scheme: T_1 every second step, T_2 every second step in the remaining unused steps, T_3 every second step in the remaining unused steps, and so on, that is, according to the sequence of indices,

$$1213121412131215121312141213121612\ldots .$$

If T_k inverts ϕ on x in t steps, then this procedure will do the same in $2^k t + 2^{k-1}$ steps. Choosing $c = 2^{k+1}$ proves the lemma. □

A similar universal optimal search procedure A was developed by L.A. Levin using Kt. It seems to have the advantage that the multiplicative constant $c = 2^{k+1}$ in Lemma 7.5.1 is reduced to $2^{K(T_k)+1}$, but closer inspection shows that SIMPLE already attains this, see Example 7.5.2, and in fact can do much better, Exercise 7.5.3 on page 582.

Theorem 7.5.1 *Let ϕ be an inverting problem. If there is an algorithm A that inverts ϕ in time $t(n)$, then the SEARCH algorithm below inverts ϕ in time $ct(n)$, where c is a constant depending only on A.*

Proof. We need a conditional and modified Kt. Define $Kt'(w|x,\phi) = \min\{l(p) + \log t(p,w) : \text{given } x, \text{ program } p \text{ prints } w \text{ and tests whether}$

$\phi(w) = x$ in time $t(p,x)\}$. Given x and ϕ, algorithm SEARCH will generate all strings w in order of increasing $Kt'(w|x,\phi)$, and try whether $\phi(w) = x$ until it has found a witness that inverts ϕ.

The algorithm SEARCH is as follows: The universal prefix machine U lexicographically runs all self-delimiting programs p (of length less than i) for $2^i 2^{-l(p)}$ steps in phase i, $i = 1, 2, \ldots$, until it has inverted ϕ on x.

Claim 7.5.1 All strings w of Kt' complexity less than or equal to k are generated and tested in 2^{k+1} steps.

Proof. If $Kt'(w|x,\phi) \leq i$, then $l(p) + \log t(p,x) \leq i$. That is, $t(p,x) \leq 2^{i-l(p)}$, which is precisely the allotted time for this program in phase i. Since U is a prefix machine, we have by the Kraft inequality, Theorem 1.11.1 on page 76, that $\sum 2^{-l(p)} \leq 1$, with the sum taken over all p for which the computation of U with input p terminates. Consequently,

$$\sum_{1\leq i\leq k} \sum_{0<i-l(p)} 2^{i-l(p)} \leq \sum_{U(p)<\infty} 2^{-l(p)} \sum_{1\leq i\leq k} 2^i \leq 2^{k+1}.$$

□

Let $m = \min\{Kt'(w|x,\phi) : w \text{ is a } \phi\text{-witness for } x\}$. Suppose there exists a prefix machine T that inverts ϕ on x in time $t(n)$ with $n = l(x)$. By definition, $m \leq Kt'(T|x,\phi)$. The SEARCH algorithm inverts ϕ on x in 2^{m+1} steps by the claim. By definition, $Kt'(T|x,\phi) \leq K(T) + \log t(n)$. Therefore, SEARCH uses a number of steps of at most

$$2^{K(T)+1} t(n).$$

Setting $c = 2^{K(T)+1}$, we prove the theorem. □

Example 7.5.2 Let T_k be an inversion algorithm running in time $t(n)$. The SEARCH algorithm will use time $2^{K(k)+O(1)}t(n)$ for the same inversion problem, which is at most $O(k(\log k)^2)t(n)$. The SIMPLE algorithm uses $2^{k+1}t(n)$ time—simulating T_k's steps. If the inversion algorithm T_k for a given inversion problem is very simple, for example, $K(T_k) = \log\log k$, then SEARCH runs in time $O(\log k)t(n)$. This is much better than the time used by SIMPLE in simulating T_k's steps, which stays at $2^{k+1}t(n)$. Or can SIMPLE do better? It turns out that actually SIMPLE does as well as SEARCH, an observation first made by M. Hutter. The reason is that the SEARCH algorithm is executed by some Turing machine, T_s say. So, whatever SEARCH does in time $t(n)$, SIMPLE will also do in time $2^{s+1}t(n)$. Therefore, SIMPLE does every task SEARCH does, and in the time of the same order of magnitude. This type of approach is improved further by FASTSEARCH; see Exercise 7.5.3 on page 582. If the latter

algorithm is executed by Turing machine $T(s')$, then SIMPLE simulates *every* inversion algorithm running in time $t(n)$ in just $2^{s'+1}5t(n)$ steps (ignoring gigantic additive terms depending on the inversion algorithm and FASTSEARCH but not on the input). ◇

Example 7.5.3 The enumeration used in the above optimal search algorithm leads to another variant of Kt. Let a monotonic machine T enumerate a sequence of strings. The *height* $h_T(x)$ of x with respect to T is the logarithm of the shortest time by which T outputs x, without necessarily halting. Since an invariance theorem such as Theorem 2.1.1 can be proved in the usual manner, we drop the subscript and write $h(x)$. It is not difficult to show that $h(x) = Kt(x) + O(1)$.

For an inverting NP problem, let $t(x)$ be the time it takes to find a witness by the optimal algorithm for input x. Then it easy to see that

$$\log t(x) = \min Kt'(w|x) + O(1),$$

where the minimum is taken over all witnesses w for x. ◇

Exercises

7.5.1. [30] Show that symmetry of information, Theorem 2.8.2 on page 190, does not hold for Kt.

Comments. Hint: For each c and large n, by diagonalization find strings $x, y \in \{0,1\}^n$ such that $Kt(x) > \frac{n}{2}$ and $Kt(y|x) > \frac{n}{2}$, but $Kt(xy) \leq \frac{n}{2} + O(\log n)$. Source: D. Ronneburger, *Kolmogorov complexity and derandomization*, Ph.D. thesis, Rutgers, 2004. This result provides a sharp contrast with Exercise 7.1.12 and Exercise 7.1.13.

7.5.2. [27] Similar to Definition 7.5.1, one can define the Ct version of Kt. Below we set $n = l(x)$.

(a) Show that $Ct(x) \leq s(n)$ implies $x \in C[s(n), 2^{s(n)}, \infty]$, and the last formula implies $Ct(x) \leq 2s(n)$.

(b) For a context-free language L, let $C_L(n) = \min\{Ct(x) : x \in L^{=n}\}$. Show that $C_L(n) = O(\log n)$.

(c) Define $C^L(n) = \max\{Ct(x) : x \in L^{=n}\}$. Show that L is P-printable iff L is in P and $C^L(n) = O(\log n)$.

(d) Show that $C_L(n) = O(\log n)$ for all L in P iff $C_L(n) = O(\log n)$ for all L in NP.

(e) Every nondeterministic exponential-time computable predicate L is computable in deterministic exponential time iff $C_L(n) = O(\log n)$.

Comments. Source: [E. Allender, In: *Kolmogorov Complexity and Computational Complexity*, O. Watanabe, ed., Springer-Verlag, 1992, pp. 4–22]. This reference contains applications of $C_L(n)$ and $C^L(n)$ in random oracle constructions, pseudorandom generators, and circuit complexity.

7.5.3. [31] Show that there there is an algorithm FASTSEARCH that does the following: Let $A : X \to Y$ be a given algorithm or specification. Let B be an algorithm, computing provably the same function as A with computation time provably upper bounded by the function $t_B(x)$. Let time $t_B(x)$ be the time needed to compute the time bound $t_B(x)$. Then the algorithm FASTSEARCH computes $A(x)$ in time at most $5t_B(x) + d_B\text{time } t_B(x)+c_B$, with constants d_B and c_B depending on B but not on x. Neither B, t_B, nor the mentioned proofs need to be known in advance for the construction of FASTSEARCH.

Comment. The algorithm FASTSEARCH is an optimized version of SEARCH, where the proportionality constant of $2^{K(k)+O(1)}$ of the latter is replaced by just 5, but at the cost of truly gigantic additive terms depending on k but not on x. By the reasoning in Example 7.5.2, there is a fixed constant c such that the trivial SIMPLE algorithm simulates every inversion problem running in $t(n)$ steps in at most $c \cdot t(n)$ steps ignoring additive terms depending on the inversion problem but not on the input. Source: M. Hutter, *Int. J. Found. Comput. Sci.*, 13:3 (2002), 431–443.

7.6 Time-Limited Universal Distributions

The universal distribution $\mathbf{m}$ was applied to the average-case analysis of algorithms in Section 4.4 and learning in Section 5.3.3. A drawback of $\mathbf{m}$ is that it is not computable. It is not difficult to scale the entire theory down to a more feasible domain. For each time bound $t(n)$ we construct a function $t'(n) = O(nt(n)\log(nt(n)))$ such that $\mathbf{m}^{t'}$ is computable in time $nt(n)2^n$ and multiplicatively dominates every probability mass function P with a distribution function P^* that is computable in time $t(n)$.

It is convenient to formulate this section in terms of distribution functions $P^* : \{1, 2, \ldots\} \to [0, 1]$, where $P^*(x)$ is the summed probability of all elements not exceeding x. Its *probability mass function* $P(x) = P^*(x) - P^*(x-1)$ is the usual probability of x. Below, all time functions t are time-constructible, that is, functions from natural numbers to natural numbers with the property that $t(n)$ can be constructed from n by a Turing machine in time $O(t(n))$.

Definition 7.6.1 An integer function f is computable in time $t(n)$ if there is a Turing machine T that on input x computes output $f(x)$ in at most $t(n)$ steps, where $n = l(x)$. A distribution P^*, which is almost always noninteger, is computable in time $t(n)$ if there is a Turing machine T that on input x

and a positive integer k computes a rational number y in at most $t(n+k)$ steps, where $n = l(x)$, such that $|P^*(x) - y| \leq 1/2^k$.

Let U be the reference universal prefix machine U. The t-time bounded version of $K(x)$ is defined similarly to the one for $C(\cdot)$ in Definition 7.1.2 on page 533:

$$K^t(x) = \min\{l(p) : U(p) = x \text{ in } t(n) \text{ steps}\}.$$

For every t and x of length n we have $K^t(x) \leq n + 2\log n + O(1)$. In the limit, as t runs through a sequence of functions $t_1, t_2, \ldots$ with $\lim_{i\to\infty} t_i(n) = \infty$ for every fixed n, the limiting value of $K^t(x)$ is $K(x)$.

Definition 7.6.2 The t-time-bounded version of $\mathbf{m}$, denoted by $\mathbf{m}^t$, is defined by

$$\mathbf{m}^t(x) = 2^{-K^t(x)},$$
$$\mathbf{m}^{*t}(x) = \sum_{y\leq x} \mathbf{m}^t(y).$$

Note that for all t and x, with $n = l(x)$, we have $\mathbf{m}^t(x) \leq 2^{-n-2\log n+O(1)}$, and in the limit, as t runs through a sequence of functions $t_1, t_2, \ldots$ with $\lim_{i\to\infty} t_i(n) = \infty$ for every fixed n, $\mathbf{m}^t(x)$ goes to $\Theta(\mathbf{m}(x))$.

Consider the class of probability mass functions P with a corresponding distribution $P*$ that is t-time computable in the sense of Definition 7.6.1. We want to show that $\mathbf{m}^{t'}(x)$ multiplicatively dominates all probability mass functions in the class, for some $t'(n) = O(nt(n)\log(nt(n)))$.

We do not know whether $\mathbf{m}^t(x)$ and $\mathbf{m}^{*t}(x)$ themselves are t-time computable. The best we can now achieve is to compute the probability mass function $\mathbf{m}^t(x)$ in $O(t(n)n^2 2^n)$ time ($l(x) = n$), namely, computing $K^t(x)$ by simulating all programs of length up to $n + 2\log n + O(1)$, and determining the length of the shortest program that halts with output x in $t(n)$ steps. Similarly, we compute $\mathbf{m}^{*t}(x)$ in $O(t(n)n^2 2^{2n})$ time by computing the sum

$$\mathbf{m}^{*t}(x) = \sum_{y\leq x} \mathbf{m}^t(y).$$

Theorem 7.6.1 *The distribution $\mathbf{m}^{t'}$, for some $t'(n) = O(nt(n)\log(nt(n)))$, is* universal *for the class of probability mass functions P with a t-time computable distribution P^*, that is, $\mathbf{m}^{t'}$ multiplicatively dominates every P in that class as follows. There exists a constant c_P such that $c_P\mathbf{m}^{t'}(x) \geq P(x)$ for every x of length n, where $\log c_P = K^{t'}(P^*) + O(1)$ depends on P^* and t but not on x.*

Proof. The theorem follows immediately from the following claim.

Claim 7.6.1 If the distribution P^* is computable in time $t(n)$, in the sense of Definition 7.6.1, then there is a constant c_P as in the theorem such that for all x with $l(x) = n$,

$$K^{t'}(x) \leq \log \frac{1}{P(x)} + \log c_P.$$

Proof. Without a time bound, a proof similar to that of the optimality of a constructive version of the Shannon–Fano code would be sufficient, as in the proof of Theorem 4.3.1 on page 267. But we have to deal with the time bound here.

Divide the real interval $[0, 1)$ into subintervals such that the code word $c(x)$ (constructed below) for a source word x occupies $[P^*(x-1), P^*(x))$. There is room enough, since $\sum_x P(x) \leq 1$. The *binary interval* determined by the finite binary string r is the half-open interval $[0.r, 0.r + 2^{-l(r)})$ corresponding to the set of reals (cylinder) Γ_r consisting of all reals $0.r \ldots$. If Γ_r is the greatest binary interval contained in $I_x = [P^*(x-1), P^*(x))$, then x is encoded as $c(x) := r$. It is easy to show that the greatest binary interval I_r in an interval I_x of $[0, 1)$ has size at least one-fourth of I_x. Since length $l(I_x) = P(x)$, it follows that $l(c(x)) \leq \log 1/P(x) + 2$ in bits.

We want to give encoding and decoding algorithms that are efficient. The encoding algorithm is trivial. Since P^* is computable in time $t(n)$, in the sense of Definition 7.6.1, given source word x ($l(x) = n$), the code word $c(x)$ can be computed from $P^*(x-1)$ and $P^*(x)$ in altogether $O(t(n))$ time. In order to compute c^{-1}, the decoding function, given a code word $c(x)$, we proceed as follows:

Step 1. Set $k := 1$.

Step 2 (Doubling). Repeat set $k := 2k$ **until** $\Gamma_{c(x)}$ falls in or to the right of interval $[P^*(k-1), P^*(k))$. Set $l := k/2$ and $u := k$.

Step 3 (Binary search). Set $m := (u + l)/2$. **If** $\Gamma_{c(x)}$ is in $[P^*(m-1), P^*(m))$ **then** return $x := m$ **else** set $u := m$ if $\Gamma_{c(x)}$ is to the left of $P^*(m-1)$ and set $l := m$ if $\Gamma_{c(x)}$ is to the right of $P^*(m)$.

This *decoding algorithm* is similar to a binary search, and it takes at most $\sum_{i=0}^{n} O(t(i)) = O(nt(n))$ time to find x. By construction $l(c(x)) \leq \log 1/P(x) + 2$. This completes our encoding/decoding of x using distribution P.

We can reconstruct x in $O(nt(n))$ steps from the following description:

- A description of this discussion (including the decoding algorithm) in $O(1)$ bits;
- a self-delimiting program q to compute $P^*(x)$ $(l(x) = n)$ in time $t(n)$; and
- the self-delimiting code word $c(x)$.

This description takes $l(c(x)) + l(q) + O(1)$ bits and can be interpreted and executed by a particular prefix Turing machine in time $O(nt(n))$. By Equation 7.1 on page 536, the reference universal prefix Turing machine (the self-delimiting machine variant of the prefix Turing machines) can do this in time $t'(n) = O(nt(n)\log(nt(n)))$. Since $K^{t'}(x)$ is the length of a shortest program from which x can be reconstructed in $t'(n)$ steps, setting $\log c_P = l(q) + O(1)$, we have $K^{t'}(x) \leq \log 1/P(x) + \log c_P$. Program q computing P^* runs in time $t(n)$. We can also compute P^* using a shortest program q' running in time $O(t'(n))$. Then $l(q') \leq l(q)$ and $l(q')$ equals $K^{t'}(P^*)$. Therefore we can set $\log c_P = l(q') + O(1) = K^{t'}(P^*) + O(1)$, □ □

Example 7.6.1 In many algorithms, we consider only inputs in a finite set $A \subseteq \mathcal{N}$. We can precompute the time-limited conditional version $\mathbf{m}^t(x|x \in A)$ in the form of an interval representation of a table once and for all, and use it to sample by means of a sequence of fair coin flips analogous to Example 4.4.3 on page 293. Such a table needs to be precomputed only *once,* and being available, can be used repeatedly by *every* randomized algorithm that uses this probability mass function. An application of this is in simple pac-learning, Section 5.3.3. As an example take $A = \{0,1\}^n$. We show how to compute $\mathbf{m}^t(x|l(x) = n)$. Divide $[0,1)$ into 2^n half-open disjoint intervals I_y with

$$l(I_y) = \frac{\mathbf{m}^t(y)}{\mathbf{m}^t(x+1) + \cdots + \mathbf{m}^t(x+2^n)}$$

for $y = x+1, \ldots, x+2^n$. Then $\mathbf{m}^t(y|l(y) = n) = l(I_y)$ and $\bigcup_y I_y$ equals $[0,1)$. ◇

Example 7.6.2 More generally, the entire theory of simple pac-learning can be reformulated in terms of t-time limited simple distributions, $\mathbf{m}^{t'}$, and $K^{t'}$, for some $t'(n) = O(nt(n)\log(nt(n)))$. Fix a low t, such as $t(n) = O(n^2)$, and precompute once and for all the finite $\mathbf{m}^{t'}(x)$ table for x ranging over the finite set that is required. Let $\mathcal{C}$ be a concept class that is polynomially learnable under $\mathbf{m}^{t'}$. For example, suppose that $\mathcal{C}$ is the class of n^2-simple DNF (analogous to simple DNF in Exercise 5.3.3 on page 381, using K^{n^2}). We can use the table, together with random coin flips as

explained in Example 4.4.3 on page 293, to polynomially learn n^2-simple DNF under all n^2-simple distributions. ◇

Some further developments are given in Exercise 7.6.5 on page 588 and Exercise 7.6.6 on page 589. *Computational depth* is defined and studied in [L. Antunes, L. Fortnow, D. van Melkebeek, and N.V. Vinodchandran, *Theoret. Comput. Sci.*, 354(2006), 391–404; L. Antunes and L. Fortnow, *Electr. Coll. Comput. Complexity*, 144(2005); L. Antunes, L. Fortnow, A. Pinto, and A. Souto, *Proc. 22nd IEEE Conf. Comput. Complexity*, 2007, 46–51]. Without explaining unknown terminology: For t a time-constructible function, define *t-bounded computational depth* by $\mathrm{cd}^t(x|y) = K^t(x|y) - K(x|y)$. Then, random and simple strings have small depth but there do exist strings of depth nearly n. For every satisfiable formula ϕ, we can find an assignment of ϕ probabilistically in time $\min_{t,a:a \text{ satisfies } \phi}\{2^{\mathrm{cd}^t(a|\phi)+O(\log t)}\}$. Under a reasonable derandomization assumption, an algorithm runs in time polynomial on average in L.A. Levin's sense, explained in Exercise 7.6.6 on page 589, iff for some polynomial p, the algorithm on input x runs in time $p(l(x))2^{\mathrm{cd}^{p(x)}}$ in the worst case. A set A is cd-*shallow* if for some k and for every initial segment x of the characteristic sequence of A, we have $\mathrm{cd}^{\log^k}(x) \leq k \log l(x)$. Random and sparse sets are both shallow. Every computable set B that polynomial-time Turing reduces to a shallow set A is in P/poly.

Exercises

7.6.1. [35] Call a probability distribution $P : \mathcal{N} \to \mathcal{R}$ *malign* for a class of algorithms if each algorithm in the class runs in P-average time (space) equal to worst-case time (space). In Section 4.4 it was shown that $\mathbf{m}$ is malign for the recursive algorithms. As usual $P^*(x) = \sum_{y \leq x} P(y)$ and the function P^* is computable in time t if there exists a $t(n)$-time-bounded Turing machine that on input $\langle x, 1^k \rangle$ writes the truncated binary expansion y of $P(x)$ such that $|P^*(x) - y| \leq 1/2^k$.

(a) Show that for every total recursive function f there exists a probability distribution P that is malign for the class of f-time bounded algorithms, and P^* is computable in polynomial time.

Define a probability ensemble as the set of probability distributions $\{P_n\}$ defined by $P_n(x) = P(x|l(x) = n)$. It is polynomial-time computable if each P_n^* is polynomial-time computable. Similarly for exponential time.

(b) Show that there exists an exponential-time computable probability ensemble that is malign for the class of polynomial-time algorithms.

(c) Show that there does not exist a polynomial-time computable probability ensemble that is malign for the class of polynomial-time algorithms.

Comments. Source: P.B. Miltersen, *SIAM J. Comput.*, 22:1(1993), 147–156. This contains many more results on malignness. K. Kobayashi studied the differences and relations between malign measures and universal

distributions in classes of distributions in [*IEICE Trans. Inform. Systems*, E76-D:6(1993), 634–640; *Theoret. Comput. Sci.*, 181(1997), 289–306]. A. Jakoby, R. Reischuk, and C. Schindelhauer studied malign distributions for average-case circuit complexity in [*Inform. Comput.*, 150(1999), 187–208].

7.6.2. [18] (a) Let $l(x) = n$. Show that probabilities $\mathbf{m}^t(x) = 2^{-K^t(x)}$ for the set of all x's with $K^t(x) = O(\log n)$ can be computed in time polynomial in $t(n)$.

(b) Use Item (a) to show that one can precompute $\mathbf{m}^t(x)$ with $x \in A$ for the set A of high-probability x's ($K^t(x) = O(\log n)$) in polynomial time for $t(n)$ polynomial.

Comments. Source: M. Li and P.M.B. Vitányi, *SIAM J. Comput.*, 20:5 (1991), 911–935.

7.6.3. [26] Recall Definition 4.3.5 on page 272. The *t-time-bounded universal a priori probability* is defined as $Q_U^t(x) = \sum_{U^t(p)=x} 2^{-l(p)}$, and $U^t(p) = x$ means that U computes x in at most $t(l(x))$ steps and halts. Let $<$ denote the standard lexicographic length-increasing ordering on the finite binary strings.

(a) Assume that we can determine, up to a fixed multiplicative constant, the number of programs p of length m with $U^t(p) = y \leq x$ in time polynomial in $t(n)$ ($n = l(x)$). Show that $Q^t(x) \leq \Theta(2^{-K^{r(t)}(x)})$ for some polynomial function r.

(b) If P=NP then $Q^t(x) \leq \Theta(2^{-K^{r(t)}(x)})$ for a polynomial function r.

Comments. The coding theorem, Theorem 4.3.3, without resource bounds is a major result in Kolmogorov complexity. Item (b) concerns a coding theorem about time-bounded universal distributions, conditional on the complexity-theoretic assumption P = NP. Even though this assumption is widely disbelieved, decades of research have not produced a stronger result than Item (b), or shown that a time-bounded coding theorem does not hold. Hint for Item (a): by Theorem 7.6.1, if $Q^{*t}(x)$ is computable in time $q(t(n))$ for some function q, then there is a constant c depending on Q and t such that $Q^t(x) \leq c\mathbf{m}^{t_q}(x)$, for some function $t_q(n) = O(nq(t(n)) \log nq(t(n)))$. To prove that $Q^{*t}(x)$ is computable in time polynomial in $t(n)$, we determine for each $m = 1, 2, \ldots, t(n)$ the number i_m of programs of length m with $U^t(p) \leq x$. Since no program of length exceeding $t(n)$ can be scanned to the end in the available running time $t(n)$, we have $Q^{*t}(x) = \sum_{m=1}^{t(n)} i_m 2^{-m}$. Under the assumption of Item (a) this computation of $Q^{*t}(x)$ takes time $O(q(t(n)))$ for some polynomial q. Hint for Item (b): There are fewer than $2^{t(n)+1}$ programs p of length $m \leq t(n)$. Hence, we can enumerate all of them in space $O(t(n))$.

If $t(n)$ is polynomial, then the entire computation of $Q^{*t}(x)$ can be performed in polynomial space. Naively, the complexity-theoretic assumption P = #P implies the assumption in the theorem. But using a result of L. Stockmeyer [*SIAM J. Comput.*, 14(1985), 849–861] if P = NP and $A \in$ P, then for every polynomial $p(n)$ we can determine in polynomial time a value a with $(1-1/p(n))a \leq d(A \bigcap \{0,1\}^n) \leq (1+1/p(n))a$. This shows that if P = NP we can determine $Q^{*t}(x)$ up to precision $\pm 1/p(n)$, which suffices. Source: L. Antunes and P.M.B. Vitányi, Manuscript, CWI, 2000, with H.M. Buhrman suggesting the Stockmeyer paper.

7.6.4. • [O46] Use the terminology of Exercise 7.6.3. The relation $Q^t(x) \geq \mathbf{m}^t(x)$ is obvious. Show whether $Q^t(x) = O(\mathbf{m}^t(x))$ with or without a complexity-theoretic assumption. If necessary relax the question as in Item (b) of Exercise 7.6.3.

Comments. Here we ask for the (non)existence of a time-bounded version of the Theorem 4.3.3.

7.6.5. [36] A probability mass function P is P-*computable* if it satisfies Definition 7.6.1 with the time $t(n)$ a polynomial in n. A probability mass function P is P-*samplable* if there is a probabilistic Turing machine T that on input k computes a string x such that $|\Pr(T(k) = x) - P(x)| \leq 1/2^k$ and T runs in time polynomial in $l(x) + k$. Clearly, every P-computable probability mass function is P-samplable.

(a) Show that if one-way functions exist then there is a P-samplable probability mass function that is not multiplicatively dominated by a P-computable one.

(b) Let P be a P-samplable probability mass function. For every polynomial p, there is a P-samplable probability mass function P such that $\mathbf{m}^p(x) \leq P(x)/p(x)$.

(c) Let P be a P-samplable probability mass function. Show that under a reasonable derandomization assumption, for some polynomial p, $\mathbf{m}^p(x) \geq P(x)/p(x)$.

(d) Show that if there exists a P-computable probability mass function P such that $\mathbf{m}^t(x) = O(P(x)$, for some t, then pseudorandom generators do not exist.

Comments. Existence of one-way functions also figures in Exercise 7.1.12 on page 548. Source for Item (a): S. Ben-David, B. Chor, O. Goldreich, and M. Luby, *20th ACM Symp. Theory Comput.*, 1989, 204–216; Item (b): L. Antunes, L. Fortnow, D. van Melkebeek, and N.V. Vinodchandran *Theoret. Comput. Sci.*, 354(2006), 391–404; Item (c): L. Antunes and L. Fortnow, *Electr. Coll. Comput. Complexity*, 144(2005); Item (d): R. Schuler [*16th Symp. Theoret. Aspects Comput. Sci.*, 1999, pp. 434–443; *12th IEEE Conf. Comput. Complexity* 1997, pp. 69–73].

7.6.6. [34] Let T be a constructible time function. Show that for every time-constructible function t we have $T(x) = 2^{O(K^t(x)-K(x)+\log n)}$ iff T is polynomial time on $\mathbf{m}^t$-average in L.A. Levin's sense, that is, there exists a k such that $\sum_x T^{1/k}(x)\mathbf{m}^t(x)/l(x) < \infty$.

Comments. Source: L. Antunes, L. Fortnow, D. van Melkebeek, and N.V. Vinodchandran, *Ibid.*

7.7 Logical Depth

One may consider a book on number theory difficult, or 'deep.' The book will list a number of difficult theorems of number theory. However, it has very low Kolmogorov complexity, since all theorems are derivable from the initial few definitions. Our estimate of the difficulty, or 'depth,' of the book is based on the fact that it takes a long time to reproduce the book from part of the information in it. The existence of a deep book is itself evidence of some long evolution preceding it. Currently, the sequence of primes is being broadcast to outer space, since it is deemed deep enough to prove to aliens that it arose as a result of a long evolution.

From the point of view of an investigator, a sequence is deep if it yields its secrets only slowly: one will be able to discover all significant regularities in it only if one analyzes it long enough.

Example 7.7.1 A suggestive example is provided by DNA sequences. Such a sequence is quite regular and has some 90% redundancy, possibly due to evolutionary history. A DNA sequence over an alphabet of four letters $\{A, C, G, T\}$ looks like nothing but a super-long (3×10^9 characters for humans) computer program. A particular three-letter combination literally signifies 'begin' of the encoding of a protein. Following the 'begin' command, every next block of three consecutive letters encodes one of the 20 amino acids. At the end, another three-letter combination signifies the 'end' of the program for this protein. Such a sequence is not Kolmogorov random, and it encodes the structure of a living being.

DNA is less random than, say, a typical configuration of gas in a container. On the other hand, DNA is more random than a crystal. Both gases and crystals are structurally trivial; the former is in complete chaos and the latter is in total order. Intuitively, DNA contains more useful information than both. A deep object, such as DNA, is something really simple but disguised by complicated manipulations of nature or computation by computer. To quote Charles Bennett: "A structure is deep, if it is superficially random but subtly redundant, in other words, if almost all its algorithmic probability [$\mathbf{m}$] is contributed by slow-running programs [...]. A priori the most probable explanation of 'organized information' such as the sequence of bases in a naturally occurring DNA molecule is that it is the product of an extremely long biological process." ◇

Logical depth is the necessary number of steps in the deductive or causal path connecting an object with its plausible origin. Formally, the notion of logical depth of an object is the amount of time required for an algorithm to derive the object from a shorter description. In fact, with some probability we can derive the object by simply flipping a fair coin. But for long objects this probability is small. If the object has a short description then we can flip that with higher probability. Bennett's proposal tries to express the tradeoff between the probability of flipping a short program and the shortest computation time from program to object. In order to solve some stability problems, Bennett's definition considers not only the object's shortest description but every description of the object and its computation time.

It turns out that it is quite subtle to give a formal definition of 'depth' that satisfies our intuitive notion of it. After some attempts at a definition, we will settle for Definition 7.7.1. As usual, we write x^* to denote the shortest self-delimiting program (of the reference universal prefix machine U) for x. If there is more than one of the same length, then x^* is the first such program in a fixed enumeration.

Attempt 1. The number of steps required to compute x from x^* is not a stable quantity, since there might be a program of just a few more bits using substantially less time to generate x. That this can happen is shown by the hierarchy theorems of Section 7.1.2. Therefore, a proper definition of depth probably should compromise between the program length and the computation time.

Attempt 2. Relax the strict requirement of minimum program to *almost minimum* programs. Define that a string x has depth d within error 2^{-b} if x can be computed in d steps by a program p of no more than b bits in excess of x^*. That is, $2^{-l(p)}/2^{-K(x)} \geq 2^{-b}$.

This definition is stable but is unsatisfactory because of the way it treats multiple programs of the same length. If 2^b distinct programs of length $m+b$ all compute x, then together they account for the same algorithmic probability, that is

$$\sum \{2^{-l(p)} : U(p) = x, l(p) = m + b\},$$

as one program of length m printing x does. That is, both situations are equally likely to produce x as output of the universal reference prefix machine in case the input is provided by fair coin tosses. But with the proposed definition, 2^b programs of length $m+b$ make the emerging of x no more probable than one program of length $m+b$.

We shall explicitly take the algorithmic probability into account. The universal prior probability of a string x is

$$Q_U(x) = \sum_{U(p)=x} 2^{-l(p)},$$

where U is the reference universal prefix machine. This is the probability that U would print x if its input were provided by random tosses of a fair coin. By Theorem 4.3.3 on page 273,

$$\log \frac{1}{Q_U(x)} = \log \frac{1}{\mathbf{m}(x)} = K(x), \tag{7.9}$$

with equality up to additive constants. The distinguishing characteristic of $\mathbf{m}$ is that it is a lower semicomputable discrete semimeasure such that for every other lower semicomputable discrete semimeasure P there is a constant c such that $c\mathbf{m}(x) \geq P(x)$ for every x. By Equation 7.9 this is satisfied if we simply set the reference universal semimeasure $\mathbf{m}$ in Theorem 4.3.1 precisely equal to $2^{-K(x)}$. Thus, weighing all possible causes of emergence of x appropriately, we are led to the following definition:

Definition 7.7.1 The *depth* of a string x at *significance level* $\epsilon = 2^{-b}$ is

$$\text{depth}_\epsilon(x) = \min \left\{ d : \frac{Q_U^d(x)}{Q_U(x)} \geq \epsilon \right\},$$

where $Q_U^d(x) = \sum_{U^d(p)=x} 2^{-l(p)}$ and $U^d(p) = x$ means that U computes x using program p within d steps and halts. A string x is (d, b)-*deep* if $d = \text{depth}_\epsilon(x)$ and $\epsilon = 2^{-b}$.

We can replace d by a time-constructible time function t in this definition. The relation of the new definition with Definition 7.7.1 is that $t(n) = d$ for $l(x) = n$.

Example 7.7.2 Logical depth is related to the randomness deficiency δ of Definition 4.3.9 on page 280, $\delta(x|Q_U^d(x)) = \log 1/Q_U^d(x) - K(x)$, in the sense that $-c \leq \delta(x|Q_U^d(x)) - \log Q_U(x)/Q_U^d(x) \leq 0$, where $c \geq 0$ is a constant satisfying $-c \leq \log \mathbf{m}(x)/Q_U(x) \leq 0$. (By definiton, $\mathbf{m}(x)$ is at most $Q_U(x)$, and c exists by Equation 7.9.) ◇

If x is (d, b)-deep, then x receives a $1/2^{b \pm \Delta}$ fraction of its algorithmic probability (for some small Δ) from programs running in d steps. Below we formalize this statement and make Δ precise. A binary string x is b-compressible if $K(x) \leq l(x) - b$. Otherwise, x is b-incompressible.

Theorem 7.7.1 *A string x is (d, b)-deep (b up to precision $K(d) + O(1)$) if and only if d is the least number of steps needed by a b-incompressible program to print x.*

Proof. The precise form of the statement of the theorem is

$$\frac{1}{2^{b+K(d)+O(1)}} \leq \frac{Q_U^d(x)}{Q_U(x)} \leq \frac{1}{2^{b-O(1)}}, \tag{7.10}$$

where Q_U^d is the d time-bounded analogue of Q_U, in the sense that the right inequality implies the 'if' part and the left inequality implies the 'only if' part.

(If) We prove the right inequality of Equation 7.10. Suppose that d is the least time required to compute x by a b-incompressible program. In other words, each program computing x in fewer than d steps is compressible by b bits. For each such program p, there is a program p' such that $U(p') = p$ and $l(p') \leq l(p) - b$. Let q be a program that simulates U on p' to obtain p and subsequently simulates p to obtain x. The length of q is about $l(p) - b + O(1)$. The relationship between p and q is one-to-one.

Recall that the notation $U^d(p) = x$ means that U computes x from p within d steps and halts. Let $\alpha = Q_U(x) - \sum_{U(q)=x} 2^{-l(q)}$, where the sum is taken over all q's as defined above. By Equation 7.9, we have $\alpha \geq 0$. Then,

$$\begin{aligned}
\frac{Q_U^d(x)}{Q_U(x)} &= \frac{\sum_{U^d(p)=x} 2^{-l(p)}}{\alpha + \sum_{U(q)=x} 2^{-l(q)}} \\
&\leq \frac{\sum_{U^d(p)=x} 2^{-l(p)}}{\sum_{U(q)=x} 2^{-l(q)}} \\
&\leq \frac{\sum_{U^d(p)=x} 2^{-l(p)}}{\sum_{U(q)=x} 2^{-(l(p)-b+O(1))}} \leq \frac{1}{2^{b-O(1)}}.
\end{aligned}$$

(Only If) Assume, by way of contradiction, that the left inequality of Equation 7.10 is false. Then given x^* and d^* (shortest self-delimiting programs for x and d), we can enumerate the set A of all self-delimiting programs computing x in time at most d by simulating all self-delimiting programs of length at most $l(x) + 2\log l(x) + 1$ for d steps. This set A can be computed by a program q of length

$$l(q) = K(x) + K(d) + O(1). \tag{7.11}$$

By definition of A we have $\sum_{p \in A} 2^{-l(p)} = Q_U^d(x)$. By the contradictory assumption, $2^{b+K(d)+O(1)} < Q_U(x)/Q_U^d(x)$. By Theorem 4.3.3 we have $Q_U(x) = 2^{-K(x)+O(1)}$. Together, it follows that

$$\sum_{p \in A} 2^{-l(p)} < 2^{-K(x)-b-K(d)-O(1)}. \tag{7.12}$$

Claim 7.7.1 Let B be a prefix-free set with $\sum_{x\in B} 2^{-l(x)} < 2^{-m}$, and B is enumerated by program s. Then every string in B can be effectively compressed by at least $m - l(s) - O(1)$ bits.

Proof. Increasing the probability $2^{-l(x)}$ of every x in B by a multiplicative factor 2^m, we still have $\sum_{x\in B} 2^{-l(x)}2^m < 1$. Therefore, the elements of B can be coded by the Shannon–Fano code as in Lemma 4.3.3 on page 274. The code word for each x in B has length at most $l(x)-m+2$. In order to make this coding effective, we use s to enumerate all and only strings in B. This takes an additional $l(s) + O(1)$ bits in the code for each $x \in B$. Hence, each string in B is effectively compressed by $m - l(s) - O(1)$ bits. □

We can choose $B = A$, $s = q$, and $m = K(x) + b + K(d) + O(1)$, in Claim 7.7.1, the last equality by Equation 7.12. Therefore, every program in A can be compressed by $K(x)+b+K(d)+O(1)-l(q)-O(1)$ bits. By Equation 7.11 and proper choice of the $O(1)$ constants, we conclude that every program in A can be compressed by more than b bits, contradicting the fact that x is (d, b)-deep. □

Definition 7.7.1 is not equivalent to the seemingly close Attempt 2. The reason is that in trying to work out the equivalence Theorem 4.3.3 must be used, and it brings in at least exponential time. The two are close when the depth we are interested in is much larger than exponential.

Example 7.7.3 If x is (d, b)-deep at some significance level 2^{-b}, then the universal machine must take at least d steps to compute x from x^* (the longest computation). Otherwise, x would be $(d', 0)$-deep with $d' < d$. Note that if a string is (d, b)-deep, then then it is also $(d, b + 1)$-deep. A program printing x of length $l(x) - b$ is an explanation of how the phenomenon x could occur with probability at least $2^{l(x)-b}$. Thus, with larger b, the explanation provided by the program involved is less compelling and less likely. If the object is very deep even with a larger b, then this means that even a less compelling explanation takes a long time. ◇

Definition 7.7.2 A string x is *d-shallow* if it is not $(d+1, b)$-deep for any b. Every string x must take at least $n = l(x)$ steps to be printed. If x is $n \pm O(1)$-shallow (at every significance level), then we will simply call x *shallow*.

Example 7.7.4 A random string x of length n is always shallow, because it can be printed by its shortest program x^* of length $n \pm O(1)$ in n steps. Thus, a random string is shallow at every significance level.

String 1^n is shallow, since a constant-size incompressible program prints it in time n given input 0^n. The strings $(01)^n$ and 01^n are likewise shallow. ◇

Example 7.7.5 We demonstrate the distinction between depth and information. Consider two sequences: the halting probability Ω defined in Section 3.6.2, and the halting sequence χ defined in the proof of Theorem 7.1.3. Although both encode the halting information of Turing machines, we show that χ is deep and Ω shallow.

One the one hand, since Ω encodes the halting problem with maximal density, it is recursively indistinguishable from a random string and practically useless, since $K(\Omega_{1:n}) \geq n - O(1)$ by Section 3.6.2. Thus, Ω is shallow.

On the other hand, since χ is a characteristic sequence of a recursively enumerable set constructed in the proof of Theorem 7.1.3, we have $C(\chi_{1:n}|n) \leq \log n$ by Barzdins's lemma (Theorem 2.7.2). If $d(n)$ is the minimum time (number of steps) required to compute $\chi_{1:n}|n$ from a $(\log n)$-sized incompressible program, then $\chi_{1:n}$ is $(d(n), b)$-deep for every b.

As a matter of fact, $\chi_{1:n}$ is very deep, since $d(n)$, the time required to compute an initial segment $\chi_{1:n}$ from a program of length $\log n$, increases faster than any computable function. Namely, Theorem 7.1.3 implies that $\chi_{1:n}$ can be computed from a program logarithmic in n if unlimited time is allowed, but can be computed only from a program that is linear in n if a computable bound is imposed on the decoding time. $\diamond$

Example 7.7.6 Depth is stable. That is, deep strings cannot be quickly computed from shallow ones (Exercise 7.7.3). In the genetic sense, organisms evolve relatively slowly. This may be called the *slow growth law*.

There is a mathematical version of such a law. Consider a string x and two significance parameters $b_2 > b_1$ (corresponding to significance levels $2^{-b_2} < 2^{-b_1}$). Let x be (d_1, b_1)-deep. A random program p generated by tossing a fair coin has probability less than $2^{-(b_2-b_1)+O(1)}$ of transforming x into an excessively deep output y which is (d_2, b_2)-deep and such that $d_2 > d_1 + \text{time}(p(x)) + O(1)$, where $\text{time}(p(x))$ is the run time of the transforming program p. In other words, the set of self-delimiting programs p such that $p(x) = y$ and y is (d_2, b_2)-deep has uniform measure less than $2^{-(b_2-b_1)+O(1)}$. We defer a proof to Exercise 7.7.5. $\diamond$

Example 7.7.7 It is not known whether all functions computable by algorithms that use space at most polynomial in the input length (PSPACE) can also be computed by deterministic algorithms that use time polynomial in the input length (P). Consider only computations in PSPACE. If P = PSPACE, then every string derivable from a given input has depth at most polynomial. If a string were the result of an exponentially long computation, then there would be a shorter computation of polynomial length to obtain it from the input. That is, the maximal depth of a string

computable in PSPACE would be polynomial. If, on the other hand, P $\neq$ PSPACE, then there would be strings computable in PSPACE whose depth is larger than polynomial.

Consider, for example, the development of DNA. It is a fair assumption that no computations in the evolution of DNA will exceed PSPACE. Then the maximal logical depth achievable is possibly exponential, such as 2^n, for a seed of length n. But if P = PSPACE, then there is always a shorter computation of length at most polynomial in n, and the evolved DNA is at most polynomially deep.

Similar statements can be made for micro states in evolving thermodynamic systems, for example, if the container, temperature variation, and pressure variation for a sample of gas molecules are fixed. ◇

Apparently, depth is different from Kt, since it may be the case that a random string is *greater* in Kt than a nonrandom one of equal length, but shallower in logical depth.

Exercises

7.7.1. [22] Strengthen the first inequality of Equation 7.10 to

$$2^{-b-\min\{K(b),K(d)\}-O(1)} \leq \frac{Q_U^d(x)}{Q_U(x)}.$$

Comments. Hint: given x^* and b^*, we can enumerate all programs in order of increasing running time and stop when the accumulated algorithmic probability measure exceeds $2^{-K(x)+b}$. Source: Lemma 3 in [C.H. Bennett, pp. 227–257 in: *The Universal Turing Machine: a Half-Century Survey*, R. Herken, ed., Oxford University Press, 1988]. This paper is also the source of all other exercises in this section, except Exercise 7.7.4.

7.7.2. [28] Deep strings are not easy to identify, but can be constructed by diagonalization. Prove that the following program finds a string which is (d, b)-deep for every significance parameter $b \geq n - K(d) - O(\log n)$: "Find all x of length n such that $Q_U^d(x) > 2^{-n}$; print the first string that is not in this set."

7.7.3. [30] Prove the following 'stability property.' Deep strings cannot be quickly computed from shallow ones. More precisely, there is a polynomial p and a constant c, both depending only on the universal machine U, such that if q is a program to compute x in d' steps and if q is less than (d, b)-deep, then x is less than $(d + p(d'), b + c)$-deep.

7.7.4. [25] Depth is machine-independent: If U and U' are two different reference universal machines, prove that there exists a polynomial p and a constant c, both depending only on U and U', such that $(p(d), b + c)$-depth on either machine is a sufficient condition for (d, b)-depth on the other.

7.7.5. [30] Prove the *slow growth law* of Example 7.7.6 on page 594.

7.8 History and References

The earliest discussion of resource-bounded Kolmogorov complexity is the quotation of A.N. Kolmogorov at the outset of this chapter. The earliest result in resource-bounded Kolmogorov complexity the authors know of is Theorem 7.1.3, due to J.M. Barzdins [*Soviet Math. Dokl.*, 9(1968), 1251–1254]. In the early 1970s, R.P. Daley wrote his Ph.D. thesis on Kolmogorov complexity with D.W. Loveland at Carnegie-Mellon University, publishing parts as [*J. Assoc. Comp. Mach.*, 20:4(1973), 687–695; *Inform. Contr.*, 23(1973), 301–312]. This work concerns resource bounds for uniform complexity $C(x;n)$ as defined by Loveland (Exercise 2.3.2 on page 130). These papers, and [L.A. Levin, *Problems Inform. Transmission*, 9(1973), 265–266; R.P. Daley, *Theoret. Comput. Sci.*, 4(1977), 301–309; L.M. Adleman, "Time, space, and randomness," LCS Report TM-131, 1979, MIT], are significant early documents of resource-bounded Kolmogorov complexity. The papers [J. Hartmanis, *Proc. 24th IEEE Found. Comput. Sci.*, 1983, pp. 439–445; Ker-I Ko, *Theoret. Comput. Sci.*, 48(1986), 9–33; M. Sipser, *Proc. 15th ACM Symp. Theory Comput.*, 1983, pp. 330–335] started the research wave in the eighties on resource-bounded Kolmogorov complexity. Related work on polynomial resource-bounded notions of randomness is by Y. Wang [*Randomness and Complexity*, Ph.D. thesis, Heidelberg, 1996; *Proc. 11th IEEE Conf. Structure in Complexity Theory*, 1996, pp. 180–189].

The *CD* notation in Definition 7.1.4 as well as Theorem 7.1.2 were introduced by M. Sipser in [*Ibid.*]. Theorems 7.1.4 and 7.1.5 and Definition 7.1.8 are from [Ker-I Ko, *Ibid.*]. The theory of upper semicomputable and recursive majorants of complexity is found in [A.K. Zvonkin, L.A. Levin, *Russ. Math. Surveys*, 25:6(1970), 83–124]; see also Example 4.1.1. and Exercise 7.1.5 on page 546.

The notation $C[f(n), t(n), s(n)]$ in Section 7.1.2 is from [J. Hartmanis, *Proc. 24th IEEE Found. Comput. Sci.*, 1983, pp. 439–445]. This paper had major influence on the research applying resource-bounded Kolmogorov complexity to computational complexity theory. Earlier, Daley [*J. Assoc. Comp. Mach.*, 20:4(1973), 687–695] defined similar notions for infinite sequences with uniform complexity: $\mathcal{C}[f|t] = \{x : (\forall^\infty n) C^t(x_{1:n}; n) \leq f(n)\}$, Exercises 7.1.7, and 7.1.8. The hierarchy theorems in Section 7.1.2 were first developed by J. Hartmanis [*Ibid.*]. L. Longpré's [Ph.D. thesis, Cornell University, 1986] contains Theorem 7.1.8, as well as the details of hierarchy theorems for complexity, space, and time.

Section 7.2 on language compression is based on [M. Sipser, *Proc. 15th ACM Symp. Theory Comput.*, 1983, pp. 330–335; H.M. Buhrman and

L. Fortnow, *Proc. 14th Symp. Theoret. Aspects Comput. Sci.*, Lect. Notes Comput. Sci., Springer-Verlag, Berlin, 1997, pp. 105-116; H.M. Buhrman, L. Fortnow, and S. Laplante, *SIAM J. Comput.* 31:3(2002), 887–905; A. Goldberg and M. Sipser, *SIAM J. Comput.*, 20(1991), 524–536]. The elegant Theorem 7.2.1, which is new to the third edition, is due to H.M. Buhrman and L. Fortnow, *Ibid.* Theorems 7.2.2 and 7.2.4 are from [M. Sipser, *Ibid.*]. Theorems 7.2.5 and 7.2.6 are from [A. Goldberg and M. Sipser, *Ibid.*]. The original proofs of the coding lemma, Lemma 7.2.2, and Theorem 7.2.6 use probabilistic arguments. Example 7.2.1 is one of the early applications of time-bounded Kolmogorov complexity. Originally, Sipser proved BPP $\subseteq \Sigma_4^p \bigcap \Pi_4^p$ using time-bounded Kolmogorov complexity. The current version, BPP $\subseteq \Sigma_2^p \bigcap \Pi_2^p$, is due to P. Gács. Theorem 7.2.7 is independently proved in [E. Allender, [Ph.D. thesis, Georgia Inst. Tech., 1985; A. Goldberg and M. Sipser, *Ibid.*]. Section 7.3.1 is based on [J. Hartmanis, *Proc. 24th IEEE Found. Comput. Sci.*, 1983, pp. 439–445], which also contains Lemma 7.3.1 and Theorems 7.3.1 and 7.3.2. Theorem 7.3.3 is due to R.V. Book, P. Orponen, D. Russo, and O. Watanabe [*SIAM J. Comput.,* 17(1988), 504–516]. A systematic study of oracle building tools is done by S. Fenner, L. Fortnow, S.A. Kurtz, and L. Li [*Inform. Computation*, 182:2(2003), 95–136]. See also [J. Feigenbaum, L. Fortnow, S. Laplante, and A. Naik, *Comput. Complexity* 7(1998), 174–191]. Theorem 7.3.4 is from [E. Allender and R. Rubinstein, *SIAM J. Comput.,* 17(1988), 1193–1202], while the equivalence of Items (i) and (iv) is independently due to J.L Balcázar and R.V. Book [*Acta Informatica*, 23(1986), 679–688] and J. Hartmanis and L. Hemachandra [*Inform. Process. Lett.,* 28(1988), 291–295]. Section 7.3.3 intends to capture a glimpse of the recent developments in connection with derandomization. Theorem 7.3.5 and Exercise 7.3.13 are from E. Allender, H. Buhrman, M. Koucký, D. van Melkebeek, and D. Ronneburger, *Proc. 43rd IEEE Symp. Found. Comp. Sci.*, 2002, pp. 669–678. The reader can find more connections between variations of resource-bounded Kolmogorov complexity and derandomization in: E. Allender, *Proc. 21st Conf. Found. Software Tech. Theor. Comp. Sci.*, 2001, pp. 1–15; E. Allender, M. Koucký, D. Ronneburger, and S. Roy, *Proc. 18th IEEE Conf. Comput. Complexity*, 2003, pp. 209–220; D. Ronneburger, Ph.D. thesis, Rutgers, 2004; M. Koucký, Ph.D. thesis, Rutgers, 2003; E. Allender, H. Buhrman, and M. Koucký, *Annals Pure Applied Logic*, 138(2006), 2–19. Kolmogorov complexity has played more useful roles in this field: R. Impagliazzo, R. Shaltiel, and A. Wigderson [*Proc. 32th ACM Symp. Theory Comput.* 2000, pp. 1–10] used Kolmogorov complexity to replace circuit complexity to prevent losing 'hardness' in a pseudorandom generator; A.E. Andreev, A.E.F. Clementi, J.D.P. Rolim, and L. Trevisan [*SIAM J. Comput.* 28:6(1999), 2103–2116] used Kolmogorov complexity in their work of simulating BPP in polynomial time using a weak random source; in a different direction, Y. Aumann,

Y.Z. Ding, and M.O. Rabin [*IEEE Trans. Inform. Theory*, 48:6(2002), 1668–1680] used Kolmogorov complexity to prove cryptographic security in their bounded storage model.

Application of resource-bounded Kolmogorov complexity in computational complexity is widespread. We did not cover [J.H. Lutz, *J. Comput. Syst. Sci.*, 44(1992), 220–258] generalizing a uniform (Lebesgue) measure in order to define *resource-bounded measure* μ over ESPACE $= \bigcup_{c\in\mathcal{N}} \mathrm{DSPACE}[2^{cn}]$. Note that here ESPACE is a countable set and under the usual uniform measure it has measure zero. Lutz uses a special-purpose notion of measure and shows that complete problems form a negligibly small subclass in ESPACE. For every real number $a > 1$, if

$$X = \{L : C^{\infty,s}(\chi_{L^{=n}}) \geq 2^n - an, \text{ for all but finitely many } n\},$$

where the space bound $s = 2^{O(n)}$, then $\mu(X|\mathrm{ESPACE}) = 1$, that is, almost every language computable in $2^{O(n)}$ space has high $2^{O(n)}$ space-bounded Kolmogorov complexity, almost everywhere. Every problem in a complexity class is reducible to every complete problem in that class. Intuitively, such complete problems must have highly organized structures and hence have *low* Kolmogorov complexity. This is the case for ESPACE. It is also shown that for every hard (under many-to-one polynomial reduction) language L for ESPACE there exists $\epsilon > 0$ such that $C^{-,s}(\chi_{A^{=n}}) < 2^n - 2^{n^\epsilon}$ infinitely often, with the space bound $s = 2^{2n}$. In [*Theoret. Comput. Sci.*, 81(1991), 127–135] J.H. Lutz also showed that if P is properly contained in BPP, then $\mu(\mathrm{E}|\mathrm{ESPACE}) = 0$, where $\mathrm{E} = \bigcup_{c\in\mathcal{N}} \mathrm{DTIME}[2^{cn}]$. See also [R.V. Book and J.H. Lutz, *SIAM J. Comput.*, 22:2(1993), 395–402]; the survey by D.W. Juedes and J.H. Lutz in [*Kolmogorov Complexity and Its Relation to Computational Complexity Theory*, O. Watanabe, ed., Springer-Verlag, Berlin, 1992, pp. 43–65]; and the survey by J.H. Lutz in [*Proc. 8th IEEE Conf. Struct. Compl. Theory*, 1993, pp. 158–175]. H. Buhrman and L. Longpré [*SIAM J. Comput.* 31:3(2002), 876–886] treat a resource-bounded version of compressibility of infinite sequences that is shown to be equivalent to the resource-bounded martingales in [J.H. Lutz, *J. Comput. Syst. Sci.*, 44(1992), 220–258]. In [K.W. Regan and J. Wang, *SIGACT News*, 25(1994), 106–113; A. Naik, K.W. Regan, and D. Sivakumar, *Theoret. Comput. Sci.*, 148(1995), 325–349] it is shown how to construct oracles in a quasilinear-time model.

Section 7.4, on instance complexity, is based on [P. Orponen, K. Ko, U. Schöning, and O. Watanabe, *J. Assoc. Comp. Mach.*, 41(1994), 96–121]. Several open questions in the first edition of this book were solved in [M. Kummer, *SIAM J. Comput.*, 25:6(1996), 1123–1143; H.M. Buhrman and P. Orponen, *J. Comput. Syst. Sci.*, 53:2(1996), 261–266; L. Fortnow and M. Kummer, *Theoret. Comput. Sci. A*, 161(1996), 123–140]. See Exercises 7.4.5, 7.4.8, 7.4.6, and 7.4.7.

Kt complexity of Section 7.5 is from [L.A. Levin, *Problems Inform. Transmission*, 9:3(1973), 265–266]. Levin's universal optimal search algorithm in Section 7.5.1 is Theorem 2 (without proof) of that paper. (Theorem 1 of the paper, also without proof, is the independent discovery of NP-complete problems. Earlier, S.A. Cook proved similar results in [*Proc. 3rd ACM Symp. Theory Comput.*, 1971, pp. 151–158]. Apparently, Levin did not want to publish the NP-completeness part of the paper. Kolmogorov urged him to publish and recommended the paper to *Problems Inform. Transmission*. Levin agreed on the condition that he could include the universal search part.) Our treatment partly follows [Y. Gurevich, *EATCS Bull.*, 35(1988), 71–82; R.J. Solomonoff, "Optimum sequential search," Memorandum, 1984]. In the simulation of the search procedure, it is standard to use so-called Kolmogorov–Uspensky machines, which allow a universal machine to simulate each Kolmogorov–Uspensky machine in linear time. In contrast, Turing machines have a logarithmic factor of slowdown because the universal Turing machine has only a fixed number of tapes. We ignored this problem. See [L.A. Levin, *Inform. Contr.*, 61(1984), 17–37] for further results on *Kt* complexity. Although the constant in the universal search procedure is forbiddingly large (exponential in the size of the shortest optimal program being looked for), there are some ideas to deal with this. R.J. Solomonoff [L.N. Kanal, J.F. Lemmer, eds., *Uncertainty in Artificial Intelligence*, Elsevier, 1986, pp. 473–491] proposes to learn first small chunks (subgoals) using the universal distribution, which could be done relatively fast. Then use the small chunks as building blocks to learn a more composite goal from a new copy of the universal distribution with the building blocks as the basic elements. In this way, one progresses fast to composite goals. Some computer experiments with universal optimal search are reported in [J. Schmidhuber, *Proc. 12th Int. Conf. Machine Learning*, Morgan Kaufmann, 1995, 488–496] for learning simple threshold units with high generalization capacity; and [J. Schmidhuber, J. Zhao, and M. Wiering, *Machine Learning*, 28:1(1997), 105–130] for solving partially observable mazes. M. Hutter [*Int. J. Found. Comput. Sci.*, 13:3(2002), 431-443] reduced the huge proportionality constant in the universal search algorithm to 5 at the cost of an additional tremendous additive constant, Exercise 7.5.3 on page 582. Related work on a notion of 'potential' is based on L.M. Adleman [*Ibid.*] and is not treated in this edition. In the first edition of this book we reformulated the ideas involved in terms of *Kt* complexity. L. Hemachandra and G. Wechsung [*Theoret. Comput. Sci.* 83(1991), 313–322] continued this approach. Our treatment of *Kt* complexity, universal optimal search, and potential has greatly profited from discussions with P. Gács and R.J. Solomonoff.

There is a great quantity of other work on resource-bounded complexity, including G. Peterson, *Proc. 21st IEEE Found. Comput. Sci.*, 1980, pp.

86–95; E. Allender, *J. Comput. System Sci.,* 39(1989), 101–124; J.H. Lutz, *J. Comput. System Sci.,* 41(1990), 307–320; M. Hermo and E. Mayordomo, *Math. Syst. Theory*, 27(1994), 247–356; J.L. Balcázar and U. Schöning, *Theoret. Comput. Sci.*, 99(1992), 279–290.

Section 7.6, on computable versions of the universal distribution, is partially based on [M. Li and P.M.B. Vitányi, *SIAM J. Comput.*, 20:5(1991), 911–935]. The original nonresource-bounded versions of the universal distribution are treated in Chapter 4. The computable version of the universal distribution in Section 7.6 can be used as the dominating distribution in simple pac-learning (Section 5.3). Another potential application exhibited in Section 4.4 used a distribution $\mathbf{q}(x)$ that is essentially a time-bounded version of $\mathbf{m}(x)$. See also the comments in the 'History and References' Section 4.7.

Logical depth as in Section 7.7 was first described by G.J. Chaitin in [*IBM J. Res. Develop.*, 21(1977), 350–359] and studied at greater length by C.H. Bennett in [*Emerging Syntheses in Science*, D. Pines, ed., Addison-Wesley, 1987, pp. 215–233; *The Universal Turing Machine: a Half-Century Survey*, R. Herken, ed., Oxford Univ. Press, 1988, pp. 227–258]. The material contained in Section 7.7 is primarily based on the last article of C.H. Bennett. There one can find Definition 7.7.1 and a variant (Lemma 3 in the paper) of Theorem 7.7.1. The proof given in the article is rather sketchy and informal. Bennett also defines the depth for infinite strings, which we have not discussed. Our treatment has greatly profited from discussions with P. Gács. For material on logical depth and fault-tolerant evolution, see [P. Gács, [pp. 223–326 in *Randomness in Computation*, Vol. 5, S. Micali, ed., JAI Press, 1989]; an asynchronous version is in W.G. Wang, Ph.D. thesis, Boston University, 1990]. D.W. Juedes, J.I. Lathrop, and J.H. Lutz [*Theoret. Comput. Sci.* 132(1994), 37–70] study relations between logical depth of infinite sequences and recursively time-bounded reducibility.

The edited volume [*Kolmogorov Complexity and Computational Complexity*, O. Watanabe, ed., Springer-Verlag, Berlin, 1992] contains five survey papers related to topics in this chapter. E. Allender presented applications of time-bounded Kolmogorov complexity to questions in complexity theory; see also Exercise 7.5.2 on page 581. R.V. Book treated sets with small information content, relating Sections 7.3.1 and 7.3.2 and current research in computational complexity theory. L. Longpré studied the statistical properties and tests for resource-bounded Kolmogorov random strings, supplementing Section 7.1.1. D.W. Juedes and J.H. Lutz treated the Kolmogorov complexity of characteristic sequences of languages. The survey by V.A. Uspensky analyzed the definitional and quantitative relations between the various versions of (nonresource-bounded) Kolmogorov complexity.

8

Physics, Information, and Computation

Various issues in information theory and theoretical physics can be fruitfully analyzed by Kolmogorov complexity. This is the case for physical aspects of information processing and for application of complexity to physics issues. Physicists and others have used complexity arguments in a variety of settings such as information distance, thermodynamics, chaos, biology, and philosophy. We touch briefly upon several themes, but focus on five main issues.

First, we analyze the relation between Shannon's entropy and the expected prefix complexity and the expected plain Kolmogorov complexity. Shannon's entropy measures the uncertainty in a probabilistic ensemble of messages, while Kolmogorov complexity measures the algorithmic information in an individual message. In Section 2.8 it was observed that the P-expected value of $C(\cdot)$ is asymptotic to the entropy $H(P)$ provided P satisfies some conditions. Using the prefix complexity and the theory of universal distributions we establish the precise relationships for every computable P. Similar relations exist between the other primary notions in Shannon's theory and algorithmic information theory.

Second, we look at energy dissipation in computing. Continued advance in the miniaturization and mobilization of computing devices will eventually require (near) dissipationless computing. Apparently, there are no physical laws that require reversible computations to dissipate energy. It is for this reason that notions of reversible computers, both physical and logical, are rapidly gaining prominence. We treat reversible computing first because it is used in some of the later topics in this chapter. We give both a variety of physical implementations that are theoretically possible, and a logical model—the reversible Turing machine. We then define the Kolmogorov complexity variant associated with this machine.

M. Li and P.M.B. Vitányi, *An Introduction to Kolmogorov Complexity and Its Applications*,
DOI: 10.1007/978-0-387-49820-1_8, © Springer Science + Business Media, LLC 2008

Third, while Kolmogorov complexity is an absolute measure of information in an object, it is desirable to have a similar absolute notion for the information distance between two objects. Among other things, absolute information distance is related to theories of pattern recognition in which one wants to express the notion of similarity. We define such an absolute notion of *information distance* and some of its variants based on considerations about either reversible (dissipationless) computations or irreversible computations. It turns out that these definitions define the same notion. A related distance measures the amount of nonreversibility of a computation. Next, we formulate certain weak requirements that any reasonable cognitive distance ought to satisfy. It then turns out that our earlier notion is the optimal cognitive distance. Subsequently, we introduce the theory of normalized information distance, measuring the relative similarity between two objects, which has a plethora of applications in practice in measuring similarity of general objects, be they genomes, English texts, music pieces, programs, patterns, time series, or even the semantics of words.

Fourth, we look at applications of Kolmogorov complexity in statistical thermodynamics. There, one explains the classical theory of thermodynamics by statistical and information-theoretic analysis of an underlying deterministic model. It turns out that a complexity analysis using the powerful methods developed in the first few chapters gives a basis of an algorithmic theory for physical entropy. Some applications include a proof of an 'entropy nondecrease over time' property, and 'entropy stability' property, 'entropy increase' for certain systems, and an analysis of Maxwell's demon.

Fifth, we formulate a notion of Kolmogorov complexity of pure quantum states, based on quantum Turing machines, and investigate its properties. We end with some miscellaneous topics.

8.1 Information Theory

The paradigms in the computation, manipulation, and transmission of information shift increasingly from being random-variable oriented to individual-object oriented. In theoretical terms, this means a shift from classical information theory to Kolmogorov complexity theory. Complex structural data such as sound files and video images are not well modeled by methods that assume that they consist of a typical sequence of uncorrelated independent outcomes of a simple Bernoulli source or even a stationary ergodic source. For example, storing or transmitting the first 1,000,000,000,000,000 bits of $\pi = 3.1415\ldots$ can be done the hard way using information theory, or the easy way using a small program incorporating an approximation algorithm that generates the successive bits π. Listen to Shannon and Kolmogorov:

"The fundamental problem of communication is that of reproducing at one point either exactly or approximately a message selected at another point. Frequently the messages have *meaning*; that is they refer to or are correlated according to some system with certain physical or conceptual entities. These semantic aspects of communication are irrelevant to the engineering problem. The significant aspect is that the actual message is one *selected from a set* of possible messages. The system must be designed to operate for each possible selection, not just the one which will actually be chosen since this is unknown at the time of design." [Shannon]

"The probabilistic approach is natural in the theory of information transmission over communication channels carrying 'bulk' information consisting of a large number of unrelated or weakly related messages obeying definite probabilistic laws. ... But what real meaning is there, for example, in asking how much information is contained in 'War and Peace'? Is it reasonable to include this novel in the set of 'possible novels,' or even to postulate some probability distribution for this set? Or, on the other hand, must we assume that the individual scenes in this book form a random sequence with 'stochastic relations' that damp out quite rapidly over a distance of several pages?"

"Our definition of the quantity of information has the advantage that it refers to individual objects and not to objects treated as members of a set of objects with a probability distribution given on it. The probabilistic definition can be convincingly applied to the information contained, for example, in a stream of congratulatory telegrams. But it would not be clear how to apply it, for example, to an estimate of the quantity of information contained in a novel or in the translation of a novel into another language relative to the original. I think that the new definition is capable of introducing in similar applications of the theory at least clarity of principle." [Kolmogorov]

Shannon ignores the object itself but considers only the characteristics of the random source of which the object is one of the possible outcomes, while Kolmogorov considers only the object itself to determine the number of bits in the ultimate compressed version irrespective of the manner in which the object arose. For almost every Shannon theory notion there turns out to be a Kolmogorov complexity theory notion that is equivalent in the sense that the expectation of the latter is closely related to the former.

8.1.1 Algorithmic Complexity and Shannon's Entropy

There is a close relation between information theory and coding, Section 1.11. In particular, the quantity

$$H(P) = \sum_x P(X = x) \log \frac{1}{P(X = x)}$$

is the entropy of a random variable X with probability $P(X = x)$ of outcome x (Definition 1.11.1 on page 67). For convenience we abbreviate the notation $P(X = x)$ to $P(x)$, conforming to our customary usage.

Let E be a prefix-code achieving the minimum average code-word length. It assigns code word $E(x)$ to source word x. The noiseless coding theorem, Theorem 1.11.2 on page 77, asserts that the average code-word length $L(P) = \sum_x P(x)l(E(x))$ for the prefix-code E satisfies

$$H(P) \leq L(P) \leq H(P) + 1.$$

By definition $K(x) = l(x^*)$, where x^* is a shortest self-delimiting program for x with respect to the reference prefix machine. If there is more than one such shortest program then x^* is the first one in the standard enumeration. Consider the mapping E defined by $E(x) = x^*$. This is a prefix-code, and by its definition a very parsimonious one. Suppose the source words x are distributed as a random variable X with probability $P(x)$. The expected code-word length of source words with respect to probability mass function P is $\sum_x P(x)K(x)$.

What we would like to know is the following: While $K(x)$ is fixed for each x and *independent* of the probability mass function P, is K still so universal that its P-expected code-word length $\sum_x P(x)K(x)$ achieves the minimal average code-word length $H(P) = \sum_x P(x) \log 1/P(x)$?

This requirement on individually shortest programs for source words contrasts with the Shannon–Fano code of Example 1.11.2 on page 68, which does on average achieve the $H(P)$ bound by taking the code-word length equal to the negative logarithm of the specific source-word probability.

Surprisingly, under some mild restrictions on the probability mass functions P, the expectation of $K(x)$ comes close to $H(P)$. We can view the $K(x)$'s as the code-word length set of a 'universal' Shannon–Fano code based on the universal probability, Theorem 1.11.2 on page 77. The expectation of $K(x)$ differs from $H(P)$ by a constant depending on P. Namely, $|\sum_x P(x)K(x) - H(P)| \leq c_P$, where the constant c_P depends on the length of the program to compute the probability mass function P. In Exercise 8.1.4 this dependence is removed.

If the source-word alphabet is infinite, then it is possible that $H(P)$ is infinite. For example, with x ranging over all of $\mathcal{N}$ and $P(x) = c/(x \log^2 x)$ with c a constant chosen such that $\sum_x P(x) = 1$, we have $H(P) > \sum_x 1/(x \log x)$, which diverges. We consider P such that $H(P) < \infty$.

Theorem 8.1.1 *Let $H(P) = \sum_x P(x) \log 1/P(x)$ be the entropy of a computable probability distribution P, and $H(P) < \infty$. Then,*

$$0 \leq \sum_x P(x)K(x) - H(P) \leq c_P,$$

with $c_P = K(P) + O(1)$ a constant that depends only on P.

Proof. Since $K(x)$ is the code-word length of a prefix-code for x, the first inequality of the noiseless coding theorem, Theorem 1.11.2 on page 77, states that

$$H(P) \leq \sum_x P(x)K(x).$$

By Theorem 4.3.1 on page 267, if $\mathbf{m}$ is the universal lower semicomputable distribution, then $P(x) \leq 2^{K(P)}\mathbf{m}(x)$. By Theorem 4.3.3 on page 273, we have $\log 1/\mathbf{m}(x) = K(x) + O(1)$. Together this shows that $\log 1/P(x) \geq K(x) - K(P) + O(1)$. It follows that

$$\sum_x P(x)K(x) \leq H(P) + K(P) + O(1).$$

Define the constant c_p by

$$c_p := K(P) + O(1),$$

and the theorem is proven. Note that the constant implied in the $O(1)$ term depends on the lengths of the programs occurring in the proof of Theorem 4.3.3. These depend only on the reference universal prefix machine. □

In other words, probabilistic entropy of a random variable X taking outcomes in $\mathcal{N}$ is equal, within an additive constant, to the expected value of complexity of these outcomes.

Above we required P to be computable. Actually, we require only that P be a lower semicomputable function, which is a weaker requirement than computability. However, together with the condition that P be a probability mass function satisfying $\sum P(x) = 1$, this means that P is computable anyway by Example 4.3.2 on page 266.

What does this mean for the plain C complexity? Since $K(x) \leq C(x) + K(C(x)) + O(1)$ by Example 3.1.4 on page 203, and $C(x) \leq K(x) + O(1)$, also $\log 1/P(x)$ and $C(x)$ are close to each other with high probability. Substituting $K(x)$ in Theorem 8.1.1 as above, we obtain ($H(P) < \infty$)

$$-c_P \leq H(P) - \sum_x P(x)C(x) \leq \sum_x P(x)K(C(x)), \tag{8.1}$$

all inequalities up to a constant additive term. The right-hand side is bounded only if $\sum_x P(x)K(C(x))$ converges. This is certainly the case if P runs through a sequence of distributions $P_1, P_2, \ldots$ such that $H(P_i) \to \infty$ for $i \to \infty$ and $\lim_{i\to\infty} \sum_x P_i(x)K(C(x))/H(P_i) = 0$. Then, by Equation 8.1 we find that $H(P_i)$ is asymptotically equal to the expected complexity $\sum_x P_i(x)C(x)$, as was also established in a weaker form in Example 2.8.1 on page 188. Theorem 8.1.2 below shows that this scenario is possible.

A prominent example of a distribution with infinite entropy is the universal distribution: $\sum_x \mathbf{m}(x) \log 1/\mathbf{m}(x) = H(\mathbf{m}) = \infty$. That is, the $\mathbf{m}$-expectation of $K(x)$ is infinite. See Exercise 4.3.4 on page 287. However, $\sum_x \mathbf{m}(x) < 1$ unlike a probability mass function P that satisfies $\sum_x P(x) = 1$, and $\mathbf{m}$ is lower semicomputable but not computable.

Theorem 8.1.2 *Let P be a computable probability mass function and $H(P) = \infty$. There exists a sequence of functions P_i, each with a finite support, and $P_i \to P$ for $i \to \infty$, such that* $\lim_{i\to\infty} \sum_x P_i(x)C(x)/\, H(P_i) = 1$.

Proof. Given i, determine n such that $\sum_{x\le n} P(x) \log 1/P(x) \ge i$. Define P_i by $P_i(x) = P(x)$ for $x \le n$, and 0 otherwise. Since $c_{P_i} \le K(P_i) + O(1) \le 2 \log i + K(P) + O(1)$, and $H(P_i) \ge i$, we have

$$\lim_{i\to\infty} \frac{c_{P_i}}{H(P_i)} = 0. \tag{8.2}$$

Using Equations 8.1 and 8.2 we only need to establish

$$\lim_{i\to\infty} \frac{\sum_x P_i(x)K(C(x))}{H(P_i)} = 0.$$

From Equation 8.2 and Theorem 8.1.1 we have

$$\lim_{i\to\infty} \frac{\sum_x P_i(x)K(x)}{H(P_i)} = 1.$$

Since $K(C(x)) \le 2 \log(K(x) + O(1)) + O(1)$, we have

$$\lim_{i\to\infty} \frac{\sum_x P_i(x)K(C(x))}{\sum_x P_i(x)K(x)} \le \lim_{i\to\infty} \frac{\sum_x 2P_i(x) \log K(x)}{\sum_x P_i(x)K(x)}.$$

We need to prove that the right-hand side equals 0, which is not immediate. We need to exclude the possibility that P concentrates probability such that equality to 0 does not hold. We reason as follows. For every $\epsilon > 0$ we can find an N with $(\log K(x))/K(x) \le \epsilon$ for all $x \ge N$. Divide the sums in the numerator and the denominator of the right-hand side of the last displayed equation into a finite sum for all $x \le N$ and an infinite sum for $x > N$. Deleting the finite sum in the denominator makes the right-hand side larger. Subsequently, divide both sums in the numerator by the infinite sum left in the denominator. Then, the first term goes to 0 because the numerator is finite and the denominator goes to infinity. Finally, the second term is at most ϵ, which can be chosen arbitrarily close to 0. □

Since the expected complexities C and K are close to the probabilistic entropy, the intended interpretation of $C(x)$, and especially $K(x)$, as a measure of the information content of an individual object x is supported by a tight quantitative relationship to Shannon's probabilistic notion.

Example 8.1.1 We examine the expected complexity over a set $A \subseteq \mathcal{N}$ of source words under a probability mass function P, where the conditional probability is defined by $P_A(x) = P(x|x \in A)$ and $P_A(x) = 0$ for $x \notin A$. Let P_A be computable and $H(P_A) < \infty$. Then, Theorem 8.1.1 applied to P_A yields

$$0 \leq \sum_x P_A(x)K(x) - H(P_A) \leq K(P, A) + O(1). \tag{8.3}$$

If A is finite, then the expected complexity and the entropy of the conditional probability P_A are always finite. A natural case is to consider the set A of all x's of length n, that is $A = \{0,1\}^n$. Then $K(P, A) = K(P, n) + O(1)$. Note that $K(n)$ can be very small for regular n and is bounded from above by $\log n + 2 \log\log n$ for all n. ◇

Example 8.1.2 Example 8.1.1 essentially looks at the influence of the support of a computable probability mass function P (set of arguments with positive probabilities) with $H(P) < \infty$, on the difference between the expected prefix complexity and the entropy. It is instructive to consider the following example. Let x be a string of high complexity $K(x)$, and let P concentrate all probability on x, that is, $P(x) = 1$. Then, $H(P) = 0$ and the expected prefix complexity is $K(x)$. Since we can choose x such that $K(x)$ is arbitrarily high, this seems to contradict the equality dictated by Theorem 8.1.1. The contradiction is only apparent, since $K(P) = K(x) + O(1)$: We can compute P from x and vice versa by a fixed program. ◇

A theorem of A.E. Romashchenko shows that *every* linear entropy (in)-equality holds for the corresponding Kolmogorov complexities (both C- and K-type) to within a logarithmic additive error. Morover, every linear (in)equality that is valid for Kolmogorov complexity is also valid for Shannon entropy—provided we require the Kolmogorov complexity (in)equalities to hold up to additive logarithmic precision only. In view of the importance of the result we state it precisely: Let $X_1, X_2, \ldots, X_m$ be random variables with a joint distribution. For a set $A \subseteq \{1, 2, \ldots, m\}$, X_A denotes the tuple $\langle X_i : i \in A\rangle$. For example, $X_{\{2,3,5\}} = \langle X_2, X_3, X_5\rangle$. In the same way, if $x_1, x_2, \ldots, x_m$ denotes a sequence of strings, then $x_A = \langle x_i : i \in A\rangle$. For example, $x_{\{2,3,5\}} = \langle\langle x_2, x_3\rangle, x_5\rangle$. The proof of the following theorem is deferred to Exercise 8.1.5 on page 626.

Theorem 8.1.3 *If an inequality of the form*

$$\sum_{A,B} \alpha_{A,B} H(X_A|X_B) \leq 0$$

holds for all random variables $X_1, X_2, \ldots, X_m$, then for some function $f(n) = O(\log n)$ the inequality

$$\sum_{A,B} \alpha_{A,B} K(x_A | x_B) \leq f(n)$$

holds for all strings $x_1, x_2, \ldots, x_m$ with $n = l(x_1 x_2 \ldots x_m)$, where the summation is over all subsets A, B of $\{1, 2, \ldots, m\}$, and the $\alpha_{A,B}$ are real numbers (which can be 0). Conversely, if for some function $f(n) = o(n)$ the second inequality holds for all $x_1, x_2, \ldots, x_m$, then the first inequality holds for all $X_1, X_2, \ldots, X_m$.

We have to be very careful when extending prefix complexity inequalities such as Theorem 3.9.1 on page 247 that hold up to $O(1)$ additive terms. See, for example, the discussion of the conditional version of the symmetry of information, Equation 3.22 on page 253. But up to logarithmic precision we do not need to be that careful and can ignore the difference between x^* and x, as witnessed by the fact that both C-complexity and K-complexity satisfy the symmetry of information property within an additive logarithmic term.

8.1.2 Mutual Information

The probabilistic mutual information $I(X;Y)$ is defined as in Equation 1.15 on page 71. Recall the definition of algorithmic mutual information given in Equation 3.20 on page 252,

$$I(x;y) = K(x) + K(y) - K(x,y),$$

which is truly symmetric, since $I(x;y) = I(y;x)$.

See Exercise 3.9.5 on page 254 about its relation to the version $I(x : y) = K(y) - K(y|x)$ of Equation 3.15 on page 248.

It is an important fact that the expectation of the algorithmic mutual information is close to the probabilistic mutual information:

Lemma 8.1.1 *Let X, Y be random variables with outcomes in sets of finite objects $\mathcal{X}$, $\mathcal{Y}$, respectively, with a computable joint probability mass function p, and such that $H(X,Y)$ is finite. Then,*

$$\begin{aligned} I(X;Y) - K(p) &\leq \sum_x \sum_y p(x,y) I(x;y) + O(1) \qquad (8.4) \\ &\leq I(X;Y) + 2K(p) + O(1), \end{aligned}$$

where $K(p)$ is the length of the shortest prefix-free program that computes $p(x,y)$ for every $(x,y) \in \mathcal{X} \times \mathcal{Y}$.

We need to require only that p be a lower semicomputable function, Definition 1.7.4 on page 35, which is a weaker requirement than computability. But together with the condition that $\sum_{x,y} p(x,y) = 1$, this means that p is computable anyway, by Example 4.3.2 on page 266.

Proof.

$$\sum_x \sum_y p(x,y) I(x;y) = \sum_x \sum_y p(x,y)[K(x) + K(y) - K(x,y)].$$

Define $p_1(x) = \sum_y p(x,y)$ and $p_2(y) = \sum_x p(x,y)$ to obtain

$$\sum_x \sum_y p(x,y) I(x;y)$$
$$= \sum_x p_1(x) K(x) + \sum_y p_2(y) K(y) - \sum_{x,y} p(x,y) K(x,y).$$

Given the program that computes p, we approximate $p_1(x)$ monotonically nondecreasing by $q_1(x, y_0) = \sum_{y \le y_0} p(x,y)$ with increasing y_0, and a similar approximation holds for p_2. That is, the probability mass functions p_1 and p_2 are lower semicomputable, and by the comment following the statement of the lemma, they are computable. For every computable probability mass function q, we have

$$H(q) \le \sum_x q(x) K(x) + O(1) \le H(q) + K(q) + O(1),$$

by Theorem 8.1.1 on page 604. Hence,

$$H(p_i) \le \sum_x p_i(x) K(x) + O(1) \le H(p_i) + K(p_i) + O(1),$$

for $i = 1, 2$, and

$$H(p) \le \sum_{x,y} p(x,y) K(x,y) + O(1) \le H(p) + K(p) + O(1).$$

On the other hand, the probabilistic mutual information of Equation 1.15 can be expressed in the entropies of the marginal distributions p_1, p_2 and the joint distribution p as

$$I(X;Y) = H(p_1) + H(p_2) - H(p).$$

By construction of the q_i's above, we have $K(p_1), K(p_2) \le K(p) + O(1)$. Since the complexities are positive, substitution yields the lemma. □

Can we get rid of the $K(p)$ error term? The answer is affirmative; by putting p in the conditional we even get rid of the computability requirement.

Lemma 8.1.2 *Given a joint probability mass function $p(x,y)$ (not necessarily computable) we have*

$$I(X;Y) = \sum_x \sum_y p(x,y) I(x;y|p) + O(1),$$

where the auxiliary p in the conditional means that we can directly access the values $p(x, y)$ on the auxiliary conditional information tape of the reference universal prefix machine.

Proof. If we show that $\sum_x p(x)K(x|p) = H(p) + O(1)$, where the $O(1)$ term is independent of p, then the lemma follows from the definition of conditional algorithmic mutual information, Example 3.9.3 on page 252. To prove the desired equality, equip the reference universal prefix machine with an $O(1)$ length program to compute a Shannon–Fano code from the auxiliary table of probabilities. Then, given an input r, it can determine whether r is the Shannon–Fano code word for some x. Such a code word has length $\log 1/p(x) + O(1)$. If this is the case, then the machine outputs x; otherwise it halts without output. Therefore, $K(x|p) \leq \log 1/p(x) + O(1)$. This shows the upper bound on the expected prefix complexity. The lower bound follows from the noiseless coding theorem. □

8.1.3 Mutual Information Nonincrease

Algorithmic mutual information is expressed in terms of the individual outcomes rather than probabilistic characteristics of random variables, as in the probabilistic version. Yet the data-processing inequality, the average notion of Equation 1.19 on page 72, also holds between individual objects, by a far more subtle argument, and not precisely but with a small tolerance. The first to observe this fact was Leonid A. Levin, who proved 'information nongrowth,' and 'information conservation inequalities' for both finite and infinite sequences under both deterministic and randomized data processing. We prove a strong version of the information nonincrease law under deterministic processing (later we need the attached corollary).

Theorem 8.1.4 *Given x and z, let q be a program computing z from x^*. Then*

$$I(z; y) \leq I(x; y) + K(q) + O(1). \tag{8.5}$$

Proof. By the triangle inequality, Lemma 3.9.1 on page 249,

$$\begin{aligned} K(y|x^*) &\leq K(y|z^*) + K(z|x^*) + O(1) \\ &= K(y|z^*) + K(q) + O(1). \end{aligned}$$

Thus,

$$\begin{aligned} I(x; y) &= K(y) - K(y|x^*) \\ &\geq K(y) - K(y|z^*) - K(q) + O(1) = I(z; y) - K(q) + O(1). \end{aligned}$$

□

This also implies the slightly weaker but intuitively more appealing statement that the mutual information between strings x and y cannot be increased by processing x and y separately by deterministic computations.

Corollary 8.1.1 Let f, g be recursive functions. Then

$$I(f(x); g(y)) \leq I(x; y) + K(f) + K(g) + O(1). \tag{8.6}$$

Proof. Let q_f, q_g be shortest programs computing functions f, g, respectively. Ignoring additive constant terms,

$$\begin{aligned} I(f(x); g(y)) &\leq I(x; g(y)) + K(q_f) \\ &\leq I(x; y) + K(q_f) + K(q_g) \\ &\leq I(x; y) + K(f) + K(g), \end{aligned}$$

where the first two inequalities are true by Equation 8.5. □

Even randomized computation cannot increase information except with negligible probability. Recall the universal probability $\mathbf{m}(x) = 2^{-K(x)}$. By Theorem 4.3.1 on page 267 and Theorem 4.3.3 on page 273, this function is maximal within a multiplicative constant among lower semicomputable semimeasures. This property also holds within certain restrictions (satisfied below) when we have an extra parameter, like y^*, in the condition, Theorem 4.3.2 on page 270. In particular, for every computable measure $\nu(x)$ we have $\nu(x) = O(\mathbf{m}(x))$, where the constant factor implied in the order-of-magnitude symbol O depends on ν.

Suppose that z is obtained from x by randomized computation. For every fixed x, the probability $p(z|x)$ of obtaining z from x is lower semicomputable. Therefore,

$$p(z|x) = O(\mathbf{m}(z|x)) = O(\mathbf{m}(z|x^*)) = O(2^{-K(z|x^*)}).$$

The information increase $I(z; y) - I(x; y)$ by random computation on x about y satisfies the theorem below.

Theorem 8.1.5 *For all x, y, z we have*

$$\mathbf{m}(z|x^*)2^{I(z;y)-I(x;y)} = O(\mathbf{m}(z|x^*, y, K(y|x^*))).$$

For example, the probability of an increase of mutual information by an amount d is $O(2^{-d})$. The theorem implies $\sum_z \mathbf{m}(z|x^*)2^{I(z;y)-I(x;y)} = O(1)$; the $\mathbf{m}(\cdot|x^*)$-expectation of the exponential of the increase is bounded by a constant.

Proof. We have

$$\begin{aligned} I(z;y) - I(x;y) &= K(y) - K(y|z^*) - (K(y) - K(y|x^*)) \\ &= K(y|x^*) - K(y|z^*). \end{aligned}$$

The negative logarithm of the left-hand side in the theorem is therefore

$$K(z|x^*) + K(y|z^*) - K(y|x^*).$$

Using Lemma 3.9.1 and the conditional additivity, Equation 3.22 on page 253, this is at least

$$K(y, z|x^*) - K(y|x^*) + O(1) = K(z|x^*, y, K(y|x^*)) + O(1).$$

□

Example 8.1.3 **(Extending Gödel's incompleteness)** An example of the use of algorithmic mutual information is given by L.A. Levin: see the 'History and References' section on page 714 for references. We follow Levin's explanation: D. Hilbert asked whether formal arithmetic can be consistently extended to a complete theory. The question was somewhat vague, since an obvious answer was "yes": just add to the axioms of Peano arithmetic (PA) a maximal consistent set, clearly existing albeit hard to find. K. Gödel formalized this question as existence among such extensions of recursively enumerable ones, and gave it a negative answer. Its mathematical essence is the lack of total recursive extensions of universal partial recursive predicates. This negative answer apparently was never accepted by Hilbert, and Gödel himself had reservations:

> "Namely, it turns out that in the systematic establishment of the axioms of mathematics, new axioms, which do not follow by formal logic from those previously established, again and again become evident. It is not at all excluded by the negative results mentioned earlier that nevertheless every clearly posed mathematical yes-or-no question is solvable in this way. For it is just this becoming evident of more and more new axioms on the basis of the meaning of the primitive notions that a machine cannot imitate." [Gödel]

As is well known, the absence of algorithmic solutions is no obstacle when the requirements do not make a solution unique. A notable example is generating strings of linear Kolmogorov complexity, for example, those that cannot be compressed to half their length. Algorithms fail, but a set of dice does a perfect job! Thus, while enumerable sets of axioms cannot complete PA, completion by other realistic means remained a possibility.

Of course, Gödel's remark envisioned much more sophisticated ways to choose axioms than coin flips. However, the impossibility of a task can be formulated generically using Kolmogorov's concept of algorithmic mutual information in two finite strings. It has been refined and extended

to infinite sequences by L.A. Levin, so that it satisfies conservation inequalities: cannot be increased by deterministic algorithms or in random processes or by any combinations of both. In fact, it seems reasonable to assume that no physically realizable process can increase information about a specific sequence.

In this framework one can ask whether the nonmechanical means envisioned by Gödel could really enable the task of consistent completion for PA. A negative answer follows from the existence of a specific sequence that has infinite mutual information with *all* total extensions of a universal partial recursive predicate. It plays the role of a password: no substantial information about it can be guessed, no matter what methods are allowed. ◇

8.1.4 Rate Distortion

Rate-distortion theory analyzes the transmission and storage of information at insufficient bit rates. The aim is to minimize the resulting information loss expressed in a distortion measure. The choice of distortion measure is a selection of which aspects of the data are relevant, in the setting at hand, and which aspects are irrelevant (noise). For example, lossy compression of a sound file results in a compressed file where, among others, the very high and very low inaudible frequencies have been suppressed. The distortion measure is chosen such that it penalizes the deletion of the inaudible frequencies only lightly because they are not relevant for the auditory experience. The classical probabilistic theory is reviewed in Section 1.11.6. Algorithmic rate-distortion theory is a generalization of the structure function approach of Section 5.5.

As before, let $\mathcal{X}$ be a set of *source* objects, called *words* or *messages*. Suppose we want to communicate source words $x \in \mathcal{X}$ using *destination words* $y \in \mathcal{Y}$ that can be encoded with at most r bits apiece. If the Kolmogorov complexity $K(x)$ of the source word $x \in \mathcal{X}$ is greater than r, or if x is not a finite object, then $K(y) \leq r < K(x)$ for every code word $y \in Y$. Therefore, x cannot be reproduced from any such y. Assume furthermore that we are given a *distortion* function $d : \mathcal{X} \times \mathcal{Y} \to \mathcal{R} \bigcup \{\infty\}$ that measures the fidelity of the coded version against the source data.

Definition 8.1.1 Given $\mathcal{X}, \mathcal{Y}, d(\cdot,\cdot)$ and a particular $x \in \mathcal{X}$, we define the *rate-distortion function* r_x as the minimum number of bits we need to transmit a code word y (so that y can be effectively reconstructed from the transmission) to obtain a distortion of at most δ:

$$r_x(\delta) = \min_y \{K(y) : d(x,y) \leq \delta\}.$$

The 'inverse' of the above function is is called the *distortion-rate function* and is defined by

$$d_x(r) = \min_y \{d(x,y) : K(y) \leq r\}.$$

These functions are analogues for individual data x of the probabilistic functions, expressing the least expected rate or distortion at which outcomes from a random source X can be transmitted, Section 1.11.6.

Definition 8.1.2 Let $\mathcal{X}$ be a source alphabet, $\mathcal{Y}$ a destination alphabet, and d a distortion measure. A *distortion ball* $B(y, \delta)$ centered on $y \in \mathcal{Y}$ with radius δ is defined by

$$B(y, \delta) = \{x : d(x, y) \leq \delta\} \text{ and } b(y, \delta) = d(B(y, \delta)).$$

Every (destination alphabet, distortion measure) pair gives rise to a set of distortion balls. Without loss of generality, every such distortion ball corresponds uniquely with a (destination word, distortion) pair (if not, take the first pair in a given order).

Definition 8.1.3 A *distortion family* $\mathcal{A}$ is a set of finite nonempty subsets of the set of source words $\mathcal{X}$. By $\mathcal{A}_n$ we denote the restriction of $\mathcal{A}$ to strings of length n.

Example 8.1.4 The class of distortion families obviously includes every family of distortion balls (or distortion spheres, which is sometimes more convenient), and hence all combinations of destination alphabets and distortion measures. Let $\mathcal{X}$, $\mathcal{Y}$, and distortion measure d be as before. The corresponding distortion family $\mathcal{A}$ is defined by

$$\mathcal{A} = \{B(y, \delta) : y \in \mathcal{Y},\ \delta \text{ in the range of } d\}.$$

Given a string x, we can look for a finite set $A \in \mathcal{A}$ that contains x and is both small and simple. For every $x \in \{0, 1\}^*$ we want to identify the set of pairs of integers (k, l) such that there is $A \in \mathcal{A}$ with $x \in A$, $K(A) \leq k$, and $\log d(A) \leq l$. The set P_x of all such pairs will be called the *profile of* x. Strings of the same complexity can have quite different profiles. All such pairs (k, l) satisfy the inequality $k + l \geq K(x)$ (since we can specify x by providing a k-bit description of A and l-bit ordinal number of x in A). ◇

Definition 8.1.4 The *canonical rate-distortion function* g_x of a string x is defined by

$$g_x(l) = \min_{A \in \mathcal{A}} \{K(A) : x \in A, \log d(A) \leq l\}.$$

Example 8.1.5 Thus, for every natural $l \leq n$, the function value $g_x(l)$ is the minimum k such that the pair (k, l) belongs to the profile of x. The rate-distortion function r_x differs from g_x just by a change of scale depending on the distortion family involved. If for $\mathcal{X}$, $\mathcal{Y}$, and d, we have $b(y, \delta) = b(y', \delta)$

for all $y, y' \in \mathcal{Y}$, then we drop parameter y and let $b(\delta)$ denote the cardinality of a distortion ball of radius δ. Then,

$$r_x(\delta) \approx g_x(\lceil \log b(\delta) \rceil). \tag{8.7}$$

◇

Example 8.1.6 Let $\mathcal{X} = \{0,1\}^n$. Consider Hamming distortion, with δn the number of bits flipped out of n. Then, by Exercise 8.1.10 on page 627,

$$r_x(\delta) = g_x(nH(\delta)) + O(\log n),$$

for $0 \leq \delta \leq \frac{1}{2}$, and $H(\delta) = \delta \log 1/\delta + (1-\delta) \log 1/(1-\delta)$ is the binary entropy function, Definition 1.11.1 on page 67. Describing the family of g_x's, we obtain a description of all possible rate-distortion functions r_x for elements of $x \in \mathcal{X}$, one possibility being depicted in Figure 8.1. ◇

Distortion Measures

The *prefix complexity of a finite family $\mathcal{A}$ of finite nonempty subsets* $A_1, \ldots, A_m$ of $\{0,1\}^*$ is defined as $K(\mathcal{A}) = K(A_1, \ldots, A_m)$, where the sequence $A_1, \ldots, A_m$ is in a fixed order, say lexicographic.

We consider arbitrary distortion measures only restricted by the following mild conditions on families $\mathcal{A}$:

Property 1. For every natural number n, the family $\mathcal{A}$ contains the set $\{0,1\}^n$ of all strings of length n as an element.

Property 2. All $x, y \in A \in \mathcal{A}$ satisfy $l(x) = l(y)$.

Property 3. Let $\mathcal{A}_n = \{A \in \mathcal{A} : A \subseteq \{0,1\}^n\}$. We assume that $K(\mathcal{A}_n) = O(\log n)$.

Property 4. For every natural n, let α_n denote the minimal number that satisfies the following. For every positive integer c every set $A \in \mathcal{A}_n$ can be covered by at most $\alpha_n d(A)/c$ sets $B \in \mathcal{A}_n$ with $d(B) \leq c$. Call α_n the *covering coefficient* related to $\mathcal{A}_n$. We desire that α_n be bounded by a polynomial of n. The smaller the covering coefficient is, the more accurate will be the description that we obtain of the shapes of the structure functions.

The following three example families $\mathcal{A}$ satisfy all four properties. Let n be a natural number.

Example 8.1.7 $\mathcal{L}$, *the list distortion family.* Let $\mathcal{L}_n$ be the family of all nonempty subsets of $\{0,1\}^n$. This is the family of distortion balls consisting of strings of length n, for list distortion, which we define as follows. Let $\mathcal{X} = \{0,1\}^n$

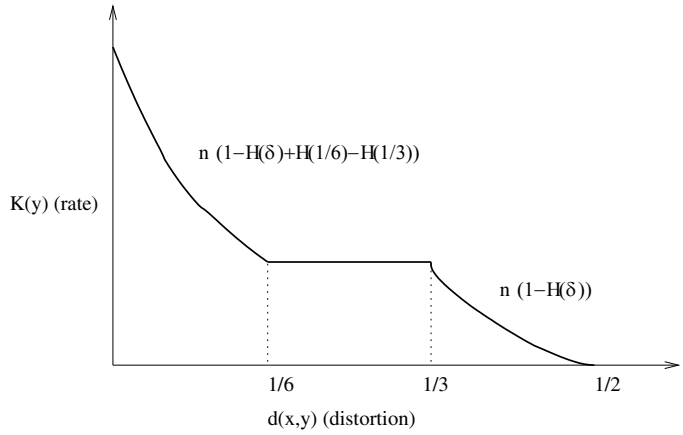


FIGURE 8.1. A rate-distortion function for Hamming distortion

and $\mathcal{Y}$ be the set of all subsets of $\{0,1\}^n$. An $x \in \{0,1\}^n$ is encoded by a subset or *list* $S \subseteq \{0,1\}^n$ with $x \in S$. Given S, we can retrieve x by its index of $\log d(S)$ bits in S, ignoring rounding up, whence the name 'list code.' The distortion measure is $d(x,S) = \log d(S)$ if $x \in S$, and ∞ otherwise. Thus, distortion balls come only in the form $B(S, \log d(S))$ with cardinality $b(S, \log d(S)) = d(S)$. Trivially, the covering coefficient as defined in property 4 on page 615, for the list distortion family $\mathcal{L}_n$ satisfies $\alpha_n \leq 2$. In Section 5.5.2 we described all possible profiles in terms of Kolmogorov's structure function $h_x(i)$ of Definition 5.5.6 on page 405. More precisely, the function $h_x(i)$ equals $d_x(i)$, the distortion-rate function for the distortion family $\mathcal{L}_n$. The distortion-rate function is the inverse of the rate-distortion function r_x, up to some minor issues. The rate-distortion function of x of length n for list distortion is

$$r_x(\delta) = \min_{S \subseteq \{0,1\}^n} \{K(S) : x \in S,\ \log d(S) \leq \delta\}.$$

The canonical rate-distortion function g_x can be converted to the particular rate-distortion function r_x for a family $\mathcal{L}_n$ according to Equation 8.7 on page 615. ◇

Example 8.1.8 *$\mathcal{H}$, the Hamming distortion family.* Let $\mathcal{H}_n$ be the family of all Hamming balls in $\{0,1\}^n$. Let $\mathcal{X} = \mathcal{Y} = \{0,1\}^n$. An $x \in \{0,1\}^n$ is encoded by a $y \in \{0,1\}^n$. The Hamming distance between two strings $x = x_1 \ldots x_n$ and $y = y_1 \ldots y_n$ is defined as

$$d(x,y) = \frac{1}{n} d(\{i : x_i \neq y_i\}). \tag{8.8}$$

A Hamming ball with center $y \in \{0,1\}^n$ and radius δ is the set $B(y,\delta) = \{x \in \{0,1\}^n : d(x,y) \le \delta\}$. The cardinality $b(y,\delta)$ depends only on n and δ but not on the center y; we denote it by $b_n(\delta)$. Every x is in either $B(00\ldots0, \frac{1}{2})$ or $B(11\ldots1, \frac{1}{2})$, so we need to consider only Hamming distortion $0 \le \delta \le \frac{1}{2}$. We will use the following approximation of $b_n(\delta)$. Suppose that $0 \le \delta \le \frac{1}{2}$ and δn is an integer, and let $H(\delta)$ be the binary entropy function as before. Then,

$$2^{nH(\delta)-\log n/2-O(1)} \le b_n(\delta) \le 2^{nH(\delta)}. \tag{8.9}$$

In Exercise 8.1.9 on page 626 it is shown that the covering coefficient as defined in property 4 on page 615, for the Hamming distortion family $\mathcal{H}_n$ satisfies $\alpha_n = n^{O(1)}$. The function

$$r_x(\delta) = \min\{K(y) : d(x,y) \le \delta\}$$

is the rate-distortion function of x for Hamming distortion. One such function is depicted in Figure 8.1. ◇

Example 8.1.9 $\mathcal{E}$, *the Euclidean distortion family.* Let $\mathcal{E}_n$ be the family of all intervals in $\{0,1\}^n$, where an interval is a subset of $\{0,1\}^n$ of the form $\{x : a \le x \le b\}$ and $\le$ denotes the lexicographic ordering on $\{0,1\}^n$. Let $\mathcal{X} = \mathcal{Y} = \{0,1\}^n$. An $x \in \{0,1\}^n$ is encoded by a $y \in \{0,1\}^n$. Interpret strings in $\{0,1\}^n$ as binary notations for rational numbers in the segment $[0,1]$. Consider the Euclidean distance $|x-y|$ between rational numbers x and y. The balls in this metric are intervals; the cardinality of a ball of radius δ is about $\delta 2^n$. Trivially, the covering coefficient as defined in property 4 on page 615, for the Euclidean distortion family $\mathcal{E}_n$ satisfies $\alpha_n \le 2$. The function

$$r_x(\delta) = \min\{K(y) : |x-y| \le \delta\}$$

is the rate-distortion function of x for Euclidean distortion. ◇

All the properties are straightforward for all three families, except property 4 in the case of the family of Hamming balls.

Every Shape Let $\mathcal{A}$ be a distortion family satisfying properties 1 through 4 on page 615. Property 4 applied to $A = \{0,1\}^n$ and $c = 1$, for every n, implies that the family $\mathcal{A}$ contains the singleton set $\{x\}$ for every $x \in \{0,1\}^*$. Thus,

$$g_x(0) = K(\{x\}) = K(x) + O(1).$$

Property 1 implies that for every n and string x of length n,

$$g_x(n) \le K(\{0,1\}^n) = K(n) + O(1) \le \log n + O(\log\log n).$$

Together this means that for every n and every string x of length n, the function $g_x(l)$ decreases from about $K(x)$ to about 0 as l increases from 0 to n.

Lemma 8.1.3 *Let $\mathcal{A}$ be a distortion family satisfying properties* 1 *through* 4 *on page* 615. *For every n and every string x of length n we have $g_x(n) = O(\log n)$, and $0 \leq g_x(m) - g_x(l) \leq l - m + O(\log n)$ for all $m < l \leq n$.*

Proof. The first equation and the left-hand inequality of the second equation are straightforward. To prove the right-hand inequality, translate it into the following property of the profile of x. If a pair (k, l) is in P_x and $m < l$, then also $(k + l - m + O(\log n), m) \in P_x$. Let A witness $(k, l) \in P_x$. Find a covering of A by at most $\alpha_n d(A)/2^m$ sets in $\mathcal{A}$, each of cardinality at most 2^m. Let B be a covering set containing x. It can be specified by A and the index i of B among the covering sets, given the list of all sets in $\mathcal{A}_n$ and m. We need also $O(\log k + \log\log i + \log\log m)$ extra bits to separate the description of A, the binary representations of i and m, and the description of $\mathcal{A}_n$ from one another. Without loss of generality we can assume that k is less than n. Thus all the extra information and separator bits are included in $O(\log n)$. Altogether, $K(B) \leq K(A) + l - m + O(\log n) \leq k + l - m + O(\log n)$. □

Example 8.1.10 Lemma 8.1.3 shows that

$$K(x) - i - O(\log n) \leq g_x(i) \leq n - i + O(\log n),$$

for every $0 \leq i \leq n$. The right-hand inequality is obtained by setting $l = n$ in the lemma, yielding

$$g_x(m) - g_x(n) = g_x(m) - O(\log n) \leq n - m + O(\log n).$$

The left-hand inequality is obtained by setting $m = 0$:

$$g_x(0) - g_x(l) = K(x) - g_x(l) + O(1) \leq l + O(\log n).$$

This can also be shown by a simple direct argument: x can be described by the minimal description of the set $A \in \mathcal{A}$ witnessing $g_x(i)$ and by the ordinal number of x in A. ◇

Let G_n stand for the class of all functions $g : \{0, 1, \ldots, n\} \to \mathcal{N}$ such that $g(n) = 0$ and $g(l-1) \in \{g(l), g(l)+1\}$ for all $1 \leq l \leq n$. In other words, a function g is in G_n iff it is nonincreasing and the function $g(i) + i$ is nondecreasing, and $g(n) = 0$. The following result should be compared with Theorem 5.5.2 dealing with $h_x = d_x$ in the particular case of the distortion family $\mathcal{L}_n$. There, the precision in Item (ii) is $O(\log n)$, rather than the $O(\sqrt{n \log n})$ we obtain for general distortion families.

Theorem 8.1.6 *Let $\mathcal{A}$ be a distortion family satisfying properties* 1 *through* 4 *on page* 615.

(i) *For every n and every string x of length n, the function $g_x(l)$ is equal to $g(l) + O(\log n)$ for some function $g \in G_n$.*

(ii) *Conversely, for every n and every function g in G_n, there is a string x of length n such that for all $l = 0, \ldots, n$, $g_x(l) = g(l) + O(\sqrt{n \log n})$.*

Proof. We defer the proof to Exercise 8.1.8 on page 626. □

Fitness of Destination Word

Section 5.5 deals with the particular distortion family $\mathcal{L}_n$. Every set containing a string x is considered to be a model for x. For every string x of length n, Theorem 5.5.1 and its examples show that x has minimal randomness deficiency in every witness set of $h_x(i)$ ($h_x = d_x$ in the current terminology), ignoring $O(\log n)$ terms. That is, up to the stated precision every such witness set is the best-fitting model that is possible at model complexity at most i. It is remarkable that the analogue also holds for more general distortion families $\mathcal{A}$.

Theorem 8.1.7 *Let $\mathcal{A}$ be a distortion family satisfying properties* 2 *and* 3 *on page* 615. *Let B be a set in $\mathcal{A}_n$ and let x be a string in B. Let A_x denote a set of minimal Kolmogorov complexity among the sets $A \in \mathcal{A}$ with $x \in A$ and $\lceil \log d(A) \rceil = \lceil \log d(B) \rceil$. Then,*

$$K(A_x) + \log d(A_x) - K(x) \leq \delta(x|B) + O(\log K(B) + \log n).$$

Proof. Note: $\delta(x|A_x) + O(\log n) = K(A_x) + \log d(A_x) - K(x)$ by the symmetry of information, Theorem 3.9.1, and property 3. We defer the proof to Exercise 8.1.15 on page 628. □

Example 8.1.11 For every $A \in \mathcal{A}$ we consider the uniform probability distribution over A. Assume that we are given a string x that was obtained by a random sampling in an unknown set $B \in \mathcal{A}$. Given x we want to recover B, or some $A \in \mathcal{A}$ that is 'a good hypothesis to be the source of x.' The quoted expression has a clear meaning in algorithmic statistics, Example 5.5.1 on page 402. According to Definition 5.5.4 on page 403, a set $A_x \ni x$ is a sufficient statistic for x iff $K(A_x) + \log d(A_x) = K(x) + O(1)$. By Lemma 5.5.1 on page 403, in that case x is a typical element of A_x; it has small randomness deficiency in A_x. By Theorem 8.1.7, if x is chosen at random in B then the probability of the event

$$K(A_x) + \log d(A_x) - K(x) > \beta$$

is less than

$$\epsilon = 2^{-\beta + O(\log K(B) + \log n)}.$$

By Item (ii) of Example 5.5.1 on page 402, the properties of randomness deficiency, the probability that the right-hand side of the inequality in Theorem 8.1.7 exceeds β is at most ϵ. Thus, with high probability the set A_x is a sufficient statistic for x. $\diamond$

Example 8.1.12 **(Denoising)** Assume the conditions of Example 8.1.11, and let $k = K(A_x)$ and $l = \lceil \log d(A_x) \rceil$. Since $\delta(x|A_x) \leq K(A_x) + \log d(A_x) - K(x) \leq \delta(x|B) + O(\log K(B) + \log n)$, we have

$$\delta(x|A_x) \leq \delta(x|B) + O(\log K(B) + \log n),$$

and A_x is an (almost) best possible model for x at either complexity k, or of log-cardinality l, and hence both. This gives a method to identify hypotheses via compression: if A_x is a set in $\mathcal{A}$ of minimal Kolmogorov complexity among sets A with $x \in A$ and $\lceil \log d(A) \rceil = l$, then the hypothesis '$x$ is chosen at random in A_x' is (almost) at least as plausible as the hypothesis 'x is chosen at random in B' for any other simply described $B \in \mathcal{A}$ with $\lceil \log d(B) \rceil = l$.

Let us look at an example of denoising by compression (in the ideal sense of Kolmogorov complexity). Fix a target string x_0 of length n and a nonnegative $\delta \leq \frac{1}{2}$. Let a string x be a noisy version of x_0 by changing at most $n\delta$ randomly chosen bits in x_0. That is, the string x is chosen uniformly at random in the Hamming ball $B = B(x_0, \delta)$. Let $\hat{x}$ be a string witnessing $r_x(\delta)$, that is, a string of minimal Kolmogorov complexity in the Hamming ball $B(x, \delta)$. We claim that $\hat{x}$ is a good candidate for a denoised version of x. Indeed, let $l = \lceil nH(\delta) \rceil$. Theorem 8.1.7 implies that

$$g_x(l) + l - K(x) \leq \delta(x|B) + O(\log n)$$

(the term $\log K(B)$ is absorbed by $O(\log n)$). Since the Hamming distortion family satisfies all properties 1 through 4 on page 615, the canonical structure functions g_x satisfy Theorem 8.1.6. For every x the rate-distortion function r_x of x differs from g_x just by changing the scale of the argument as in Equation 8.7 on page 615. More specifically, for every $0 \leq \delta \leq \frac{1}{2}$, we have

$$r_x(\delta) = g_x(\lceil nH(\delta) \rceil) + O(\log n),$$

and therefore

$$r_x(\delta) + l - K(x) \leq \delta(x|B) + O(\log n).$$

Since we assume that x is chosen uniformly at random in B, the randomness deficiency $\delta(x|B)$ is small with high probability, and the right-hand

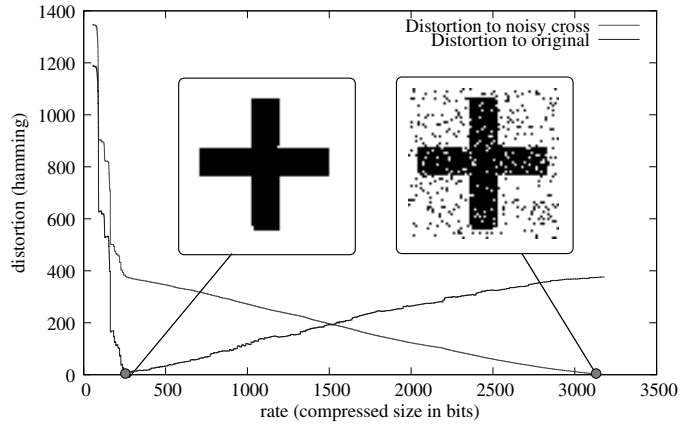


FIGURE 8.2. Denoising of the noisy cross

side of the last displayed inequality is small, ignoring values of order $O(\log n)$. Thus with high probability,

$$0 \approx K(\hat{x}) + l - K(x) \approx K(B(\hat{x}, \delta)) + \log b(\hat{x}, \delta) - K(x),$$

and the ball $B(\hat{x}, \delta)$ is a sufficient statistic of x. In other words, in the two-part description $(\hat{x}, \hat{x} \oplus x)$ of x, the second part (the bitwise XOR of x and $\hat{x}$) is noise. This provides a method of denoising via compression, at least in theory (since the Kolmogorov complexity is not computable). Practically, it turns out that using real compressors to approximate the Kolmogorov complexity, the method works surprisingly well.

As an example, we approximated the distortion-rate function of a noiseless cross of very low complexity, to which artificial noise was added to obtain a noisy cross. Figure 8.2 shows two graphs. The first graph, hitting the horizontal axis at about 3100 bits, denotes the Hamming distortion on the vertical axis of the best model for the noisy cross with respect to the original noisy cross at the rate given on the horizontal axis. The line hits zero distortion at model cost bit rate about 3100, when the original noisy cross is retrieved. The best model of the noisy cross at this rate, actually the original noisy cross, is attached to this point. The second graph, hitting the horizontal axis at about 250 bits, denotes on the vertical axis the Hamming distortion of the best model for the noisy cross with respect to the noiseless cross at the rate given on the horizontal axis. The line hits almost zero distortion (Hamming distance 3) at model cost bit rate about 250. The best model of the noisy cross at this rate is attached to this point. (The three wrong bits are

at the bottom left corner and upper right armpit.) This coincides with a sharp slowing of the rate of decrease of the first graph. Subsequently, the second graph rises again because the best model for the noisy cross starts to model more noise. Thus, the second graph shows us the denoising of the noisy cross, underfitting left of the point of contact with the horizontal axis, and overfitting right of that point. This point of best denoising can also be deduced from the first graph, where it is the point where the distortion-rate curve sharply levels off. Since this point has distortion of only 3 to the noiseless cross, the distortion-rate function separates structure and noise very well in this example. ◇

Example 8.1.13 Theorem 8.1.7 says that for fixed log-cardinality l the model that has minimal complexity has also minimal randomness deficiency among models of that log-cardinality. Since g_x satisfies Lemma 8.1.3, we have also that for every k the model of complexity at most k that minimizes the log-cardinality also minimizes randomness deficiency among models of that complexity. These models can be computed in the limit, in the first case by running all programs up to k bits and always keeping the one that outputs the smallest set in $\mathcal{A}$ containing x, and in the second case by running all programs up to $n = l(x)$ bits and always keeping the shortest one that outputs a set in $\mathcal{A}$ containing x having log-cardinality at most l. ◇

Characterization

Let X be a random variable with outcomes in $\mathcal{X}$ and $X_1, X_2, \ldots, X_n$ consist of n i.i.d. copies of X, denoted by X^n. The second part of Shannon's theorem, Theorem 1.11.3, states that there exists a random variable Z taking values in the destination alphabet $\mathcal{Y}$, such that we can code the outcomes in $\mathcal{X}^n$ (the source words) in length about $nI(X;Z)$ (of the destination words) with the average distortion between the source-word outcomes of X^n and their destination words, divided by n, being close to $\mathbf{E}d(X,Z)$ as n grows large. The algorithmic version below about individual data differs from Shannon's theorem as explained in Example 8.1.14.

Theorem 8.1.8 *Let $\mathcal{A}$ be a distortion family satisfying properties* 2 *and* 3 *on page* 615. *Define $\mathcal{A}(x) = \{A \in \mathcal{A} : x \in A\}$. For every x and every $B \in \mathcal{A}(x)$ there is an $A \in \mathcal{A}(x)$ with $\lceil \log d(A) \rceil = \lceil \log d(B) \rceil$ and $K(A) \leq I(x : B) + O(\log K(B) + \log n)$, where $I(x : B) = K(B) - K(B|x)$ stands for the information in x about B and $n = l(x)$.*

Proof. The proof of Shannon's theorem, Theorem 1.11.3, and the proof of the current theorem are very different. The latter proof uses techniques that may be of independent interest; see Exercises 8.1.13 and 8.1.14 on page 628. □

Example 8.1.14 Note that $\lceil \log d(A) \rceil = \lceil \log d(B) \rceil$ equals $\lceil \log b(\delta) \rceil$ in Equation 8.7 on page 615, where it is the log-cardinality of a distortion ball in the distortion family $\mathcal{A}_n$. In this way we can determine the value of $g_x(\lceil \log b(d) \rceil)$ and subsequently retrieve both the distortion δ concerned and the value of the rate-distortion function $r_x(\delta)$. The theorem states that a destination word minimizing the algorithmic mutual information with the given source word gives no advantage in rate (a pointswise less rate-distortion curve) over a minimal complexity destination word. This result on the rate-distortion function of individual data contrasts with Shannon's rate-distortion function for a random variable (whose outcomes are the individual data). In Shannon's case the minimum information of some random variable with the source random variable can be less than the minimum entropy of a function of the source variable. $\diamond$

Example 8.1.15 Theorem 8.1.8 states that for every family $\mathcal{A}$ of finite nonempty subsets of $\{0,1\}^*$ and for every string x, if there exists an $A \in \mathcal{A}$ of cardinality 2^l or less containing x that has small information about x, then there exists another set $B \in \mathcal{A}$ containing x that has also at most 2^l elements and has small prefix complexity. For example, in the case of Hamming distortion, if for a given string x there exists a string y at Hamming distance δ from x that has small information about x, then there exists another string z that is also within distance δ of x and has small prefix complexity itself (not only small information about x). $\diamond$

Algorithmic versus Probabilistic Rate Distortion

Probabilistic rate-distortion theory was treated in Section 1.11.6. Let $\mathcal{X}$ and $\mathcal{Y}$ be finite. We generalize the setting from i.i.d. random variables to more general random variables. Let $X_1, X_2, \ldots, X_n$ be a sequence of, possibly dependent, random variables with values in $\mathcal{X}^n$ such that $p(x_1 x_2 \ldots x_n) = P(X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n)$ is rational. With $X = X_1, X_2, \ldots, X_n$ and $x = x_1 x_2 \ldots x_n$, let $K(X)$ denote the prefix complexity of the set of pairs $(x, p(x))$ ordered lexicographic. Let $E : \mathcal{X}^n \to \mathcal{Y}^n$ be a prefix-free code. Define the rate-distortion function by

$$r^n(\delta) = \min_E \{\log d(E(\mathcal{X}^n)) : \mathbf{E} d(x, E(x)) \leq \delta\}, \tag{8.10}$$

the expectation $\mathbf{E}$ taken over the probability mass function p. Roughly speaking, we prove that $r^n(\delta)$ is close to the p-expected value of $r_x(\delta)$ for all δ. See also Exercise 8.1.16 on page 628.

Theorem 8.1.9 *Assume the discussion above. Let E_0 be a many-to-one coding function defined by $E_0(x) = y$ with $d(x,y) \leq \delta$ and $r_x(\delta) = K(y)$. Then,*

$$\mathbf{E} r_x(\delta) - \epsilon_1 \leq r^n(\delta) \leq \min \left\{ \mathbf{E} r_x(\delta) + \epsilon_2, \max_{x \in \mathcal{X}^n} r_x(\delta) \right\},$$

with $\epsilon_1 = O(K(\delta, r^n, X, n))$, $\epsilon_2 = H(L) - H(S)$ *with* $S(y) = \sum\{p(x) : E_0(x) = y\}$, $L(y)$ *is the uniform distribution over the* y*'s over* $\mathcal{Y}^n$*, and the expectation* $\mathbf{E}$ *is taken over* p*.*

Proof. *Left inequality.* Given δ, n, p, and the (discrete) graph of r^n, we can compute an optimal E as in Equation 8.10 such that $r^n(\delta) = \log d(E(\mathcal{X}^n))$. Retrieve $E(x)$ for every x by its index of $r^n(\delta)$ bits in the set $E(\mathcal{X}^n)$. Then,

$$K(E(x)) \leq r^n(\delta) + O(K(\delta, r^n, X, n)),$$

and by definition, $r_x(\delta) \leq K(E(x))$. Taking the expectation of $r_x(\delta)$ over p, we are done.

Right inequality. Define a code E_0 such that $K(E_0(x)) = r_x(\delta)$ for every $x \in \mathcal{X}^n$. Let $E_0(\mathcal{X}^n)$ be the range of E_0. Although $E_0(\mathcal{X}^n)$ cannot be computed, it is finite, and trivially

$$\log d(E_0(\mathcal{X}^n)) \leq \max_{x \in \mathcal{X}^n} K(E_0(x)).$$

By definition $r^n(\delta) \leq \log d(E_0(\mathcal{X}^n))$, which yields $r^n(\delta) \leq \max_{x \in \mathcal{X}^n} r_x(\delta)$.

The noiseless coding theorem, Theorem 1.11.2 on page 77, shows that

$$\sum_{x \in \mathcal{X}^n} p(x) r_x(\delta) = \sum_{y \in E_0(\mathcal{X}^n)} S(y) K(y) \geq H(S),$$

with S the distribution defined in the statement of the lemma. By definition, $r^n(\delta) \leq d(\mathcal{Y}^n)$, which yields $r^n(\delta) \leq H(L)$, with L as in the statement of the lemma. Together, we obtain $r^n(\delta) \leq \mathbf{E} r_x(\delta) + \epsilon_2$. □

Exercises

8.1.1. [25] Let P be a (possibly incomputable) probability mass function, and let $P_n(x) = P(x|l(x) = n)$ and $P_n(x) = 0$ for $l(x) \neq n$. Show that $H(P_n) - \sum_x P_n(x) C(x) \leq \log n + 2 \log \log n + O(1)$, where the $O(1)$ term is independent of P and n.

Comments. Hint: By $K(x) \leq C(x) + K(C(x)) + O(1)$ and the left side of Equation 8.3 on page 607 we have $H(P_n) - \sum_x P_n(x) C(x) \leq \sum_x P_n(x) K(C(x)) + O(1)$.

8.1.2. [19] Let P be a probability mass function and let $A \subseteq \mathcal{N}$ be recursively enumerable. Define $P_A(x) = P(x|x \in A)$ and $P_A(x) = 0$ for $x \notin A$, and let P_A be computable and $H(P_A)$ finite. Show that $\sum_x P_A(x) C(x|A, P_A) \leq H(P_A) + O(1)$, where the $O(1)$ term is independent of P and A.

Comments. Hint: let $x^1, x^2, \ldots$ be an enumeration of A by nonincreasing $P_A(x^i)$ probabilities. We can find this enumeration, since we can enumerate A until the first k elements constitute A_k and $\max\{P_A(x) : x \in A_k\} \geq 1 - \sum_{y \in A_k} P_A(y)$. The x that achieves the maximum is x^1, and so on. Clearly, $C(x^i|A, P_A) \leq \log i + O(1)$. This yields $\sum_x P_A(x) C(x|A, P_A) \leq \sum_i P_A(x^i) \log i + O(1) \leq H(P_A) + O(1)$ with the $O(1)$ term independent of P and A. (The last inequality follows from $P_A(x^i) \leq \sum_{1 \leq j \leq i} P_A(x^j)/i$ yielding $\log(iP_A(x^i)) \leq \log 1 = 0$. Therefore, $\log i \leq \log 1/P_A(x^i)$.) Source: Adaption of an idea of B. List, personal communication, 1996.

8.1.3. [21] Let P be a computable probability mass function of m identically distributed random variables. Show that the analogue of Theorem 8.1.1 holds. In this case, the entropy is proportional to m, but c_P is $O(\log m)$.

Comments. Hint: use $K(P) = O(\log m)$ and a similar proof to Theorem 8.1.1. Source: T.M. Cover, P. Gács, and R.M. Gray, *Ann. Probab.*, 17:3(1989), 840–865.

8.1.4. [32] Let P be a (possibly incomputable) probability mass function.

(a) Show that $0 \leq \sum_x P(x)K(x|P) - H(P) = O(1)$, where the $O(1)$ term is independent of P and depends only on the reference prefix machine.

(b) Show that $0 \leq \sum_x P(x)K(x|P) - H(P) < 1$ for all P for some appropriate reference prefix machine. This achieves exactly the optimum expected code-word length of the noiseless coding theorem, Theorem 1.11.2 on page 77.

Comments. Hint: in Item (a) use a universal prefix machine with an oracle for $(x, P(x))$ pairs. With an $O(1)$ program to compute a Shannon–Fano code, this machine when given an input y determines whether y is the Shannon–Fano code word for some x. By Lemma 4.3.3 on page 274 such a code word has length $\log 1/P(x) + O(1)$. If this is the case, then the machine outputs x; otherwise, it halts without output. Therefore, $K(x|P) \leq \log 1/P(x) + O(1)$. This establishes the upper bound. The lower bound follows as usual from the noiseless coding theorem, Theorem 1.11.2. Item (b) follows by appropriate modification of the reference machine. Source: the basic Equation 8.3 appears in [P. Gács, *Lecture Notes on Descriptional Complexity and Randomness*, Manuscript, Boston University, 1987; T.M. Cover, P. Gács, and R.M. Gray, *Ann. Probab.*, 17:3(1989), 840–865; W.H. Zurek *Phys. Rev. A*, 40(1989), 4731–4751] but the result is presumably older. A complex argument for Item (a) in a physics setting is [C. Caves, pp. 91–115 in: W.H. Zurek, ed., *Complexity, Entropy and the Physics of Information*, Addison-Wesley, 1991]. Item (b) is spelled out in [R. Schack, *Int. J. Theoret. Phys.*, 36(1997),

209–226]. The last two papers use the nonstandard complexity variant of Example 3.9.2, but this is not required to prove the result.

8.1.5. [39] Prove Theorem 8.1.3.

Comments. Source: D. Hammer, A.E. Romashchenko, A.K. Shen, and N.K. Vereshchagin, *J. Comput. Syst. Sci.*, 60(2000), 442–464.

8.1.6. [30] (a) Show that for every x and y there exists a z such that $2K(z)+K(y|z,x)+K(x|z,y) \leq 2\max\{K(x|y),K(y|x)\}+O(\log n)$, where $n = l(x)+l(y)$.

(b) Show that for every n there are random variables X, Y with 2^n+1 outcomes each such that for every random variable Z we have $2H(Z)+H(X|Y,Z)+H(Y|X,Z) \geq 2\max\{H(X|Y),H(Y|X)+n$.

Comments. Hint for Item (a): use the fact that for every x and y there exists a z such that $K(z) \leq \max\{K(x|y),K(y|x)\}+O(\log n)$, $K(y|z,x) = O(\log n)$, and $K(x|z,y) = O(\log n)$, where $n = K(x|y)+K(y|x)$. Source: C.H. Bennett, P. Gács, M. Li, P.M.B. Vitányi, and W.H. Zurek, *IEEE Trans. Inform. Theory*, 44:4(1998), 1407–1423. Source for Item (b): An.A. Muchnik and N.K. Vereshchagin, *Proc. Int. Comput. Sci. Symp. Russia* (*CSR*), Lect. Notes. Comput. Sci., Vol. 3967, Springer-Verlag, Berlin, 2006, 281–291. While all universally quantified Shannon entropy inequalities hold for Kolmogorov complexity, and conversely, as in Theorem 8.1.3, this is not the case for inequalities of the same type that are second-order quantified as $\forall\exists$. Item (b) shows that the Kolmogorov complexity inequality in Item (a) does not hold for Shannon entropy, and vice versa.

8.1.7. [26] Prove Equation 8.7 on page 615.

8.1.8. [39] Prove Theorem 8.1.6.

Comments. Compare the similar Theorem 5.5.2 dealing with the particular distortion family $\mathcal{L}_n$. Its proof yields a logarithmic error term, but does not seem to generalize to the current, much more general, case. Source: N.K. Vereshchagin and P.M.B. Vitányi, Arxiv cs.IT/0411014; *IEEE Trans. Inform. Theory*, Submitted.

8.1.9. [36] Let $\mathcal{H}_n$ be the Hamming distortion family of strings of length n. Show that $\mathcal{H}_n$ satisfies property 4 on page 615: For every $\delta \leq \delta' \leq \frac{1}{2}$, every Hamming ball of radius δ' can be covered by at most $\alpha_n b_n(\delta')/b_n(\delta)$ Hamming balls of radius δ, where the covering coefficient α_n is $n^{O(1)}$, a polynomial in n.

Comments. A more precise estimate shows that $\alpha_n = O(n^4)$. The exercise implies that the set of all strings of length n can be covered by at most $N = n^{O(1)}2^n/b_n(\delta)$ balls of radius δ. Source: N.K. Vereshchagin and P.M.B. Vitányi, *Ibid.*

8.1.10. [32] The distortion family $\mathcal{H}_n$ satisfies properties 1 through 4 on page 615, where property 4 follows from Exercise 8.1.9.

(a) Show that $r_x(\delta) = g_x(\lceil nH(\delta)\rceil) + O(\log n)$ for every $0 \le \delta \le \frac{1}{2}$.

(b) For every x of length n, we have $r_x(\frac{1}{2}) = O(\log n)$, and $r_x(\delta) - r_x(\delta') \le n(H(\delta) - H(\delta')) + O(\log n)$, for every $0 \le \delta < \delta' \le \frac{1}{2}$.

(c) Let r be a function mapping the set $\{0, 1/n, 2/n, \ldots, \frac{1}{2}\}$ to the natural numbers, satisfying the second inequality of Item (b) without the $O(\log n)$ term, and such that $r(\frac{1}{2}) = 0$. Show that there is a string x of length n such that $r_x(\delta) = r(\delta) + O(\sqrt{n \log n})$ for every $0 \le \delta \le \frac{1}{2}$.

Comments. Hint for Item (a): observe that the complexity of the Hamming ball in $\mathcal{H}_n$ of radius $\delta \le \frac{1}{2}$ with center y is equal to $K(y)$ up to an additive term of order $O(\log n)$. Thus, $g_x(l) + O(\log n) \le r_x(\delta) \le g_x(l-1) + O(\log n)$, where $l = \lceil \log b_n(\delta)\rceil$. By Equation 8.9 on page 617, we have $l = nH(\delta) + O(\log n)$. By Lemma 8.1.3, $g_x(l)$ changes at most by $O(\log n)$ as l changes by $O(\log n)$. Thus the left-hand side and the right-hand side are equal to $g_x(\lceil nH(\delta)\rceil) + O(\log n)$. Hint for Items (b) and (c): use Item (a) and Theorem 8.1.6. Source: N.K. Vereshchagin and P.M.B. Vitányi, *Ibid.*

8.1.11. [35] Let $r(\delta)$ be the function shown in Figure 8.1 on page 616.

(a) Show that $K(r) = O(\log n)$, and that r satisfies the conditions of Exercise 8.1.10.

(b) Show that the rate-distortion graph $r_x(\delta)$ of the string x existing by Exercise 8.1.10 is in the strip of size $O(\sqrt{n \log n})$ of the graph of $r(\delta)$. Therefore $r_x(\delta)$ is almost constant for $\frac{1}{6} \le \delta \le \frac{1}{3}$.

(c) Show that for the x of Item (b), the number of bits we have to transmit for a string within Hamming distance $\frac{1}{3}$ is about equal to that we have to transmit for a string within Hamming distance $\frac{1}{6}$.

(d) Show that only if the fraction of incorrect bits with respect to x is allowed to be sufficiently greater than $\frac{1}{3}$ can we save in the number of transmitted bits.

Comments. Source: N.K. Vereshchagin and P.M.B. Vitányi, *Ibid.*

8.1.12. [25] Consider the rate-distortion family $\mathcal{E}_n$ of Euclidean distortion.

(a) Show that the rate-distortion function of x is related to the structure function g_x by $r_x(\delta) = g_x(n + \log \delta) + O(1)$.

(b) Show that for every n-bit rational number $0 \le x \le 1$, we have $r_x(\frac{1}{2}) = O(1)$, and $0 \le r_x(\delta) - r_x(\delta') \le \log \delta' - \log \delta + O(\log n)$, for all $0 \le \delta \le \delta' \le \frac{1}{2}$.

(c) Let r be a nonincreasing function from the rational numbers to the natural numbers such that $r(\frac{1}{2}) = 0$ and $r(\delta) + \log \delta$ is monotonic nondecreasing. Show that there is an n-bit rational number $0 \leq x \leq 1$ such that $r_x(\delta) = r_x(\delta) + O(\sqrt{n \log n})$ for all $0 \leq \delta \leq \frac{1}{2}$.

Comments. Hint for Items (b) and (c): use Theorem 8.1.6. Source: N.K. Vereshchagin and P.M.B. Vitányi, *Ibid.*

8.1.13. [42] Let $\mathcal{B}$ be a family of nonempty subsets of $\{0,1\}^n$. Let m, k be natural numbers. If a string x of length n is an element of at least 2^m sets of complexity at most k in $\mathcal{B}$, then x is an element of a set in $\mathcal{B}$ of complexity at most $k - m + O(K(\mathcal{B}) + \log n + \log k + \log m)$.

Comments. For $l = 0$ this is Exercise 4.3.10 on page 288 expressed in another form: If a binary string x has at least 2^m descriptions of length at most k, then $K(x) \leq k - m + O(\log k + \log m)$. As usual, p is called a description of x if $U(p) = x$, where U is the reference universal prefix machine. In [N.K. Vereshchagin and P.M.B. Vitányi, *IEEE Trans. Inform. Theory*, 50:12(2004), 3265–3290] this was generalized to all $l \geq 0$: If a binary string belongs to at least 2^m sets B of cardinality 2^l and complexity $K(B) \leq k$, then x belongs to a set A of cardinality 2^l and complexity $K(A) \leq k - l + O(\log m + \log k + \log l)$. Hint: The proof is based on a finite two-player game with complete information. In his moves, the first player generates sets in $\mathcal{B}$ of complexity at most k. The second player marks some of the generated sets in his moves, so that after each of his moves, every string that belongs to at least 2^m of the sets generated by player one belongs also to at least one marked set. If the number of marked sets grows much greater than 2^{k-m}, then the second player loses. We need to prove that the second player has a winning strategy. To this end we can prove that there exists a probabilistic strategy for the second player that has a nonzero probability of winning the game. This makes a deterministic winning strategy for the first player impossible, and therefore (since the game is deterministic) proves the existence of a deterministic winning strategy for the second player. There is also a more complicated constructive proof. Source: N.K. Vereshchagin and P.M.B. Vitányi, Arxiv cs.IT/0411014; *IEEE Trans. Inform. Theory*, Submitted.

8.1.14. [41] Use Exercise 8.1.13 to prove Theorem 8.1.8.

Comments. Source: N.K. Vereshchagin and P.M.B. Vitányi, *Ibid.*

8.1.15. [41] Prove Theorem 8.1.7.

Comments. Hint: use Theorem 8.1.8. Source: N.K. Vereshchagin and P.M.B. Vitányi, *Ibid.*

8.1.16. [27] Use the terminology of Section 8.1.9. Compare $r_x(\delta)$ for $x = x_1 \ldots x_n$ obtained by random sampling a sequence $X_1, \ldots, X_n$ of n random variables, i.i.d. copies of a random variable X, with $r^n(\delta)$ of

Equation 8.10 for the same sequence of n random variables. Let X have a finite range, and all probabilities be rational.

(a) Show that almost surely (i.e., with probability 1) for every δ the limit $\lim_{n\to\infty} r_x(\delta)/n$ is equal to $r^n(\delta)/n$.

(b) Show that for $n \to \infty$ the limit of the expectation $\mathbf{E}r_x(\delta)/n$ is equal to $r^n(\delta)/n$.

Comments. This relates the classical and algorithmic approaches according to traditional information-theoretic concerns. Source: E.-H. Yang and S.-Y. Shen, *IEEE Trans. Inform. Theory*, 39:1(1993), 288–292; J. Muramatsu and F. Kanaya, *IEICE Trans. Fundamentals*, E77-A:8(1994), 1224–1229; D.M. Sow and A. Eleftheriadis, *IEEE Trans. Inform. Theory*, 49:3(2003), 604–608.

8.2 Reversible Computation

Computers can be regarded as engines that must dissipate energy in order to process information. Von Neumann reputedly thought that a computer operating at temperature T must dissipate at least $kT \ln 2$ joules per elementary bit operation, where $k \approx 1.38 \times 10^{-23}$ joule/kelvin is Boltzmann's constant and T is the absolute temperature in kelvin. This is about 2.8×10^{-21} joules at room temperature.

Around 1960, R. Landauer more thoroughly analyzed this question and concluded that it is only logically irreversible operations that must dissipate energy. An operation is *logically reversible* if its inputs can always be deduced from the outputs. Erasure of information is not reversible. Erasing each bit costs $kT \ln 2$ energy when the computer operates at temperature T.

8.2.1 Energy Dissipation

Briefly, Landauer's line of reasoning runs as follows: Distinct logical states of a computer must be represented by distinct physical states of the computer hardware. Each bit has one degree of freedom (0 or 1), corresponding to one or more degrees of freedom in the physical hardware. The n bits collectively have n degrees of freedom. This corresponds to 2^n possible logical states and hence to at least 2^n physical states of the hardware.

Suppose n bits are irreversibly erased (reset to zeros). Before the erasure operation, these n bits could be in any of 2^n possible logical states. After the erasure, they are compressed to just one unique state. According to the second law of thermodynamics, this loss of degrees of freedom of the physical system must be compensated for. If the information is not simply transmitted from the system to the environment, but can

no longer be retrieved, then the compensation must happen by an increase of temperature in the system and its environment, that is, by heat dissipation.

Thus, the only computer operations that are necessarily thermodynamically costly are those that are logically irreversible, that is, operations that map several distinct logical states of the computer onto a common successor, thereby throwing away information about the computer's previous state. Any computation that discards information irreversibly costs energy. For example, a logic gate with more input lines than output lines inevitably loses information, and hence is irreversible and therefore dissipative.

Computer power has roughly doubled every 18 months for the last half-century (Moore's law). This increase in power is due primarily to the continuing miniaturization of the elements of which computers are made, resulting in more and more elementary gates per unit area with higher and higher clock frequency, accompanied by less and less energy dissipation per elementary computing event. Roughly, a linear increase in clock speed is accompanied by a square increase in elements per unit area—so if all elements compute all of the time, then the dissipated energy per time unit rises at a cubic (linear times square) rate in absence of energy decrease per elementary event. The continuing dramatic decrease in dissipated energy per elementary event is what has made Moore's law possible. But there is a foreseeable end to this: There is a minimum quantum of energy dissipation associated with elementary events. This puts a fundamental limit on how far we can go with miniaturization, does it?

The question of how to reduce the energy dissipation of computation determines future advances in computing power. Since battery technology improves by only twenty percent every ten years, low-power computing will similarly govern advances in mobile communication and computing.

Over the last fifty years, the energy dissipation per logic operation has been reduced by approximately a factor of ten every five years. Such an operation in 1988 dissipated about 1/10 picojoule (1 picojoule is 10^{-12} joule) versus around 10^9 picojoule in 1945 [R.W. Keyes, *IBM J. Res. Devel.*, 32(1988), 24–28].

Extrapolations of current trends suggest that reduction of the energy dissipation per logic operation below kT (thermal noise, on the order of 10^{-8} picojoule at room temperature) becomes a relevant issue by 2015. Even at kT level, a future device containing 1 trillion (10^{12}) gates operating at room temperature at 1 terahertz (10^{12}) switching all gates all of the time dissipates about 3000 watts. This is difficult to cool.

The drive for ever greater computing power through more densely packed logic circuits will eventually require that we find methods other than cooling. An alternative way is to develop reversible logic that computes (almost) without energy dissipation.

Logical reversibility does not imply dissipation freedom. A computer may compute in a logically reversible manner and yet dissipate heat.

But the laws of physics allow for technologies to make logical reversible computers operate in a dissipationless manner. Logically reversible computers are those built from Fredkin gates, or the reversible Turing machine, discussed later. Thought experiments exhibit a computer that is both reversible and dissipationless. An example is the billiard-ball computer discussed below.

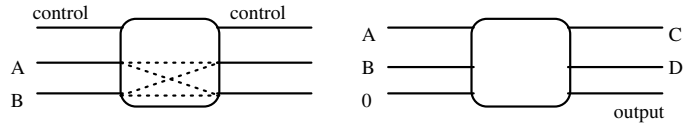
8.2.2 Reversible Logic Circuits

To build a computer we need only two types of logical gates: AND, NOT; all other gates can be built from AND and NOT. The NOT gate is already logically reversible. But the AND gate is not. The AND and OR gates have each two inputs but only one output. Hence, they must lose information.

Can we design something like an AND gate that has the same number of input lines and output lines and is reversible? In principle, we need only to add garbage output lines. This is easy, and there are many such gates. However, we would like to find a universal reversible gate from which all Boolean gates can by synthesized. One such universal reversible gate is the so-called Fredkin gate as shown in Figure 8.3(a). If the control bit is 0, then the input values of A and B are transmitted unaltered to the outputs on the same level; and if the control bit is 1, then they are switched to the opposite output.

The Fredkin gate is universal in the sense that it can be used to construct all other Boolean gates in a reversible variant. For example, Figure 8.3(b) shows a reversible AND gate built from a Fredkin gate. Here, the top wire marked by A and C is the control wire of the Fredkin gate. The bottom wire marked 0 has a constant 0 input. The inputs to the AND gate are marked A and B, the single output wire is marked 'output.' If $A = 0$, then the constant input 0 is transmitted unaltered to the output. If $A = 1$, then the input from B is transmitted unaltered to the output. Hence, the output is A AND B.

Fredkin's gate and the AND gate in Figure 8.3 are reversible because we have added garbage output bits in order to ensure that the inputs can always be deduced from the outputs. Therefore, in principle they do not need to dissipate energy. In theory one can possibly build a dissipationless computer using Fredkin gates or reversible AND and NOT



(a) A Fredkin gate (b) A reversible AND gate

FIGURE 8.3. Reversible Boolean gates

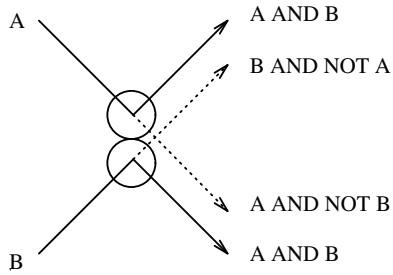


FIGURE 8.4. Implementing reversible AND gate and NOT gate

gates. There is no *thermodynamic* reason that such a computer should cost any energy. A computation would be started by an energy jolt, say some amount of electricity, proceed without dissipating energy, and at the end of the computation simply return the same amount of electricity to the source.

8.2.3 Reversible Ballistic Computer

Consider an idealized computer using elastic frictionless billiard balls. The presence of a ball represents a 1, and no ball represents a 0. The ballistic computer contains walls at some positions that perfectly elastically reflect the balls. All collisions between balls are perfectly elastic as well. Between the collisions, the balls travel in straight lines with constant speed, as in Newtonian mechanics. For example, we can think of the balls as idealized molecules. To start the computation, if an input bit is 1, we fire a ball, if an input bit is 0, we do not fire a ball. All input balls are fired simultaneously, and at the same speed.

Figure 8.4 implements an AND gate for inputs A and B. If we set $B = 1$, then we obtain a NOT gate for A (and setting $A = 1$ yields a NOT gate for B). We will also need the constructions in Figure 8.5 using walls to deflect a ball's path, to shift a path, to delay the ball's motion without changing its original direction, and to allow two lines of motion to cross.

It is possible to emulate any computation using the above gadgets. Suppose the setup lets all of the balls reach the output end simultaneously. After we observe the output, we can simply reflect back all the output balls, including the many garbage balls, to reverse the computation. The billiard balls will then come out of the ballistic computer exactly where we sent them in, with the same speed. The kinetic energy can be returned to the device that kicked the balls in. Then the computer is ready for a next round of dissipationless action. An example input–output+garbage scheme for a ballistic computer is shown in Figure 8.6.

A ballistic computer with molecules as balls (the molecular computer) is extremely unstable in principle. Any small initial error in the position or speed

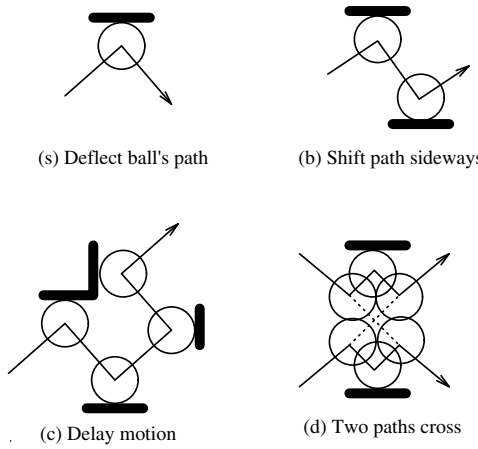


FIGURE 8.5. Controlling billiard-ball movements

of the balls is amplified by a factor of 2 in each collision. Due to the quantum-mechanical uncertainty relation there are unavoidable errors of size at least about Planck's constant. After a few dozen collisions, the whole computer deteriorates.

To prevent deterioration, one may use objects at the quantum level that are more stable. R.P. Feynman suggested the use of electron spin orientation in [*Int. J. Theoret. Phys.*, 21(1982), 467–488; *Optics News*, 11(1985), 11]. Other quantum-mechanical computer proposals are [P.A. Benioff, *Int. J. Theoret. Phys.*, 21(1982), 177–202; *Ann. New York Acad. Sci.*, 480(1986), 475–486; N. Margolus, pp. 273–287 in W.H. Zurek, ed., *Complexity, Entropy and the Physics of Information*, Addison-Wesley, 1991].

C.H. Bennett [*Int. J. Theoret. Phys.*, 21:2(1982), 905–940] suggested a Brownian computer, where the computing particles are larger than molecules but small enough to be subject to Brownian motion. The idea is to use thermal noise to drive the computing balls. The computing balls can move on a fixed computation trajectory between start and finish of the computation. By Brownian motion they perform a random walk, back and forth along the trajectory, until they finally arrive at the finish. These computers do dissipate energy, but this can in principle be made arbitrarily small.

C.H. Bennett and R. Landauer [*Scientific American*, 253(July 1985), 48–56] also described how to construct an idealized Fredkin gate and given such gates, another alternative way of constructing a billiard-ball computer. All the balls are linked together and pushed forward by one mechanism. The balls move along pipes. The entire assembly is immersed in an ideal viscous fluid. The frictional forces that act on the balls will be proportional to their velocity; there will be no static friction.

If we move the balls very slowly, then the frictional force will be very weak. The energy that must be expended to work against friction is equal to the product of the frictional force and the distance the ball traveled. Thus, we can use as little energy as we wish simply by slowing down the computation.

There is no minimum amount of energy that must be expended to perform any computation.

In [R.C. Merkle, *Nanotechnology*, 4(1993), 21–40], two methods to realize such reversible computations using electronic switching devices in conventional technologies (such as nMOS, CMOS, and charge-coupled devices) are proposed. In fact, such reversible almost dissipationless electronics have already been used in existing laboratory computers. (They happened to have been designed with other aims in mind.)

8.2.4 Thermodynamics of Computing

Using the physical theory of reversible computation, the simple difference $K(x) - K(y)$ turns out to be an appropriate (universal, antisymmetric, and transitive) measure of the amount of thermodynamic work required to transform string x into string y by the most efficient process.

Thermodynamics, among other things, deals with the amounts of heat and work ideally required, by the most efficient process, to convert one form of matter to another. For example, at 0°C and atmospheric pressure, it takes 80 calories of heat and no work to convert a gram of ice into water at the same temperature and pressure.

From an atomic point of view, the conversion of ice to water at 0°C is a reversible process, in which each melting water molecule gains about 3.8 bits of entropy (representing the approximately $2^{3.8}$-fold increased freedom of motion it has in the liquid state), while the environment loses 3.8 bits.

During this ideal melting process, the entropy of the universe remains constant, because the entropy gain by the melting ice is compensated by an equal entropy loss by the environment. Perfect compensation takes

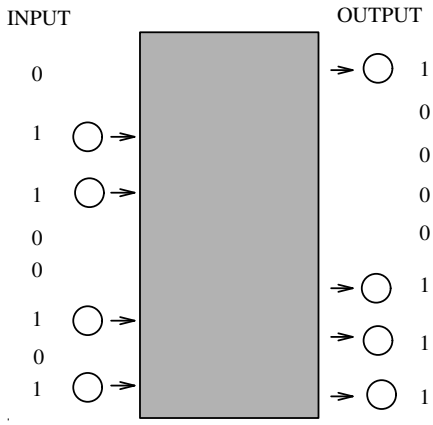


FIGURE 8.6. A billiard-ball computer

place only in the limit of slow melting, with an infinitesimal temperature difference between the ice and the water.

Rapid melting, for example, when ice is dropped into hot water, is thermodynamically irreversible and inefficient, with the environment (the hot water) losing less entropy than the ice gains, resulting in a net and irredeemable entropy increase for the combined system. Strictly speaking, the microscopic entropy of the universe as a whole does not increase, being a constant of motion in both classical and quantum mechanics. Rather, what happens when ice is dropped into hot water is that the marginal entropy of the (ice + hot water) system increases, while the entropy of the universe remains constant, due to a growth of mutual information mediated by the subtle correlations between the (ice + hot water) system and the rest of the universe. In principle, these correlations could be harnessed and redirected so as to cause the warm water to refreeze, but in practice the melting is irreversible.

Turning again to ideal reversible processes, the entropy change in going from state x to state y is an antisymmetric function of x and y; thus, when water freezes at 0°C by the most efficient process, it gives up 3.8 bits of entropy per molecule to the environment. When more than two states are involved, the entropy changes are transitive: thus the entropy change per molecule of going from ice to water vapor at 0°C (+32.6 bits) plus that for going from vapor to liquid water (−28.8 bits) sum to the entropy change for going from ice to water directly.

Because of this antisymmetry and transitivity, entropy can be regarded as a thermodynamic potential, or state function: each state has an entropy, and the entropy change in going from state x to state y by the most efficient process is simply the entropy difference between states x and y.

Thermodynamic ideas were first successfully applied to computation by Landauer (Section 8.2.1). According to Landauer's principle, an operation that maps an unknown state randomly chosen from among n equiprobable states onto a known common successor state must be accompanied by an entropy increase of $\log_2 n$ bits in other, noninformation-bearing degrees of freedom in the computer or its environment. At room temperature, this is equivalent to the production of $kT \ln 2$ (about 6.7×10^{-22}) calories of waste heat per bit of information discarded. The point here is the change from ignorance to knowledge about the state, that is, the gaining of information and not the erasure in itself (instead of erasure one could consider measurement that would make the state known).

Landauer's principle follows from the fact that such a logically irreversible operation would otherwise be able to decrease the thermodynamic entropy of the computer's data without a compensating entropy

increase elsewhere in the universe, thereby violating the second law of thermodynamics.

Converse to Landauer's principle is the fact that when a computer takes a physical *randomizing* step, such as tossing a coin, in which a single logical state passes stochastically into one of n equiprobable successors, that step can, if properly harnessed, be used to remove $\log_2 n$ bits of entropy from the computer's environment. Models have been constructed (Section 8.2.1) obeying the usual conventions of classical, quantum, and thermodynamic thought experiments showing both the ability in principle to perform logically reversible computations in a thermodynamically reversible fashion, and the ability to harness entropy increases due to data randomization within a computer to reduce correspondingly the entropy of its environment.

In view of the above considerations, it seems reasonable to assign each string x an effective thermodynamic entropy equal to its complexity $K(x)$. A computation that erases an n-bit random string would then reduce its entropy by n bits, requiring an entropy increase in the environment of at least n bits, in agreement with Landauer's principle.

Conversely, a randomizing computation that starts with a string of n zeros and produces n random bits has, as its typical result, an algorithmically random n-bit string x, that is, one for which $K(x) \approx n$. By the converse of Landauer's principle, this randomizing computation is capable of removing up to n bits of entropy from the environment, again in agreement with the identification of the algorithmic complexity K and the thermodynamic entropy.

What about computations that start with one (randomly generated or unknown) string x and end with another string y? By the transitivity of entropy changes one is led to say that the thermodynamic cost, that is, the minimal entropy increase in the environment, of a transformation of x into y should be

$$W(y|x) = K(x) - K(y),$$

because the transformation of x into y could be thought of as a two-step process in which one first erases x, and then allows y to be produced by randomization. By the elementary properties of self-delimiting programs, this cost measure is transitive within an additive constant. A similar antisymmetric measure of the thermodynamic cost of data transformations, such as

$$W'(y|x) = K(x|y) - K(y|x),$$

is slightly nontransitive. There are strings x with $K(x^*|x) \approx K(K(x))$, where x^* is the minimal program for x (Theorem 3.8.1 on page 242).

According to the W' measure, erasing such an x via the intermediate x^* would generate $K(K(x))$ less entropy than erasing it directly, while for the W measure the two costs would be equal within an additive constant. Indeed, erasing in two steps would cost only $K(x|x^*) - K(x^*|x) + K(x^*|\epsilon) - K(\epsilon|x^*) = K(x) - K(x^*|x)$, while erasing in one step would cost $K(x|\epsilon) - K(\epsilon|x) = K(x)$. (Everything up to an additive constant.)

Finally, we consider entropy changes in nonideal computations. Consider the thermodynamics of an intelligent demon or engine that has some capacity to analyze and transform data x before erasing it. If the demon erases a random-looking string, such as the digits of π, without taking the trouble to understand it, it will commit a thermodynamically irreversible act, in which the entropy of the data is decreased very little, while the entropy of the environment increases by a full n bits. On the other hand, if the demon recognizes the redundancy in π, it can transform π to a short string by a reversible computation, and thereby accomplish the erasure at very little thermodynamic cost.

More generally, given unlimited time, a demon could approximate the upper semicomputable function $K(x)$ and so compress a string x to size $K(x)$ before erasing it. But in limited time, the demon will not be able to compress x so much and will have to generate more entropy to get rid of it. This tradeoff between speed and thermodynamic efficiency is superficially similar to the tradeoff between speed and efficiency for physical processes such as melting, but the functional form of the tradeoff is very different.

For typical physical state changes such as melting, the excess entropy produced per molecule goes to zero inversely in the time t allowed for melting to occur. But the time-bounded prefix complexity $K^t(x)$, that is, the size of the smallest program to compute x in time at most t, in general approaches $K(x)$ with incomputable slowness as a function of t and x. A formal result along these lines is stated as Exercise 8.3.5.

8.2.5 Reversible Turing Machines

In the standard model of a Turing machine (Section 1.7) the elementary operations are rules in quadruple format (p, s, a, q), meaning that if the finite control is in state p and the machine scans tape symbol s, then the machine performs action a and subsequently the finite control enters state q. Such an action a consists in either printing a symbol s' in the tape square scanned, or moving the scanning head one tape square left or right.

Quadruples are said to *overlap in domain* if they cause the machine in the same state and scanning the same symbol to perform different actions. A *deterministic Turing machine* is defined as a Turing machine with quadruples no two of which overlap in domain.

Now consider the special-format (deterministic) Turing machines using quadruples of two types: *read/write* quadruples and *move* quadruples. A read/write quadruple (p, a, b, q) causes the machine in state p scanning tape symbol a to write symbol b and enter state q. A move quadruple $(p, *, \sigma, q)$ causes the machine in state p to move its tape head by $\sigma \in \{-1, +1\}$ squares and enter state q, oblivious to the particular symbol $*$ in the currently scanned tape square. (Here -1 means 'one square left', and $+1$ means 'one square right'.) Quadruples are said to *overlap in range* if they cause the machine to enter the same state and either both write the same symbol or (at least) one of them moves the head. Said differently, quadruples that enter the same state overlap in range unless they write different symbols. A *reversible Turing machine* is a deterministic Turing machine with quadruples no two of which overlap in range. A k-tape reversible Turing machine uses $(2k+2)$ tuples, which for every tape separately, select a read/write or move on that tape. Moreover, any two tuples can be restricted to some single tape where they don't overlap in range.

To show that every partial recursive function can be computed by a reversible Turing machine one can proceed as follows. Take the standard irreversible Turing machine computing that function. We modify it by adding an auxiliary storage tape called the 'history tape'. The quadruple rules are extended to 6-tuples to additionally manipulate the history tape. To be able to reversibly undo (retrace) the computation deterministically, the new 6-tuple rules have the effect that the machine keeps a record on the auxiliary history tape consisting of the sequence of quadruples executed on the original tape. Reversibly undoing a computation entails also erasing the record of its execution from the history tape.

This notion of reversible computation means that only one-to-one recursive functions can be computed. To reversibly simulate t steps of an irreversible computation from x to $f(x)$ one reversibly computes from input x to output $\langle x, f(x)\rangle$. We analyze the simplest time–space overhead. Say the simulation takes $t' = O(t)$ time. Since the reversible simulation at some time instant has to record the entire history of the irreversible computation, its space use increases linearly with the number of simulated steps t. That is, if the simulated irreversible computation uses s space, then for some constant $c > 1$ the simulation uses $t' \approx c + ct$ time and $s' \approx c + c(s + t)$ space. After computing from x to $f(x)$ the machine reversibly copies $f(x)$, reversibly undoes the computation from x to $f(x)$, erasing its history tape in the process, and ends with one copy of x and one copy of $f(x)$ in the format $\langle x, f(x)\rangle$ and otherwise empty tapes.

Let $T_1, T_2, \ldots$ be the standard enumeration of prefix Turing machines with input tape symbols 0 and 1 only (no blanks), as defined in Chap-

ter 3. Each such Turing machine has the property that the set of programs for which it halts is a prefix set (no element in the set is a proper prefix of another element in the set). These (in general irreversible) prefix machines compute all and only partial recursive prefix functions $\phi_1, \phi_2, \ldots$ (Definition 3.1.1 on page 200).

The (tedious) construction of reversible Turing machines can be found elsewhere and is of no direct importance to us. We can find a standard enumeration of them as a subsequence of the list of Turing machines. We can even find a standard enumeration of the reversible prefix machines as a subsequence of the list of prefix machines.

We list some relevant properties. Let ψ_i be the partial recursive function computed by the ith such reversible prefix machine. Let $\langle\cdot\rangle$ be a bijective recursive pairing mapping over the integers, which can be decoded from left to right by a prefix machine. Among the more important properties of reversible prefix machines are the following:

Universal reversible prefix machine: There exists a reversible prefix machine that is universal, say UR, computing ψ_0, such that for all k and x, we have that $\psi_0(\langle k, x\rangle) = \langle k, \psi_k(x)\rangle$.

Irreversible to reversible: Two irreversible algorithms, one for computing y from x and the other for computing x from y, can be efficiently combined to obtain a reversible algorithm for computing y from x. More formally, for any two indices i and j one can effectively obtain an index k such that for any strings x and y, if $\phi_i(x) = y$ and $\phi_j(y) = x$, then $\psi_k(x) = y$.

Saving input copy: From any index i one may obtain an index k such that ψ_k has the same domain as ϕ_i and for every x, we have $\psi_k(x) = \langle x, \phi_i(x)\rangle$. In other words, an arbitrary prefix machine can be simulated by a reversible one that saves a copy of the irreversible machine's input in order to ensure a global one-to-one mapping.

Efficiency: The above simulation can be performed rather efficiently. In particular, for any $\epsilon > 0$ one can find a reversible simulating machine that runs in time $O(t^{1+\epsilon})$ and space $O(s \log t)$ compared to the time t and space s of the irreversible machine being simulated.

One-to-one functions: From every index i one may effectively obtain an index k such that if ϕ_i is one-to-one, then $\psi_k = \phi_i$. The reversible Turing machines $\{\psi_k\}$ therefore provide an effective enumeration of all one-to-one partial recursive functions.

Definition 8.2.1 Let $\psi(p, x)$ be a partial recursive function satisfying the following: For every p, the function $\psi(p, x)$ is one-to-one as a function of x; for every

x, the set of p's such that $\psi(p,x) < \infty$ is a prefix set; and for each y the set of p's such that for some x we have $\psi(p,x) = y$ is a prefix set. Then ψ is a *reversible prefix partial recursive function.*

Such a ψ may be thought of as the function computed by a reversible prefix machine that performs a one-to-one mapping on $x \leftrightarrow y$ under the control of a program p that acts like a catalyst in that it remains on the program tape throughout the computation. Such a machine has a one-way read-only program tape initially containing program p, a read-only conditional data tape initially containing input x, and a one-way write-only output tape containing y on termination.

Any other work tapes used during the computation are supplied in blank condition at the beginning of the computation and must be left blank at the end of the computation. The program tape's head begins and ends scanning the leftmost square of the program, which is self-delimiting both for forward computations from each input x and for backward computations from each output y.

A *universal reversible prefix machine UR* whose program size is minimal to within an additive constant can readily be shown to exist.

Definition 8.2.2 The *reversible prefix complexity* is $KR(y|x) := \min\{l(p) : UR(p,x) = y\}$.

Exercises

8.2.1. [20] Construct reversible OR, NOT, and XOR gates using Fredkin gates.

8.2.2. [36] Prove the existence of a universal reversible Turing machine.

Comments. Source: C.H. Bennett, *IBM J. Res. Develop.*, 17(1973), 525–532. If the original computation takes t steps and uses s space then Bennett's simulation requires $t' = \Theta(t)$ steps and $s' = \Theta(st)$ space.

8.2.3. [34] Let an irreversible computation use t steps and s space.

(a) Show how to simulate it reversibly using $t' = \Theta(t^{1+\epsilon}/s^{\epsilon})$ steps and $s' = \Theta(c(\epsilon)s(1 + \log t/s))$ space with $c(\epsilon) = \epsilon 2^{1/\epsilon}$ for any $\epsilon > 0$ using always the same simulation method with different parameters. Typically, $\epsilon = \log 3$.

(b) Show that Item (a) implies that each irreversible computation using s space can be simulated by a reversible computation using s^2 space in $t' = \Theta(t^{1+\epsilon})$ time.

Comments. Hint: in Item (a) do not save the entire history of the irreversible computation, but break up the simulated computation into segments of about s steps and save in a hierarchical manner *checkpoints* consisting of complete instantaneous descriptions of the simulated

machine (entire tape contents, tape heads positions, state of the finite control). After a later checkpoint is reached and saved, the simulating machine reversibly undoes its intermediate computation reversibly erasing the intermediate history and reversibly canceling the previously saved checkpoint. Subsequently, the computation is resumed from the new checkpoint onward. The reversible computation simulates k^n segments of length m of irreversible computation in $(2k-1)^n$ segments of length $\Theta(m+s)$ of reversible computation using $n(k-1)+1$ checkpoint registers using $\Theta(m+S)$ space each, for each k, n, m. In this way, it is established that there are various tradeoffs possible in time–space between $t' = \Theta(t)$ and $s' = \Theta(ts)$ at one extreme ($k = 1, m = t, n = 1$) and the t' and s' in Item (a) at the other extreme using always the same simulation method but with different parameters k, n, where $\epsilon = \log_k(2k-1)$ and $m = \Theta(s)$. Typically, for $k = 2$ we have $\epsilon = \log 3$. Since for $t > 2^s$ the machine goes into a computational loop, we always have $s \leq \log t$. This proves Item (b). Source: for Items (a) and (b) [C.H. Bennett, *SIAM J. Comput.*, 18(1989), 766–776; R.Y. Levine and A.T. Sherman, *SIAM J. Comput.*, 19(1990), 673–677]. In [M. Li and P.M.B. Vitányi, *Proc. Royal Soc. London, Ser. A*, 452(1996), 769–789], reversible simulations are investigated using reversible pebble games. It is shown that the simulations as in Item (a) are optimal in the sense of simulating the greatest number of steps using least space overhead $s' - s$. It is shown that the space overhead can be reduced at the cost of limited irreversible erasing. By Landauer's principle this implies a space–energy tradeoff. For further results on time–space bounds for reversible simulation of irreversible computation, see [M. Li, J.T. Tromp, and P.M.B. Vitanyi, *Physica D*, 120(1998) 168–176; K.J. Lange, P. McKenzie, and A. Tapp, *J. Comput. Syst. Sci.*, 60:2(2000), 354–367; H.M. Buhrman, J.T. Tromp, and P.M.B. Vitanyi , *J. Physics A: Math. and General*, 34(2001), 6821–6830].

8.3 Information Distance

Kolmogorov complexity is a measure of absolute information content of individual objects. It is desirable to have a similar measure of absolute information distance between individual objects. Such a notion should be universal in the sense that it covers all other alternative or intuitive notions of computable informational distance as special cases. Such a notion should also be asymptotically machine-independent and can serve as an absolute measure of the informational, or cognitive, distance between discrete objects x and y. A universal informational distance between two strings satisfying these requirements is the minimal quantity of information sufficient to translate between x and y, generating either string effectively from the other. The universality requirement necessarily makes our information distance not computable. However, the study of the abstract properties of such absolute information distance will lead to applicable formulas and practical approaches.

Intuitively, the minimal information distance between x and y is the length of the shortest program for a universal computer to transform x into y and y into x. This program then functions in a catalytic manner, being retained in the computer before, during, and after the computation. This measure will be shown to be, up to a logarithmic additive term, equal to the *maximum* of the conditional Kolmogorov complexities. The conditional complexity $K(y|x)$ itself is unsuitable as optimal information distance because it is asymmetric: $K(\epsilon|x)$, where ϵ is the empty string, is small for all x, yet obviously a long random string x is not close to the empty string. The asymmetry of the conditional complexity $K(x|y)$ can be remedied by defining the algorithmic informational distance between x and y to be the sum of the relative complexities, $K(y|x)+K(x|y)$. The resulting metric will overestimate the information required to translate between x and y in case there is some redundancy between the information required to get from x to y and the information required to get from y to x.

8.3.1 Definitions

Let $T_1, T_2, \ldots$ be the standard enumeration of prefix machines of Example 3.1.1 on page 201. For a partial recursive function ϕ computed by T, let

$$E_\phi(x,y) = \min\{l(p) : \phi(p,x) = y,\ \phi(p,y) = x\}.$$

Lemma 8.3.1 *There is a universal prefix machine U computing ϕ_0 such that for every partial recursive prefix function ϕ and all x, y,*

$$E_{\phi_0}(x,y) \leq E_\phi(x,y) + c_\phi,$$

where c_ϕ is a constant that depends on ϕ but not on x and y.

Proof. This is a consequence of the existence of a universal prefix machine and is identical to the proof of Theorem 2.1.1 on page 105. □

By Lemma 8.3.1, for every two universal prefix machines computing ϕ_0 and ψ_0, we have for all x, y that $|E_{\phi_0}(x,y) - E_{\psi_0}(x,y)| \leq c$, with c a constant depending on ϕ_0 and ψ_0 but not on x and y. Thus the following definition is machine-independent.

Definition 8.3.1 Fixing a particular universal prefix machine U as reference machine, we define *information distance* as

$$E_0(x,y) = \min\{l(p) : U(p,x) = y, U(p,y) = x\}.$$

Definition 8.3.2 The *max distance* between x, y is $E_1(x,y) = \max\{K(x|y), K(y|x)\}$.

We shall prove that up to an additive logarithmic term, the information distance E_0 is equal to the max distance.

8.3.2 Maximal Overlap

To what extent can the information required to compute x from y be made to overlap with that required to compute y from x? In some simple cases, complete overlap can be achieved, so that the same minimal program suffices to compute x from y as to compute y from x.

Example 8.3.1 If x and y are independent random binary strings of the same length n (that is, up to additive constants $K(x|y) = K(y|x) = n$), then their bitwise exclusive-or $x \oplus y$ serves as a minimal program for both computations. Similarly, if $x = uv$ and $y = vw$, where u, v, and w are independent random strings of the same length, then $u \oplus w$ is a minimal program to compute either string from the other.

Now suppose that more information is required for one of these computations than for the other, say,

$$K(y|x) > K(x|y).$$

Then the minimal programs cannot be made identical because they must be of different sizes. Nevertheless, in simple cases, the overlap can still be made complete, in the sense that the larger program (for y given x) can be made to contain all the information in the smaller program, as well as some additional information. This is so when x and y are independent random strings of unequal length, for example u and vw above. Then $u \oplus v$ serves as a minimal program for u from vw, and $(u \oplus v)w$ serves as one for vw from u. ◇

The following *conversion theorem* asserts the existence of a difference string p of length $l(p) = \max\{K(x|y), K(y|x)\}$, up to an additive logarithmic term, that converts both ways between x and y and at least one of these conversions is optimal. If $K(x|y) = K(y|x)$, then the conversion is optimal in both directions. Note that the use of prefix complexity is not essential; the theorem also holds with K replaced by C except for the term $K(K(x|y), K(y|x))$, which then becomes $K(C(x|y), C(y|x))$.

Theorem 8.3.1 *Let x and y be strings such that $K(y|x) \geq K(x|y)$. There is a string r of length $K(y|x) - K(x|y)$ such that*

$$E_0(rx, y) = K(x|y) + K(K(x|y), K(y|x)) + O(1).$$

Proof. Let $K(x|y) = k_1$ and $K(y|x) = k_2$, and $l = k_2 - k_1 \geq 0$. Given k_1, k_2, we can enumerate the set

$$S = \{(u, v) : K(u|v) \leq k_1, K(v|u) \leq k_2\}.$$

Without loss of generality, assume that S is enumerated without repetition. Now consider a dynamic graph $G = (V, E)$, where V is the set of binary strings, and E is a dynamically growing set of edges that starts out empty.

Whenever a pair (u, v) is enumerated, we add an edge $e = \{ru, v\}$ to E. Here, r is chosen to be the $\lfloor i/2^{k_1} \rfloor$th binary string of length l, where i is the number of times we have enumerated a pair with u as the first element. So the first 2^{k_1} times we enumerate a pair $(u, \cdot)$ we choose $r = 0^l$, for the next 2^{k_1} times we choose $r = 0^{l-1}1$, and so on. The condition $K(v|u) \leq k_2$ implies that $i \leq 2^{k_2}$, hence $i/2^{k_1} \leq 2^l$, so this choice is well defined. (The device of adding a prefix r to u to obtain ru takes care that there are at most 2^{k_1} incident edges with ru, not 2^{k_2} as could be the case with u without the prefix r.)

In addition, we 'color' edge e with a binary string of length $k_1 + 3$. Call two edges *adjacent* if they have a common endpoint. If c is the minimum color not yet appearing on any edge adjacent to either ru or v, then e is colored c. Since the degree of every node is bounded by 2^{k_1} (when acting as an ru) plus 2^{k_1} (when acting as a v), a color is always available.

A *matching* is a maximal set of nonadjacent edges. Note that the colors partition E into at most 2^{k_1+3} matchings, since no edges of the same color are ever adjacent. Since the pair (x, y) in the statement of the theorem is necessarily enumerated, there is some r of length l and color c such that the edge $\{rx, y\}$ is added to E with color c.

Knowing k_1, k_2, c and either of the nodes rx and y, one can dynamically reconstruct G, find the unique c-colored edge adjacent to the given node, and output the neighbour. Therefore a self-delimiting program of size $k_1 + K(k_1, k_2) + O(1)$ suffices to compute in either direction between rx and y. □

Example 8.3.2 The theorem asserts that up to a logarithmic additive term, the information required to translate between two strings can always be represented in the maximally overlapping way of Example 8.3.1. Namely, there is a string q of length $k_1 + O(\log k_1)$ and a string r of length $l + \log l$ such that q serves as the minimal program both from rx to y and from y to rx. This means that the information required to pass from x to y is always maximally correlated with the information required to get from y to x. It is therefore not necessary that a large amount of information is required to get from x to y and a large *but independent* amount of information is required to get from y to x. (It is very important here that the time of computation is completely ignored: this is why this result does not contradict the idea of one-way functions.)

The process of going from x to y may be broken into two stages. First, add the string r; second, use the difference program q between rx and

y. In the reverse direction, first use q to go from y to rx; second, erase r. Thus, the computation from x to y needs both q and r, while the computation from y to x needs only q as program. $\diamond$

Corollary 8.3.1 Let x, y be strings with $K(y|x) \geq K(x|y)$. Let p be a shortest program to compute y from x and let q be a shortest program to compute x from y. Then we can choose p and q such that $p = rq$, achieving maximum overlap of $l(q) = \min\{K(x|y), K(y|x)\}$ between the programs p and q.

The foregoing is true of ordinary computations, but if one insists that the computation be performed reversibly, that is, by a machine whose transition function is one-to-one, then the full program $p = rq$ is needed to perform the computation in either direction. This is because reversible computers cannot get rid of unwanted information simply by erasing it as ordinary irreversible computers do. If they are to get rid of unwanted information at all, they must cancel it against equivalent information already present elsewhere in the computer.

Corollary 8.3.2 $E_0(x, y) = \max\{K(x|y), K(y|x)\} + O(\log \max\{K(x|y), K(y|x)\})$.

8.3.3 Universality

Suppose we want to quantify how much objects differ in terms of a given feature, for example the length in bits of files, the number of beats per second in music pieces, the number of occurrences of a given base in the genomes. Every specific feature induces a distance, and every specific distance measure can be viewed as a quantification of an associated feature difference. Every reasonable distance to measure how much one object resembles another one should be an effectively approximable positive function of x and y satisfying a reasonable density condition and obeying the triangle inequality. It turns out that E_1 is minimal up to an additive constant among all such distances. Hence, it is a universal information distance that accounts for every effective resemblance between two objects.

Let us consider an example of measuring distance between two pictures. Identify digitized black-and-white pictures with binary strings. There are many distances defined for binary strings, for example, the Hamming distance and the Euclidean distance. Such distances are sometimes appropriate. For instance, if we take a binary picture and change a few bits in that picture, then the changed and unchanged pictures have small Hamming or Euclidean distance, and they do look similar.

However, this is not always the case. The positive and negative prints of a photo have the largest possible Hamming and Euclidean distance, yet they look similar to us. Also, if we shift a picture one bit to the

right, again the Hamming distance may increase considerably, but the two pictures remain similar.

Many approaches to pattern recognition try to define resemblance metrics with respect to pictures, language sentences, vocal utterances, and so on. We have seen evidence that $E_1(x,y) = \max\{K(x|y), K(y|x)\}$ is a natural way to formalize a notion of the minimal algorithmic informational distance between x and y. Let us show that the distance E_1 is, in a sense, minimal among all reasonable resemblance measures.

Let Ω be a nonempty set and $\mathcal{R}^+$ the set of nonnegative real numbers. A *distance function* on Ω is a function $D : \Omega \times \Omega \to \mathcal{R}^+$. It is a *metric* if it satisfies the metric (in)equalities:

- $D(x,y) = 0$ iff $x = y$ (the identity axiom)
- $D(x,y) = D(y,x)$ (the symmetry axiom), and
- $D(x,y) \leq D(x,z) + D(z,y)$ (the triangle inequality).

The value $D(x,y)$ is called the *distance* between $x, y \in \Omega$. A familiar example of a distance that is also a metric is the Euclidean metric, the everyday distance $e(a,b)$ between two geographical objects a, b expressed in, say, meters. Clearly, this distance satisfies the properties $e(a,a) = 0$, $e(a,b) = e(b,a)$, and $e(a,b) \leq e(a,c) + e(c,b)$ (for instance, $a =$ Amsterdam, $b =$ Beijing, and $c =$ Chicago.) Our goal is to generalize such a concept of distance from our physical space to the cyberspace and characterize the set of all reasonable distance functions that would measure informational distances between objects.

A priori we allow asymmetric distances, but we would like to exclude degenerate distance measures such as $D(x,y) = 1$ for all $x \neq y$. For each x and d, we want only finitely many elements y at a distance d from x. Exactly how fast we want the distances of the strings y from x to go to ∞ is not important; it is only a matter of scaling. For convenience, we will require the following *density conditions*:

$$\sum_{y:y\neq x} 2^{-D(x,y)} \leq 1, \quad \sum_{x:x\neq y} 2^{-D(x,y)} \leq 1. \tag{8.11}$$

We consider only distances that are computable in some broad sense. This condition will not be seen as unduly restrictive. As a matter of fact, only upper semicomputability of $D(x,y)$ will be required (Section 4.1).

Definition 8.3.3 An *admissible information distance*, $D(x,y)$, is a total, possibly asymmetric, nonnegative function on the pairs x, y of binary strings that is 0 if and only if $x = y$, is upper semicomputable, and satisfies the density requirement in Equation 8.11.

Example 8.3.3 The Hamming distance between two strings $x = x_1 \dots x_n$ and $y = y_1 \dots y_n$ was defined in Equation 8.8 on page 616 as $d(x,y) = \frac{1}{n} d(\{i : x_i \neq y_i\})$. This distance does not directly satisfy the density requirements of Equation 8.11. With minor modification, we can scale it to satisfy these requirements. In representing the Hamming distance d between x and y, strings of equal length n differing in positions $i_1, \dots, i_{dn}$, we can use a simple prefix-free encoding of $(n, dn, i_1, \dots, i_d)$ in $H_n(x,y) = 2\log n + 4\log\log n + 2 + dn \log n$ bits. We encode n and dn prefix-free in $\log n + 2\log\log n + 1$ bits each and then the literal indexes of the actual flipped-bit positions. Thus, $H_n(x,y)$ is the length of a prefix code word specifying the positions where x and y differ. This modified Hamming distance is symmetric, and it is an admissible distance by the Kraft inequality, $\sum_{y:y\neq x} 2^{-H_n(x,y)} \leq 1$. It is easy to verify that H_n is a metric in the sense that it satisfies the metric (in)equalities up to $O(\log n)$ additive precision. ◇

The following theorem shows that E_1 is an optimal admissible information distance. It is remarkable that this distance happens also to have a more or less physical interpretation as the approximate length of the conversion program of Theorem 8.3.1 and, as shown in the next section, of the smallest program that transforms x into y on a reversible machine.

Theorem 8.3.2 *The distance $E_1(x,y) = \max\{K(x|y), K(y|x)\}$ is admissible, it is a metric, and it is minimal in the sense that for every admissible distance function $D(x,y)$, we have $E_1(x,y) \leq D(x,y)$, every (in)equality up to an additive constant term.*

Proof. The nonnegativity and symmetry properties are immediate from the definition. Since $K(x|z) \leq K(x|y) + K(y|z) + O(1)$, the triangle inequality is immediate as well. It is straightforward that $E_1(x,y)$ is upper semicomputable.

We verify the density requirement of Equation 8.11. For fixed x, the function $f_x(y)$ is given by

$$f_x(y) = 2^{E_1(x,y)} = 2^{-\max\{K(x|y),K(y|x)\}},$$

and hence $f_x(y) \leq 2^{-K(y|x)}$. The right-hand side $2^{-K(y|x)} \leq \sum 2^{-l(p)}$, where the sum is taken over all programs p for which the reference prefix machine U, given x, computes y. This sum is the probability that U, given x, computes y from a program p generated bit by bit uniformly at random. Hence, $\sum_y f_x(y) \leq 1$. Defining $f_y(x)$ similarly, the same argument shows that $f_y(x) \leq 1$. Therefore, $E_1(x,y)$ satisfies the density requirement of Equation 8.11.

We prove minimality. Let x be fixed. If $\sum_{y:y\neq x} 2^{-D(x,y)} \leq 1$, then by Theorem 4.3.1 there is a constant c, independent of y, such that

$c\mathbf{m}(y|x) \geq 2^{-D(x,y)}$. Theorem 4.3.4 shows that $\log 1/\mathbf{m}(y|x) = K(y|x) + O(1)$. Hence, $D(x,y) \geq K(x|y) + \log 1/c + O(1)$. Repeat this argument with x and y interchanged. Altogether, there is a constant c' such that $D(x,y) \geq \max\{K(x|y), K(y|x)\} - c'$. □

8.3.4 Reversible Distance

Another information distance is based on the idea that one should aim for dissipationless computations, and hence for reversible ones as in Section 8.2. Such an information distance is given by the length of the shortest reversible program that transforms x into y and y into x on a universal reversible computer. The reversible prefix complexity KR was defined in Definition 8.2.2 on page 640. The reversible information distance turns out to be $E_2(x,y) = KR(x|y) = KR(y|x)$. It will be shown also that $E_2 = E_1$, up to a logarithmic additive term. Previously, we have determined that $E_1 = E_0$, up to a logarthmic additive term. It is remarkable that three so differently motivated definitions turn out to define one and the same notion.

In Theorem 8.3.1 it was demonstrated that for all strings x and y, there exists a conversion program p, of length at most logarithmically greater than $E_1(x,y) = \max\{K(y|x), K(x|y)\}$, such that $U(p,x) = y$ and $U(p,y) = x$. We show that the length of this minimal conversion program is equal within a constant to the length of the minimal *reversible* program for transforming x into y.

Theorem 8.3.3 *Up to an additive constant,*

$$KR(y|x) = \min\{l(p) : U(p,x) = y \text{ and } U(p,y) = x\}\,.$$

Proof. This proof is an example of the general technique for combining two irreversible programs, for y from x and for x from y, into a single reversible program for y from x. In this case the two irreversible programs are almost the same, since by Theorem 8.3.1 the minimal conversion program p is both a program for y given x and a program for x given y.

The computation proceeds by several stages, as shown in the table in Figure 8.7. To illustrate motions of the head on the self-delimiting program tape, the program p is represented by the string 'prog' in the table, with the head position indicated by a caret. Each of the stages can be accomplished without using any many-to-one operations.

In stage 1, the computation of y from x, which might otherwise involve irreversible steps, is rendered reversible by saving a history, on previously blank tape, of all the information that would have been thrown away.

In stage 2, making an extra copy of the output onto blank tape is an intrinsically reversible process, and therefore can be done without writing anything further in the history. Stage 3 exactly undoes the work of stage 1, which is possible because of the history generated in stage 1.

Stage and Action	Program		Work Tape	
0. Initial configuration	$\hat{\text{p}}\text{rog}$	x		
1. Compute y, saving history	$\text{pro}\hat{\text{g}}$	y	$(y\|x)$-history	
2. Copy y to blank region	$\text{pro}\hat{\text{g}}$	y	$(y\|x)$-history	y
3. Undo comp. of y from x	$\hat{\text{p}}\text{rog}$	x		y
4. Swap x and y	$\hat{\text{p}}\text{rog}$	y		x
5. Compute x, saving history	$\text{pro}\hat{\text{g}}$	x	$(x\|y)$-history	x
6. Cancel extra x	$\text{pro}\hat{\text{g}}$	x	$(x\|y)$-history	
7. Undo comp. of x from y	$\hat{\text{p}}\text{rog}$	y		

FIGURE 8.7. Combining irreversible computations of y from x and x from y to achieve a reversible computation of y from x

Perhaps the most critical stage is stage 5, in which x is computed from y for the sole purpose of generating a history of that computation. Then, after the extra copy of x is reversibly disposed of in stage 6 by cancellation (the inverse of copying onto blank tape), stage 7 undoes stage 5, thereby disposing of the history and the remaining copy of x, while producing only the desired output y.

Not only are all its operations reversible, but the computations from x to y in stage 1 and from y to x in stage 5 take place in such a manner as to satisfy the requirements for a reversible prefix interpreter. Hence, the minimal irreversible conversion program p, with constant modification, can be used as a reversible program for UR to compute y from x.

Conversely, the minimal reversible program for y from x, with constant modification, serves as a program for y from x for the ordinary irreversible prefix machine U, because reversible prefix machines are a subset of ordinary prefix machines. This establishes the theorem. □

Definition 8.3.4 The *reversible distance* $E_2(x, y)$ between x and y is defined by

$$E_2(x, y) = KR(y|x) = \min\{l(p) : UR(p, x) = y\} .$$

As just proved, this is within an additive constant of the size of the minimal conversion program of Theorem 8.3.1. Although it may be logarithmically greater than the optimal distance E_1, it has the intuitive advantage of being the actual length of a concrete program for passing in either direction between x and y. The optimal distance E_1, on the other hand, is defined only as the greater of two one-way program sizes.

Theorem 8.3.4 *The reversible distance E_2 is a metric up to additive constants in the metric (in)equalities.*

Proof. Clearly, E_2 satisfies the identity axiom and is trivially symmetric. It remains to prove that it obeys the triangle inequality.

Claim 8.3.1 $E_2(x,z) < E_2(x,y) + E_2(y,z) + O(1)$.

Proof. We will show that given reversible *UR* programs p and q for computing $(y|x)$ and $(z|y)$, respectively, a program of the form spq, where s is a constant supervisory routine, serves to compute z from x reversibly. Because the programs are self-delimiting, no punctuation is needed between them. If this were an ordinary irreversible U computation, the concatenated program spq could be executed in an entirely straightforward manner, first using p to go from x to y, then using q to go from y to z.

However, with reversible *UR* programs, after executing p, the head will be located at the beginning of the program tape, and so will not be ready to begin reading q. It is therefore necessary to remember the length of the first program segment p temporarily, to enable the program head to space forward to the beginning of q, but then cancel this information reversibly when it is no longer needed.

A scheme for doing this is shown in the table in Figure 8.8, where the program tape's head position is indicated by a caret. To emphasize that the programs p and q are strings concatenated without any punctuation between them, they are represented respectively in the table by the expressions 'pprog' and 'qprog,' and their concatenation pq by 'pprogqprog.' □ □

8.3.5 Sum Distance

Only the irreversible erasures of a computation need to dissipate energy. This raises the question of the minimal amount of irreversibility required in transforming string x into string y, that is, the number of bits we have to add to x at the beginning of a reversible computation from x to y,

Stage and Action	Program	Work Tape	
0. Initial configuration	p̂progqprog	x	
1. Compute $(y\|x)$, transcribing pprog.	p̂progqprog	y	pprog
2. Space forward to start of qprog.	pprogq̂prog	y	pprog
3. Compute $(z\|y)$.	pprogq̂prog	z	pprog
4. Cancel extra pprog as head returns.	p̂progqprog	z	

FIGURE 8.8. Reversible execution of concatenated programs for $(y|x)$ and $(z|y)$ to transform x into z

and the number of garbage bits left (apart from y) at the end of the computation that must be irreversibly erased to obtain a clean y.

Even though consuming and producing information may seem to be operations of opposite sign, we can define a distance $E_3(\cdot,\cdot)$ based on the notion of information flow as the minimal *sum* of amounts of extra information flowing into and out of the computer in the course of the computation transforming x into y. This may be viewed as a measure of the work required during a reversible computation in which the program is not retained.

The resulting distance turns out to be within a logarithmic additive term of the sum of the conditional complexities $K(y|x) + K(x|y)$. The reversible distance E_2 defined in the previous section is equal to the length of a catalytic program, which allows the interconversion of x and y while remaining unchanged itself. Here we consider noncatalytic reversible computations that consume some information p besides x and produce some information q besides y.

We start with the enumeration of all reversible Turing machines. We can take either the prefix machines or the nonprefix ones. For the validity of the properties below it doesn't matter whether we use prefix machines. Using nonprefix machines gives a smaller sum distance variant, though. We therefore use the nonprefix machines. For a function ψ computed on a reversible Turing machine, let

$$E_\psi(x,y) = \min\{l(p) + l(q) : \psi(\langle x,p\rangle) = \langle y,q\rangle\}\,.$$

Lemma 8.3.2 *There is a universal (nonprefix) reversible Turing machine U computing the function ψ_0 such that for all functions ψ computed on a reversible Turing machine, we have*

$$E_{\psi_0}(x,y) \le E_\psi(x,y) + c_\psi$$

for all x and y, where c_ψ is a constant that depends on ψ but not on x or y.

Proof. Use the universal reversible Turing machines in the manner of the proof of Theorem 2.1.1 on page 105. □

Definition 8.3.5 The *sum distance* E_3 is defined by $E_3(x,y) = E_{\psi_0}(x,y)$.

Theorem 8.3.5 $E_3(x,y) = K(x|y) + K(y|x) \pm O(\log(K(x|y) + K(y|x)))\,.$

Proof. ($\ge$) We show that $E_3(x,y) \ge C(y|x) + C(x|y)$. Since $C(x|y) \ge K(x|y) - 2\log C(x|y) - O(1)$ for all x and y (Example 3.1.4 on page 203),

this will prove the lower bound. To compute y from x we must be given a program p to do so with which to start. By definition,

$$C(y|x) \leq l(p) + O(1).$$

Assume that the computation from x, p ends up with y, q. Since the computation is reversible, we can compute x from y, q. Consequently, $C(x|y) \leq l(q) + O(1)$.

($\leq$) Assume $k_1 = K(x|y) \leq k_2 = K(y|x)$ and let $l = k_2 - k_1$. According to Theorem 8.3.1, there is a string r of length $l + O(\log l)$ such that $K(rx|y) = k_1 + O(\log k_1)$ and $K(y|rx) = k_1 + O(\log k_1)$. We can even assume r to be self-delimiting: the cost can be included in the $O(\log l)$ term. According to Theorem 8.3.1 and Theorem 8.3.3, there is a program q of length $k_1 + O(\log k_1)$ going reversibly between rx and y. Therefore, with a constant extra program s, the universal reversible machine will go from (rq, x) to (q, y). And by the above estimates,

$$l(rq) + l(q) \leq 2k_1 + l + O(\log k_2) = k_1 + k_2 + O(\log k_2)\,.$$

□

Note that all bits supplied in the beginning to the computation, apart from input x, as well as all bits erased at the end of the computation, are *random* bits. This is because we supply and delete only shortest programs, and a shortest program p satisfies $K(p) \geq l(p)$, that is, it is maximally random.

Example 8.3.4 It does not matter whether the irreversible providing and erasing operation executions are scattered throughout the computation, or whether all required bits are irreversibly provided at the beginning and all garbage bits are irreversibly erased at the end.

Namely, we can provide the sequence of bits that are irreversibly consumed in the computation as a specially delimited sequence at the beginning of the computation. Whenever the computation requires an irreversibly provided bit, we use the next unused bit from this special string.

Bits that need to be irreversibly erased are put in a special delimited string in the order in which they need to be erased. This special string is then erased at the very end of the computation. ◇

Theorem 8.3.6 *The distance $E_4(x, y) = K(x|y) + K(y|x)$ is admissible, it is a metric, and it is minimal in the sense that for every admissible distance function $D(x, y)$, we have $E_4(x, y) \leq 2D(x, y)$, all (in)equalities up to an additive constant term.*

Proof. Similar to that of Theorem 8.3.2. □

8.3.6 Metrics Relations

The metrics we have considered can be arranged in increasing order. Within an additive logarithmic term, we have

$$E_0(x,y) = E_1(x,y) = E_2(x,y) \leq E_3(x,y) = E_4(x,y) \leq 2E_1(x,y),$$

where the various items were defined or proven to be precisely

$$\begin{aligned}
E_0(x,y) &= \max\{K(y|x), K(x|y)\} + O(\log \max\{K(y|x), K(x|y)\}),\\
E_1(x,y) &= \max\{K(y|x), K(x|y)\},\\
E_2(x,y) &= KR(y|x) = \min\{l(p) : U(p,x) = y,\ U(p,y) = x\} + O(1),\\
E_3(x,y) &= K(x|y) + K(y|x) \pm O(\log(K(x|y) + K(y|x))),\\
E_4(x,y) &= K(x|y) + K(y|x).
\end{aligned}$$

The sum distance E_3, in other words, can be anywhere between the optimum distance E_1 and twice the optimal distance. The former occurs if one of the conditional entropies $K(y|x)$ and $K(x|y)$ is zero, the latter if the two conditional entropies are equal.

8.3.7 Minimal Overlap

Let p be a shortest program converting y to x, and let q be a shortest program converting x to y. Naively we expect that the shortest program that that maps y to x contains the information about x that is lacking in y. However, this is too simple, because different short programs mapping y to x may have different properties.

Example 8.3.5 Let x and y be strings of length n with $C(x|y), C(y|x) = n$. Let p_1 be a program that ignores the input and prints x. Let p_2 be a program such that $y \oplus p = x$ (that is, $p = x \oplus y$), where $\oplus$ denotes bitwise addition modulo 2. The programs p_1 and p_2 have nothing in common. ◇

Example 8.3.6 The fundamental Theorem 8.3.1 stated that p and q can be made dependent on each other as much as possible. We express this in terms of the mutual information, as given in Equation 3.15 on page 248, $I(x:y) = K(x) - K(x|y)$. There is also $I(x;y) = K(x) + K(y) - K(x,y)$ of Equation 3.20 on page 252. Since p and q are both shortest programs, that is, $p = p^*$ and $q = q^*$, we have $I(p:q) = I(q:p) + O(1)$ by Theorem 3.9.2, and in fact $I(p:q) = I(p;q) + O(1)$. The mutual information in p and q is maximal: $I(p:q) = \min\{l(p), l(q)\}$ up to an additive $O(\log(K(x|y) + K(y|x)))$ term. The opposite question is whether p and q can always be made completely independent. That is, we wonder whether we can choose p and q such that $I(p:q) = 0$, up to some small additive term.

It turns out that the answer to this question depends on the size of the additive term. If the additive term is $O(\log(K(x|y)+K(y|x)))$, then the answer is negative; and if the additive term is $O(\log K(x,y))$, then the answer is positive as a consequence of Theorem 8.3.1 and Exercise 8.3.9 on page 659. The latter result also follows from a deeper analysis below, with more profound and far-reaching consequences. Below, U is the reference universal prefix machine used in Theorem 3.1.1 on page 202.

Claim 8.3.2 Let x and y be strings of length at most n. Assume that $U(p,y)=x$ and $U(q,x)=y$, so that $K(x|y)=K(p)$ and $K(y|x)=K(q)$. If $K(p|x)=0$ and $K(q|y)=0$, then $I(p:q)=0$, with the last three equalities holding up to an additive $O(\log n)$ term.

Proof. Below all (in)equalities hold up to the same additive term. We need to prove only $K(p|q)=K(p)$. By definition, $K(p|q)\leq K(p)$. If $K(p|q)<K(p)$, then by the assumption in the claim we can use y to generate q, and then compute p from q using fewer than $K(p)$ bits, and finally x from y and p. Then, $K(x|y)<K(p)$, contradiction. □

This leads to a fundamental question: is there a shortest program p that computes x from y while p is simple with respect to x? (Simplicity with respect to x implies little dependence on y.) By Exercise 8.3.9, Item (c), we know that $I(p:q)=0$ does not hold, up to an $O(\log(K(x|y)+K(y|x)))$ additive term, and hence by the claim neither do $K(p|x)=0$ and $K(q|y)=0$. But the next theorem shows that both do hold to within an $O(\log K(x,y))$ additive term. ◇

Muchnik's theorem below shows that there exists a shortest program p that converts y to x ($l(p)=K(x|y)$) such that p is simple with respect to x and therefore depends little on the origin y but for possibly an $O(\log K(x,y))$ amount. This is a fundamental coding property for individual strings that parallels the result in information theory about random variables known as the Slepian–Wolf and Körner–Csiszár–Marton theorems. It is convenient to consider this question using plain Kolmogorov complexity, although the results in this section hold for both C and K complexities.

Theorem 8.3.7 *Let x,y be binary strings of length at most n. Then there exists a string p of length $C(x|y)$ such that $C(p|x)=O(\log n)$ and $C(x|p,y)=O(\log n)$.*

Proof. Let m and n be positive integers with $m\leq n$. Let $\mathcal{X}$ be a set of cardinality $2^{n+1}-1$ (the set of strings of length at most n), and let P be a set of cardinality 2^m (the set of all strings of length m). The strategy is to define a collection of N hash functions from $\mathcal{X}$ to P such that the hash value of one of the functions can be used as a short program for x.

Claim 8.3.3 For all m, n, N there exists a family of N hash functions $\eta_i : \mathcal{X} \to P$ $(i = 1, \ldots, N)$ satisfying:

(i) $C(\eta_1, \ldots, \eta_N) = C(m, n, N) + O(1)$; and

(ii) Every subset $U \subset \mathcal{X}$ of cardinality $d(U) < 2^{m-(n+m+2)/N}$ is hashed to a subset $B \subseteq P$ defined by $B = \{\eta_i(u) : u \in U,\ i \in \{1, \ldots, N\}\}$ such that $d(B) \geq d(U)$.

Proof. (i) We need to show only that a hash function family of high Kolmogorov complexity has the expanding property of Item (ii). This shows that such a family exists. Then, the first such family (possibly not the one used to prove existence) that we find by exhaustive search has Kolmogorov complexity $C(m, n, N) + O(1)$, yielding Item (i).

(ii) Fix a string r of length $l(r) = N \times d(\mathcal{X}) \times m$ satisfying

$$C(r|n, m, N, A) \geq l(r) = Nd(\mathcal{X})m,$$

where A is an algorithm that reconstructs r from the description below. This string r is used to construct a family of N hash functions of high complexity. Partition r into N equal blocks. The ith block r_i is further partitioned into $d(\mathcal{X})$ blocks $r_{i1}, \ldots, r_{id(\mathcal{X})}$, each of length m. Then, the ith hash function η_i is defined by

$$\eta_i(x_j) = r_{ij},$$

for $j = 1, \ldots, d(\mathcal{X})$. Define $\eta(U) = \{\eta_i(u) : u \in U,\ i \in \{1, \ldots, N\}\}$. We can give r by the following description:

- Self-delimiting descriptions of $d(U)$ and $d(\eta(U))$ of length at most

  $$\alpha_1 = \log d(U)d(\eta(U)) + 2\log(\log d(U) \log d(\eta(U))) + 2.$$

- An encoding of the arguments and function values between U and $\eta(U)$, in at most

  $$\alpha_2 = d(U)(n + 1) + md(\eta(U)) + Nd(U) \log d(\eta(U))$$

  bits. The $d(U)(n + 1)$ term specifies the indices of arguments in $U \subseteq \mathcal{X}$, the second term specifies the values of $\eta(U)$ in the set P, and the third term specifies the mapping of the N hash functions η_i in terms of mapping U to the restricted range $\eta(U) \subseteq P$.

- Trivially, the remainder of r requires at most bit-length

  $$\alpha_3 = N(d(\mathcal{X}) - d(U))m.$$

The length of this description cannot be shorter than $C(r|n, m, N, A)$ bits. Therefore,

$$\alpha_1 + \alpha_2 + \alpha_3 \geq N d(\mathcal{X}) m.$$

Assume, by way of contradiction, $d(\eta(U)) < d(U)$. Then, the last displayed inequality yields $d(U) \geq 2^{m-(n+m+1+\epsilon)/N}$, with $\epsilon \leq (2 \log d(U) + 4 \log \log d(U) + 2)/d(U) < 1$, which is a contradiction. (If $d(U)$ is too small to make $\epsilon < 1$ then we can use another code to make $\epsilon < 1$, which is left as an exercise for the reader.) Hence, we can set $B = \eta(U)$. □

The family of hash functions can be represented by a bipartite graph that has strings from $\mathcal{X}$ on the left and strings from P on the right. Every string $x \in \mathcal{X}$ is connected by edges with N strings $\eta_1(x), \ldots \eta_N(x)$ on the right, the hash values. If values coincide then there are several edges connecting two vertices. Let $\mathcal{X}_y = \{z : C(z|y) \leq C(x|y)\}$. Clearly, we have $x \in \mathcal{X}_y$. Furthermore, $d(\mathcal{X}_y) < \min\{2^{C(x|y)+1}, d(\mathcal{X})\}$, the first term since the number of different programs of length up to $C(x|y)$ cannot be larger, and the second term since $\mathcal{X}_y \subseteq \mathcal{X}$. We are interested in the restriction of the graph to $\mathcal{X}_y \subseteq \mathcal{X}$; all vertices in $\mathcal{X} - \mathcal{X}_y$ and their incident edges can be removed.

Claim 8.3.4 Set $N = n + m + 2$ and $m = C(x|y) + 3$. There is a constant c and an i such that some hash value $\eta_i(x)$ has no more than n^c neighbors in $\mathcal{X}_y$ (arguments that are mapped to $\eta_i(x)$ by some hash function η_j for $1 \leq j \leq N$).

Proof. Let B be the set of hash values in P that do not satisfy the claim. These are the *bad* elements. Then,

$$d(B) \leq d(\mathcal{X}_y) N / n^c < 2^{m-2}(n + m + 2)/n^c.$$

For $n \geq 2$ we have $d(B) < 2^m / n^{c-1}$. Then, we can apply Claim 8.3.3 with $N = n + m + 2$, so that the claim holds for every U with $d(U) < 2^{m-1}$. Every $U_0 \subseteq \mathcal{X}_y$ has fewer than 2^{m-1} elements by choice of m and so is eligible as a set U. The claim guarantees that every U_0 mapping to the set of hash values B under any hash function in the family satisfies $d(U_0) \leq d(B)$. Then, every element $z \in U_0$ has complexity

$$C(z|y) \leq O(\log n) + \log d(B) \leq m - (c-1) \log n + O(\log n).$$

Choosing c large enough proves that $x \notin U_0$, since $C(x|y) = m - 3$. □

By Claim 8.3.4, the string x can be described by η_i, y, and the index of x among $\eta_i(x)$'s neighbors. By Claims 8.3.3 and 8.3.4, setting $p = \eta_i(x)$, we have $C(x|y, p) = O(\log n)$. The fact that $C(p|x) = O(\log n)$ follows by Claim 8.3.3, Item (i), and $p = \eta_i(x)$ for some $i \leq N$, since we have set $N = n + m + 2$. □

The theorem gives a code p for x when y is known. We have assumed that x and y have length at most n. But the proof does not use any assumption about y. The code p is not uniquely determined. If y and z satisfy $C(y|z) = C(z|y) = l(y) = l(z)$ and $x = yz$, then both z and $y \oplus z$ can be used for p and they have no mutual information at all.

Corollary 8.3.3 For all strings x and y, there are programs p and q such that $U(q, x) = y$ and $U(p, y) = x$, where $l(q) = C(y|x)$, $l(p) = C(x|y)$, and $C(p) - C(p|q) = C(q) - C(q|p) = 0$, and the last four equalities hold up to an additive $O(\log C(x, y))$ term.

Exercises

8.3.1. [29] For a binary string x, let $B_1(x, d)$ be the set of strings y with $E_1(x, y) \leq d$. Let the number of elements in $B_1(x, d)$ be denoted by $b_1(x, d)$.

(a) Show that up to an additive constant, $d - K(d) < \log b_1(x, d) < d - K(d|x)$, and the same bounds apply to the cardinality of $B_1(x, d) \bigcap \{y : l(y) = l(x)\}$.

(b) Let x be a binary string. Show that there is a positive constant c such that for all sufficiently large d, the number of binary strings y with $E_3(x, y) \leq d$ is at most $2^d/c$ and at least $2^d/(d \log^2 d)$.

(c) For the number of strings of length n close to a random string x of length n (that is, $K(x) \geq n$) in the distance E_3, the picture is different from that for distance E_1 in Item (a). In distance E_3, 'tough guys have few neighbors.' Show that a random string x of length n has only about $2^{d/2}$ strings of length n within distance d.

(d) Generalize Item (c): Let the binary strings x and y have length n. Show that for every x, the number of y's such that $E_3(x, y) \leq d$ is 2^α, with $\alpha = \frac{1}{2}(n + d - K(x)) \pm O(\log n)$ for $n - K(x) \leq d$. and $\alpha = d \pm O(\log n)$ for $n - K(x) > d$.

(e) Let c be a constant and let S be a set with cardinality $d(S) = 2^d$ and $K(S) = c \log d$. Show that almost all pairs of elements $x, y \in S$ have distance $E_1(x, y) \geq d$, up to an additive logarithmic term. Show that a similar statement can be proved for the distance of a string x (possibly outside S) to the majority of elements y in S. If $K(x) \geq n$, then for almost all $y \in S$ we have $E_1(x, y) \geq n + d - O(\log dn)$.

Comments. In a discrete space with a distance function, the rate of growth of the number of elements in balls of size d can be considered as a kind of dimension of the space. Hint for Item (a): the upper bound is immediate from Exercise 3.3.9 on page 214. Hint for Item (b): the upper bound follows from Item (a) since $E_3 \geq E_1$; for the lower bound,

consider strings y of the form px, where p is a self-delimiting program. Source: C.H. Bennett, P. Gács, M. Li, P.M.B. Vitányi, and W.H. Zurek, *IEEE Trans. Inform. Theory*, 44:4(1998), 1407–1423.

8.3.2. [21] Fix a reference universal reversible Turing machine, say UR_0. Define $E_3^t(x,y) = \min\{l(p) + l(q) : UR_0(\langle p, x\rangle) = \langle q, y\rangle \text{ in } t(n) \text{ steps of computation, where } n = l(x)\}$. Show that for every recursive function t there is an x such that $E_3^t(x,\epsilon) > E_3(x,\epsilon)$.

8.3.3. [32] Prove the upper bound on E_3 of Theorem 8.3.5 on page 651 by a direct construction supplying logarithmically small initial programs and ending with logarithmically small garbage.

Comments. Source: M. Li and P.M.B. Vitányi, *Proc. Royal Soc. London, Ser. A*, 452(1996), 769–789. This is also the source of the four exercises that follow.

8.3.4. [25] We determine the irreversibility cost of effective erasure. Let x be a string and $t(n) \geq n$ $(n = l(x))$ be a time bound that is provided at the start of the computation. Show that erasing the n-bit record x by an otherwise reversible computation can be done in time (number of steps) $O(2^n t(n))$ at irreversibility cost $C^t(x) + 2C^t(t|x) + 4\log C^t(t|x)$ bits. (Typically we consider t as some standard explicit time bound and the last two terms adding up to $O(1)$.)

8.3.5. [37] Use the notions of Exercise 8.3.2. Prove a time–energy tradeoff hierarchy: For every large enough n there is a string x of length n and a sequence of $m = \frac{1}{2}\sqrt{n}$ time functions $t_1(n) < t_2(n) < \cdots < t_m(n)$ such that

$$E_3^{t_1}(x,\epsilon) > E_3^{t_2}(x,\epsilon) > \cdots > E_3^{t_m}(x,\epsilon).$$

Improve this result by replacing ϵ by an arbitrary string y.

8.3.6. [28] Show that there is a recursively enumerable infinite sequence χ and some (nonrecursively) large time bound T such that for every total recursive time bound t, for each initial segment x of χ,

$$E_3^t(x,\epsilon) > c_t 2^{E_3^T(x,\epsilon)/2},$$

where $c_t > 0$ is a constant depending only on t and χ.

8.3.7. [23] Define a uniform variant E_u of E_3 based on the uniform Kolmogorov complexity variant of Exercise 2.3.2 on page 130. Show that the energy dissipation can be reduced arbitrarily far by a computation that uses enough (that is, an incomputable amount of) time. That is, there is an infinite sequence ω and a (nonrecursively) large time bound T such that for every unbounded total recursive function f, no matter

how large, for every total recursive time bound t, there are infinitely many n for which

$$E_u^t(\omega_1 \ldots \omega_n, \epsilon) > f(E_u^T(\omega_1 \ldots \omega_n, \epsilon)).$$

Comments. Hint: use Exercise 2.5.13 on page 161 and Exercise 7.1.7 on page 546.

8.3.8. [O40] Develop a theory of resource-bounded (polynomial-time or logarithmic-space) information distance.

8.3.9. [39] (a) Show that for strings x and y that are random with respect to one another in the sense that $K(x|y) \geq l(x)$ and $K(y|x) \geq l(y)$, there are p and q such that $K(p) = K(y|x)$, $K(q) = K(x|y)$, $I(p;q) = 0$, $U(p,x) = y$, and $U(q,y) = x$, where the first three equalities hold up to an additive $O(K(x|y) + K(y|x))$ term.

(b) Give a proof of Corollary 8.3.3 using the conversion theorem, Theorem 8.3.1.

(c) Show that Muchnik's theorem, Theorem 8.3.7, does not hold if we replace the additive term $O(\log C(xy))$ by $O(\log(C(x|y) + C(y|x)))$.

Comments. Source: Item (a): C.H. Bennett, P. Gács, M. Li, P.M.B. Vitányi, and W.H. Zurek, *Ibid.*, attributed to N.K. Vereshchagin; Items (b) and (c): N.K. Vereshchagin and M.V. Vyugin, *Theoret. Comput. Sci.*, 271(2002), 131–143.

8.3.10. [36] Use the techniques in Muchnik's theorem, Theorem 8.3.7, to prove the following.

(a) For all strings x, y, and z of length less than n and $C(x|y) = C(x|z) = m$, there exists a string p of length m such that $C(p|x) = C(x|p,y) = C(x|p,z) = O(\log n)$. That is, p is a program for x when y or z is known, and p does not contain extra information beyond x.

(b) Let x, y, z be strings of length less than n. Then there exist strings p and q such that one of them is a prefix of the other, $l(p) = C(x|y)$, $l(q) = C(x|z)$, and $C(x|p,y) = C(x|q,y) = C(p|x) = C(q|x) = O(\log n)$.

Comments. Source: An.A. Muchnik, *Theoret. Comput. Sci.*, 271(2002), 97–109.

8.3.11. [35] It is easy to see that for every string x and every integer n, there is a y such that the information distance $\max\{C(x|y), C(y|x)\}$ between x and y is $n + O(1)$. Prove that for every n and for every string x such that $C(x) \geq 2n + O(1)$ there exists a string y such that both $C(x|y)$ and $C(y|x)$ are equal to $n + O(1)$.

Comments. If the additive term $O(1)$ were replaced by $O(\log n)$, then this exercise becomes straightforward. Source: M.V. Vyugin, *Theoret. Comput. Sci.*, 271(2002), 145–150.

8.3.12. [29] (a) Prove that for every natural number n there exist infinitely many strings x (which may be very long compared to n) such that $C(x) > 2n$ and for every string y with $C(y|x) < n$ we have either $C(y) < n + O(1)$ or $C(x|y) < n + O(1)$.

(b) Show that for the x's of Item (a) there is no y such that $C(y|x) \approx \frac{1}{2}n$ and $C(x|y) \approx 2n$.

Comments. The string x is an indivisible piece of information; a separable part of it is either trivial or contains all of x. Source: M.V. Vyugin, *Ibid.*, attributed to An.A. Muchnik.

8.3.13. [38] Show that there are strings a, b, and p of lengths n, $2n$, and $k \geq 2n$, respectively, having the following properties: (i) $C(b|a,p) = 0$; (ii) $C(p|a) \geq k$; (iii) there is no string q such that $C(q) \leq k - n$, $C(q|p) \leq n$, and $C(b|a,q) \leq n$, all (in)equalities holding to within an additive term $O(\log(k+n))$.

Comments. Muchnik's theorem, Theorem 8.3.7, can be interpreted as follows. Given strings x and y, is it possible to find a simplification p of x (which may be regarded as a description of itself) such that $C(x|y,p) = 0$, $l(p) = C(x|y)$, and $C(p|x) = 0$, with equality up to an additive $O(\log C(xy))$ term? In general, if we are given a description p of $x|y$, a simplification of p might not exist even if p is much larger than $C(x|y)$. The exercise formalizes this question. Source: An.A. Muchnik, A.K. Shen, M.A. Ustinov, N.K. Vereshchagin, and M.V. Vyugin, *Proc. Theory Applications Models Comput.*, Lect. Notes Comp. Sci., Vol. 3959, Springer-Verlag, Berlin, 308–317. There it is also shown that the above holds for all a, b, and k except for some trivial cases such as $C(a) = 0$.

8.4 Normalized Information Distance

We continue Section 8.3, in particular Section 8.3.3. The quantitative difference in a certain feature between two objects can be considered as an admissible distance, Definition 8.3.3 on page 646. Theorem 8.3.2 on page 647 shows that the information distance E_1 is universal in that among all admissible distances it is always least. That is, it accounts for the dominant feature in which two objects are alike. Many admissible distances are absolute, but if we want to express similarity, then we are more interested in relative ones. For example, if two strings of 1,000,000 bits differ by 1,000 bits, then we are inclined to think that those strings are relatively similar. But if two strings of 1,000 bits differ by 1,000 bits, then we find them very different.

Example 8.4.1 Consider the problem of comparing genomes. The *E. coli* genome is about 4.8 megabases long, whereas *H. influenza*, a sister species of *E. coli*, has genome length only 1.8 megabases. The information distance E_1 between the two genomes is dominated by their length difference rather than the amount of information they share. Such a measure will trivially classify *H. influenza* as being closer to a more remote species of similar genome length such as *A. fulgidus* (2.18 megabases), rather than with *E. coli*. To deal with such problems, we need to normalize. ◇

Our objective is to normalize the universal information distance $E_1 = \max\{K(x|y), K(y|x)\}$ to obtain a universal similarity distance. It should give a similarity with distance 0 when objects are maximally similar and distance 1 when they are maximally dissimilar.

8.4.1 The Similarity Metric

It is paramount that the normalized version of the universal information distance metric is also a metric. Were it not, then the relative relations between the objects in the space would be disrupted and this could lead to anomalies, for instance, if the triangle inequality were to be violated for the normalized version. (In certain semantic applications in Section 8.4.2 we will relax this rule.)

In order to obtain a normalized universal information distance function, various versions of information distance occurring in Sections 8.3.3, 8.3.5, and 8.3.4 can be normalized. We will discuss only how to normalize the max distance E_1 of Definition 8.3.2 on page 642 and call it the *normalized information distance* or *the similarity metric*. See Exercise 8.4.5 on page 672 for the normalized sum distance.

Definition 8.4.1 The *normalized information distance* (NID) between two binary sequences x and y is defined as

$$e(x,y) = \frac{\max\{K(x|y), K(y|x)\}}{\max\{K(x), K(y)\}}. \tag{8.12}$$

Example 8.4.2 Several natural alternatives for the denominator turn out to be wrong.

(i) Divide by the length. Then, firstly we do not know which of the two lengths involved is the one to divide by, possibly the sum or maximum, but secondly the triangle inequality is not satisfied.

(ii) Divide by $K(x,y)$. Then one has $e(x,y) = \frac{1}{2}$ when x and y satisfy $K(x) \approx K(y) \approx K(x|y) \approx K(y|x)$. This is improper, since when x and y are completely dissimilar they should have $e(x,y) = 1$. ◇

Example 8.4.3 There is a natural interpretation of $e(x,y)$. Let $I(x:y) = K(y) - K(y|x)$ be the mutual information version of Equation 3.15 on page 248. Without

loss of generality let $K(y) \geq K(x)$. Then

$$e(x,y) = \frac{K(y) - I(x:y)}{K(y)} = 1 - \frac{I(x:y)}{K(y)}.$$

That is, $1 - e(x,y)$ is the number of bits of information that are shared between the two strings x and y per bit of information of the string with the most information. ◇

Theorem 8.4.1 *The normalized information distance $e(x,y)$ takes values in the range $[0,1]$ and it is a metric, up to negligible errors.*

Proof. Clearly, $e(x,y)$ takes values in the range $[0,1]$. It is straightforward that the normalized information distance $e(x,y)$ is symmetric: $e(x,y) = e(y,x)$. Identity $e(x,x) = 0$ holds up to an $O(1/K(x))$ additive term. Also, $e(x,y) \neq 0$ for $y \neq x$. To show that $e(x,y)$ is a metric, it remains to prove the triangle inequality.

Claim 8.4.1 $e(x,y) \leq e(x,z) + e(z,y)$ up to an additive term $O((\log K)/K)$, where $K = \max\{K(x), K(y), K(z)\}$.

Proof. **Case 1.** Suppose $K(z) \leq \max\{K(x), K(y)\}$. For all x, y, z, we have $K(x|y) \leq K(x|z) + K(z|y) + O(1)$. Therefore,

$$\max\{K(x|y), K(y|x)\} \leq \max\{K(x|z) + K(z|y), K(y|z) + K(z|x)\},$$

and consequently

$$\begin{aligned}\frac{\max\{K(x|y), K(y|x)\}}{\max\{K(x), K(y)\}} &\leq \frac{\max\{K(x|z) + K(z|y), K(y|z) + K(z|x)\}}{\max\{K(x), K(y)\}} \\ &\leq \frac{\max\{K(x|z), K(z|x)\}}{\max\{K(x), K(y)\}} + \frac{\max\{K(z|y), K(y|z)\}}{\max\{K(x), K(y)\}},\end{aligned}$$

with (in)equalities up to an additive $O(1/K)$ term. Replacing $K(y)$ by $K(z)$ in the denominator of the first term in the right-hand side, and $K(x)$ by $K(z)$ in the denominator of the second term of the right-hand side, respectively, can only increase the right-hand side, by assumption.

Case 2. Suppose $K(z) = \max\{K(x), K(y), K(z)\}$. Further assume that $K(x) \geq K(y)$ (the remaining case is symmetrical). Then, using the symmetry of information theorem, Theorem 3.9.2, to determine the maxima, we also find that $K(z|x) \geq K(x|z)$ and $K(z|y) \geq K(y|z)$, up to

an $O(\log K)$ additive term. Then the maxima in the terms of the equation $e(x,y) \leq e(x,z) + e(y,z)$ are determined, and our proof obligation reduces to

$$\frac{K(x|y)}{K(x)} \leq \frac{K(z|x)}{K(z)} + \frac{K(z|y)}{K(z)},$$

up to an additive $O((\log K)/K)$ term. We proceed as follows. Dividing both sides of the straightforward inequality $K(x|y) \leq K(x|z)+K(z|y)+O(1)$ by $K(x)$, we have

$$\frac{K(x|y)}{K(x)} \leq \frac{K(x|z) + K(z|y)}{K(x)},$$

up to an additive term of $O(1/K(x))$, where the left-hand side is less than or equal to 1.

Case 2.1. Assume that the right-hand side is ≤ 1. Set $K(z) = K(x)+\Delta$, and observe that $K(x|z)+\Delta = K(z|x)$ by the symmetry of information theorem, Theorem 3.9.2, up to an additive term $O(\log K)$. Add Δ to both the numerator and the denominator in the right-hand side of the last displayed equation, which increases the right-hand side because it is a ratio ≤ 1, and rewrite

$$\begin{aligned}\frac{K(x|y)}{K(x)} &\leq \frac{K(x|z) + K(z|y) + \Delta}{K(x) + \Delta} \\ &= \frac{K(z|x) + K(z|y)}{K(z)} + O(\log K/K),\end{aligned}$$

which was what we had to prove.

Case 2.2. The right-hand side is ≥ 1. We proceed as in Case 2.1, and add Δ to both numerator and denominator. Although now the right-hand side decreases, it must still be ≥ 1. □ □

8.4.2 Applications of Normalized Information Distance

The normalized information distance $e(x,y)$, which we call 'the' similarity metric because it accounts for the dominant similarity between two objects, is not computable. First we observe that using $K(x,y) = K(xy) + O(\log \min\{K(x), K(y)\}$ and the symmetry of information theorem, Theorem 3.9.2 on page 249, we obtain

$$\max\{K(x|y), K(y|x)\} = K(xy) - \min\{K(x), K(y)\},$$

up to an additive logarithmic term $O(\log K(xy))$, which we ignore in the sequel. If it is practically important, we can use $K(x,y)$ instead of $K(xy)$. In applications we use the compression of actual objects,

the worldwide web as background information, and mixed usage of the two. We consider phylogeny, clustering, classification, translation, and question–answer systems.

Compression

In order to use the NID in practice, admittedly with a leap of faith, our first approximation of the Kolmogorov complexity uses real compressors. This direction has yielded a very practical success of Kolmogorov complexity. Substitute the last displayed equation in the NID of Equation 8.12 on page 661, and subsequently use a real-world compressor Z (such as gzip, bzip2, PPMZ) to heuristically replace the Kolmogorov complexity. In this way, we obtain the distance e_Z, often called the *normalized compression distance* (NCD), defined by

$$e_Z(x,y) = \frac{Z(xy) - \min\{Z(x), Z(y)\}}{\max\{Z(x), Z(y)\}}, \tag{8.13}$$

where $Z(x)$ denotes the binary length of the compressed version of the file x, compressed with compressor Z. The distance e_Z is actually a family of distances parametrized with the compressor Z. The better Z is, the closer e_Z approaches the normalized information distance, the better the results are expected to be. Since Z is computable the distance e_Z is computable. In Exercise 8.4.6 on page 672 it is shown that under mild conditions on the compressor Z, the distance e_Z takes values in $[0,1]$ and is a metric, up to negligible errors.

Example 8.4.4 **(Phylogeny)** One cannot find more appropriate data than DNA sequences to test our theory. A DNA sequence is a finite string over a 4-letter alphabet $\{A, C, G, T\}$. We used the entire mitochondrial genomes of 20 mammals, each of about 18,000 base pairs, to test a hypothesis about the Eutherian orders. It has been hotly debated in biology which two of the three main placental mammalian groups, primates, ferungulates, and rodents, are more closely related. One cause of the debate is that in the analysis of the genomics the standard maximum likelihood method, which depends on the multiple alignment of sequences corresponding to an individual protein, gives (rodents, (ferungulates, primates)) for half of the proteins in the mitochondrial genome, and (ferungulates, (primates, rodents)) for the other half.

In recent years, as a result of more sophisticated methods, together with biological evidence, it is believed that (rodents, (ferungulates, primates)) reflects the true evolutionary history. We confirm this from the whole-genome perspective using the distance e_Z. We use the complete mitochondrial genome sequences from the following 20 species: rat (*Rattus norvegicus*), house mouse (*Mus musculus*), gray (or grey) seal (*Halichoerus grypus*), harbor seal (*Phoca vitulina*), cat (*Felis catus*), white rhino (*Ceratotherium simum*), horse (*Equus caballus*), finback whale

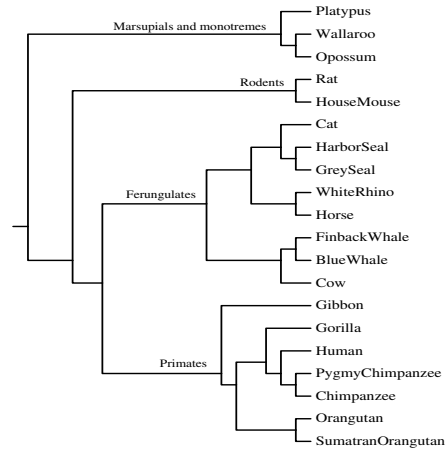


FIGURE 8.9. The evolutionary tree built from complete mammalian mtDNA sequences

(*Balaenoptera physalus*), blue whale (*Balaenoptera musculus*), cow (*Bos taurus*), gibbon (*Hylobates lar*), gorilla (*Gorilla gorilla*), human (*Homo sapiens*), chimpanzee (*Pan troglodytes*), pygmy chimpanzee (*Pan paniscus*), orangutan (*Pongo pygmaeus*), Sumatran orangutan (*Pongo pygmaeus abelii*), with opossum (*Didelphis virginiana*), wallaroo (*Macropus robustus*), and platypus (*Ornithorhynchus anatinus*) as the outgroup.

For every pair of mitochondrial genome sequences x and y, evaluate the formula in Equation 8.13 using a special-purpose DNA sequence compressor DNACompress, or a good general-purpose compressor such as PPMZ. The resulting distances are the entries in an $n \times n$ distance matrix. Constructing a phylogeny tree from the distance matrix, using common tree-reconstruction software, gives the tree in Figure 8.9. This tree confirms the accepted hypothesis of (rodents, (primates, ferungulates)), and every single branch of the tree agrees with the current biological classification. ◇

Similarity of sequences in biology is currently primarily handled using alignments. However, the alignment methods seem inadequate for postgenomic studies, since they do not scale well with data-set size and they seem to be confined only to genomic and proteomic sequences. Therefore, alignment-free similarity measures are actively pursued. In [P. Ferragina, R. Giancarlo, V. Greco, G. Manzini, and G. Valiente, *BMC Bioinformatics*, 8:1(2007) Jul 13,

252 17629909] the authors experimentally tested the normalized information distance using 25 compressors to obtain the NCD, and six data sets of relevance to molecular biology. They compared the methodology with methods based on alignments and not. They assessed the intrinsic ability of the methodology to discriminate and classify biological sequences and structures. The compression program PPMd, based on PPM (prediction by partial matching), for generic data and Gencompress for DNA, are the best performers among the compression algorithms they used. The quantitative analysis supports the conclusion that the normalized information/compression method is worth using because of its robustness, flexibility, scalability, and competitiveness with existing techniques. In particular, the methodology applies to all biological data in textual format.

Example 8.4.5 **(Hierarchical clustering)** The normalized compression distance has been used to fully automatically reconstruct language and phylogenetic trees as above. It can, and has, also be used for a plethora of new applications of general clustering and classification of natural data in arbitrary domains, for clustering of heterogeneous data, and for anomaly detection across domains. It has further been applied to authorship attribution, stemmatology, music classification, Internet knowledge discovery, to analyze network traffic and cluster computer worms and viruses, software metrics and obfuscation, web page authorship, topic and domain identification, hurricane risk assessment, ortholog detection, and clustering fetal heart rate tracings. We test gross classification of files based on heterogeneous data of markedly different file types: (i) four mitochondrial gene sequences, from a black bear, polar bear, fox, and rat obtained from the GenBank Database on the worldwide web; (ii) four excerpts from the novel *The Zeppelin's Passenger* by E. Phillips Oppenheim, obtained from the Project Gutenberg Edition on the worldwide web; (iii) four MIDI files without further processing, two works by Jimi Hendrix and two movements from Debussy's "Suite Bergamasque," downloaded from various repositories on the worldwide web; (iv) two Linux x86 ELF executables (the *cp* and *rm* commands), copied directly from the RedHat 9.0 Linux distribution; and (v) two compiled Java class files, generated directly. The program correctly classifies each of the different types of files together with like near like. The result is reported in Figure 8.10. This experiment shows the power and universality of the method: no features of any specific domain of application are used. We believe that there is no other method known that can cluster data that are so heterogeneous this reliably. ◇

Researchers from the data-mining community noticed that this methodology is in fact a parameter-free, feature-free, data-mining tool. They have experimentally tested a closely related metric on a large variety of sequence benchmarks. Comparing the compression-based method with 51 major parameter-loaded methods found in the 7 major data-mining conferences (SIGKDD, SIGMOD,

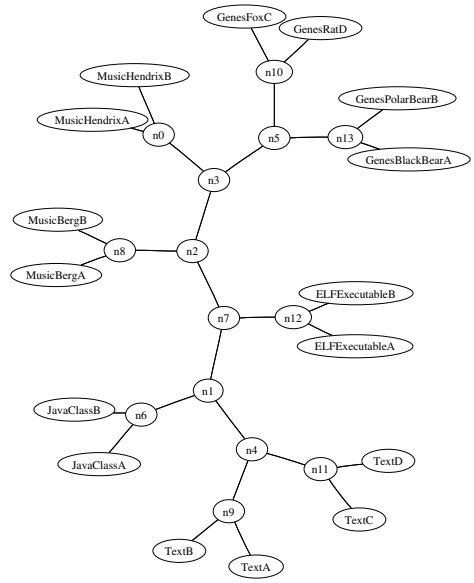


FIGURE 8.10. Clustering of heterogeneous file types

ICDM, ICDE, SSDB, VLDB, PKDD, and PAKDD) in the last decade, on every database of time sequences used, ranging from heartbeat signals to stock market curves, they established clear superiority of the compression-based method for clustering heterogeneous data, for anomaly detection, and competitiveness in clustering domain data [E.J. Keogh, S. Lonardi, and C.A. Rtanamahatana, *Proc. 10th ACM SIGKDD Int. Conf. Knowledge Discov. Data Mining*, 2004, 206–215; E.J. Keogh, S. Lonardi, C.A. Rtanamahatana, L. Wei, S.H. Lee, and J. Handley, *Data Min. Knowl. Disc.*, 14:1(2007), 99–129].

The Internet

Objects can be given literally, such as the literal four-letter human genome, or the literal text of *War and Peace* by Tolstoy. Objects can also be given by name, such as 'the four-letter human genome,' or 'the text of *War and Peace* by Tolstoy.' There are also objects that cannot be given literally, but only by name, and that acquire their meaning from their contexts in background common knowledge in humankind, such as 'home' or 'red.' A sequence contains information within itself. Names and abstract concepts also contain information, although not within themselves. The name 'human genome' implies three gigabases of information. The phrase '*War and Peace* by Tolstoy' perhaps carries information even beyond the book. While 'human genome' and '*War*

and Peace' can perhaps still be associated with sequences that can be compressed, the concept of 'home' or 'red' is even more problematic.

Can we find an equivalent of the normalized information distance for names and abstract concepts? Let us look at an alternative way of approximating the formula in Equation 8.12. Let $\mathbf{W}$ be the set of pages of the worldwide web, and let $\mathbf{x} \subseteq \mathbf{W}$ be the set of pages containing the search term x. By the conditional coding theorem, Theorem 4.3.4, we have $\log 1/\mathbf{m}(\mathbf{x}|\mathbf{x} \subseteq \mathbf{W}) = K(\mathbf{x}|\mathbf{x} \subseteq \mathbf{W}) + O(1)$, where $\mathbf{m}$ is the universal lower semicomputable discrete semimeasure of Theorem 4.3.1. This equality relates the incompressibility of the set of pages on the web containing a given search term to its universal probability. While we cannot compute $\mathbf{m}$, a natural heuristic is to use the distribution of x on the web to approximate $\mathbf{m}(\mathbf{x}|\mathbf{x} \subseteq \mathbf{W})$. Let us define the probability mass function $g(x)$ to be the probability that the search term x appears in a page indexed by a given Internet search engine G, that is, the number of pages returned divided by the overall number of pages indexed. (Actually, we need to account for the case that one page can contain more than one search term, and hence we should divide by a larger number. We ignore this issue here.) Then the Shannon–Fano code length of Example 1.11.2 on page 68 associated with g can be set at

$$G(x) = \log \frac{1}{g(x)}.$$

Replacing $Z(x)$ by $G(x)$ in the formula in Equation 8.13, we obtain the distance e_G, called the *normalized web distance* (NWD), our second heuristic approximation of normalized information distance, defined by

$$\begin{aligned} e_G(x,y) &= \frac{G(xy) - \min\{G(x), G(y)\}}{\max\{G(x), G(y)\}} \\ &= \frac{\max\{\log f(x), \log f(x)\} - \log f(x,y)}{\log N - \min\{\log f(x), \log f(y)\}}, \end{aligned} \tag{8.14}$$

where $f(x)$ is the number of pages containing x, the frequency $f(x,y)$ is the number of pages containing both x and y, and N is the total number of indexed pages. We can view the search engine G as a compressor using the web, and $G(x)$ as the binary length of the compressed version of the set of all pages containing the search term x, given the indexed pages on the web. The distance e_G is actually a family of distances parametrized with the search engine G. (Initially, the NWD was called NGD.) The better G is, the closer e_G approaches the normalized information distance (with $K(x)$ replaced by $K(\mathbf{x}|\mathbf{x} \subseteq \mathbf{W})$ and similarly the other terms), and the better the results are expected to be. In Exercise 8.4.8 on page 673 it is shown that the distance e_G is computable, takes values primarily (but not exclusively) in $[0,1]$, and is not a metric.

In the last two properties e_G differs from e and e_Z. Indeed, we should view the distance e_G between two names as a relative similarity semantics. Thus, while 'name1' is semantically close to 'name2,' and 'name2' is semantically close to 'name3,' 'name1' can be semantically very different from 'name3.'

Example 8.4.6 We describe an experiment, using a search engine G, performed in the year 2004. At the time, G indexed N =8,058,044,651 pages. A G search for 'horse' returned a page count of 46,700,000. A G search for 'rider' returned a page count of 12,200,000. A G search for both 'horse' and 'rider' returned a page count of 2,630,000. Thus $e_G(\text{horse}, \text{rider}) = 0.443$. It is interesting to note that this number stays relatively fixed as the number of pages indexed by G increases. ◇

Example 8.4.7 **(Classification)** In cases in which the set of objects can be large, in the millions, clustering cannot do us much good. We may also want to do definite classification, rather than the more fuzzy clustering. One can use the e_Z and e_G distances as an oblivious feature-extraction technique to convert generic objects into finite-dimensional vectors. We have used this technique to train a support vector machine (SVM) based OCR system to classify handwritten digits by extracting 80 distinct, ordered e_Z features from each input image, in the manner explained below in the context of e_G experiments. We achieved a handwritten single decimal digit recognition accuracy of 87%. The current state of the art for this problem, after half a century of interactive feature-driven classification research, is in the upper ninety percent level. These experiments were benchmarked on the standard NIST Special Data Base 19.

For classification using the e_G distance, the setting is, say, a binary classification problem on examples represented by search terms. In this experiment, we require a human expert to provide a list of at least 40 *training words*, consisting of at least 20 positive examples and 20 negative examples, to illustrate the contemplated concept class. The expert also provides, say, six *anchor words* $a_1, \ldots, a_6$, of which half are in some way related to the concept under consideration. Then, we use the anchor words to convert each of the 40 training words $w_1, \ldots, w_{40}$ to 6-dimensional *training vectors* $\bar{v}_1, \ldots, \bar{v}_{40}$. The entry $v_{j,i}$ of $\bar{v}_j = (v_{j,1}, \ldots, v_{j,6})$ is defined as $v_{j,i} = e_G(w_j, a_i)$ $(1 \leq j \leq 40,\ 1 \leq i \leq 6)$. The training vectors are then used to train an SVM to learn the concept, and then test words may be classified using the same anchors and trained SVM model. The LIBSVM software was used for all SVM experiments.

In an experiment to learn prime numbers, we used the literal search terms below (digital numbers and alphabetical words) in the Google search engine.
Positive training examples: 11, 13, 17, 19, 2, 23, 29, 3, 31, 37, 41, 43, 47,

5, 53, 59, 61, 67, 7, 71, 73.
Negative training examples: 10, 12, 14, 15, 16, 18, 20, 21, 22, 24, 25, 26, 27, 28, 30, 32, 33, 34, 4, 6, 8, 9.
Anchor words: composite, number, orange, prime, record.
Unseen test examples: The numbers 101, 103, 107, 109, 79, 83, 89, 91, 97 were correctly classified as primes. The numbers 36, 38, 40, 42, 44, 45, 46, 48, 49 were correctly classified as nonprimes. The number 110 was a false positive, since it was incorrectly classified as a prime. There were no false negatives. The accuracy on the test set is $18/19 = 94.74\%$. Thus, the method learns to distinguish prime numbers from nonprime numbers by example, using a search engine. ◇

The NWD has been used for clustering, classification, and translation of small samples. R.L. Cilibrasi and P.M.B. Vitányi [*IEEE Trans. Knowledge Data Engin.*, 19:3 (2007), 370–383] report a massive experiment comparing the performance of the NWD–SVM method with the human-expert-entered information in the WordNet database [G.A. Miller et al., "WordNet, A Lexical Database for the English Language," Cognitive Science Lab, Princeton University]. They showed a mean accuracy of agreement of 87.25% of the NWD–SVM method with the WordNet semantic concordance.

Example 8.4.8 **(Translation)** Assume that there are five words that appear in two different matched sentences, but the permutation associating the English and Spanish words is, as yet, undetermined. Let us say, *plant, car, dance, speak, friend* versus *bailar, hablar, amigo, coche, planta*. At the outset we assume a preexisting vocabulary of eight English words with their matched Spanish translations: *tooth, diente; joy, alegria; tree, arbol; electricity, electricidad; table, tabla; money, dinero; sound, sonido; music, musica*. Can we infer the correct permutation mapping the unknown words using the preexisting vocabulary as a basis?

We start by forming an English basis matrix in which each entry is the e_G distance between the English word labeling the column and the English word labeling the row. We label the columns by the translation-known English words, and the rows by the translation-unknown English words. Next, we form a Spanish matrix with the known Spanish words labeling the columns in the same order as the known English words. But now we label the rows by choosing one of the many possible permutations of the unknown Spanish words. For every permutation, each matrix entry is the e_G distance between the Spanish words labeling the column and the row. Finally, choose the permutation with the highest positive correlation between the English basis matrix and the Spanish matrix associated with the permutation. If there is no positive correlation report a failure to extend the vocabulary. The method inferred the correct permutation for the testing words: *plant, planta; car, coche; dance, bailar; speak, hablar; friend, amigo*. ◇

Unification

The normalized information distance e is intended to be universally applicable. In practice, various computable distances, including e_Z and e_G, can be viewed as approximations to e.

For example, all 21 measures studied in [P.N. Tan, V. Kumar, and J. Srivastava, *Proc. 8th ACM SIGKDD Int. Conf. Knowledge Discov. Data Mining*, 2002, 32–44] may after normalization be viewed as various degrees of approximations to e (the NID), or to $e_{\min}$ defined in Exercise 8.4.9.

Apart from providing a theoretical justification for these practical distances, the normalized information distance does more in that it embodies all approximations. This fact turns out to be also practically important, for instance in the question–answer search engine QUANTA, which we briefly touch upon but do not explain in any detail. The information distance between a question and an answer in QUANTA is measured by e and $e_{\min}$ defined in Exercise 8.4.9 on page 673. The Kolmogorov complexity terms in e and $e_{\min}$ are approximated as follows in QUANTA:

- Approximate pattern matching with query sentence, using e_Z style approximation.
- Shannon–Fano code approximation, as in e_G, for short conceptual answers.
- Mixed usage of the above two, choosing the shortest encoding.

Thus, within the one Equation 8.12 on page 661, and its relative $e_{\min}$, different K terms may be approximated by different methods, whichever is the shortest. Normalized information distance provides a unification of all metrics, not only for theoretical studies, but also for practical applications.

Exercises

8.4.1. [33] (a) Let $x, y \in \mathcal{R}^n$ ($\mathcal{R}$ denotes the real numbers) and denote by $\|x-y\|$ the Euclidean metric—the L_2 norm. Show that the normalized Euclidean distance $\|x-y\|/(\|x\|+\|y\|)$, has values in $[0,1]$ only and is also a metric.

(b) Let A, B be finite sets and $A \Delta B = A \bigcup B - (A \bigcap B)$ the symmetric set difference. The symmetric set difference cardinality $d(A \Delta B)$ is a metric on the family of finite sets. Show that the normalized symmetric set difference cardinality $d(A \Delta B)/d(A \bigcup B)$ has values in $[0,1]$ only and is also a metric.

(c) The sum of conditional entropies $H(X|Y)+H(Y|X)$ of joint discrete random variables X, Y is a metric. Show that the normalized version $H(X|Y)+H(Y|X)/H(X,Y)$ has values in $[0,1]$ only and is also a metric.

Comments. The normalized version of Item (c) can be written as $1 - I(X;Y)/H(X,Y)$. It is the random-variable equivalent of the normalized sum distance $(K(x|y)+K(y|x))/(K(x,y))$ in Exercise 8.4.5. But the random-variable variant is about expectations, and its proof is quite different from the Kolmogorov complexity variant that is about individual strings. That the normalized form in Item (a) is a metric is true for the Euclidean metric but for no other integral Minkowski metrics. Source for Items (a) and (b): P.N. Yianilos, "Normalized Forms for Two Common Metrics," NEC Research Institute Technical Report, 1991, 2002, where Item (a) is attributed to D. Robbins and M. Buck as Private Communication, May 1993; for Item (c): C. Rajski, *Inform. Contr.*, 4(1961), 371–377.

8.4.2. [22] Show that the NID of Definition 8.4.1 on page 661 satisfies $d(\{y : e(x,y) \leq \delta \leq 1\}) < 2^{\delta K(x)+1}$.

Comments. The density requirement is implied by a normalized version of the Kraft inequality, $\sum_{y:x\neq y} 2^{-e(x,y)K(x)} \leq 1$ for every x. Source: M. Li, X. Chen, X. Li, B. Ma, and P.M.B. Vitányi, *IEEE Trans. Inform. Theory*, 50:12(2004), 3250–3264.

8.4.3. [22] Show that we can reduce the error term in Theorem 8.4.1 to $O(1/K)$.

Comments. Use Lemma 3.9.1 on page 249. Source: M. Li, X. Chen, X. Li, B. Ma, and P.M.B. Vitányi, *Ibid.*

8.4.4. [29/O42] (a) Show that the NID of Definition 8.4.1 on page 661 is not computable.

(b) Call a function $f(x,y)$ *computable in the limit* if there exists a rational-valued computable function $g(x,y,t)$ such that $\lim_{t\to\infty} g(x,y,t) = f(x,y)$. Show that this is precisely the class of functions that are Turing-reducible to the halting set, and that the NID is in this class.

(c) Prove or disprove that the NID is a semicomputable function.

Comments. The NID, being a ratio between two maxima of pairs of upper semicomputable functions, may not itself be semicomputable. It is easy to see that this is likely, but a formal proof is lacking. Source for Item (b): W. Gasarch, email of 12 August, 2001. Item (c) is open.

8.4.5. [25] The sum distance $E_4 = K(x|y) + K(y|x)$ is a metric by Theorem 8.3.6. Prove that its normalization defined by $e_4 = (K(x|y) + K(y|x))/K(x,y)$ takes values in $[0,1]$ and is also a metric.

Source: M. Li, J.H. Badger, X. Chen, S. Kwong, P. Kearney, and H. Zhang, *Bioinformatics* 17:2(2001), 149–154.

8.4.6. • [38] Let $Z(x)$ denote the binary length of the compressed version of x using compressor Z. A compressor Z is *normal* if it is a

real-world compressor and satisfies the axioms (identity) $Z(xx) = Z(x)$ and $Z(\lambda) = 0$ where λ is the empty string, (monotonicity) $Z(xy) \geq Z(x)$, (symmetry) $Z(xy) = Z(yx)$, and (distributivity) $Z(xy) + Z(z) \leq Z(xz) + Z(yz)$, all (in)equalities up to an additive $O(\log n)$ term, with n the maximal binary length of a string involved in the (in)equality concerned.

(a) Show that $E_Z(x, y) = Z(xy) - \min\{Z(x), Z(y)\}$ with Z a normal compressor, is computable, satisfies the density requirement in Equation 8.11 on page 646, and satisfies the metric (in)equalities up to additive $O(\log n)$ terms, with n the maximal binary length of a string involved in the (in)equality concerned.

(b) Show that the distance e_Z of Equation 8.13 on page 664, a normalized version of $E_Z(x, y)$, with Z a normal compressor, has values in $[0, 1]$ and satisfies the metric (in)equalities up to additive $O((\log n)/n)$ terms, with n the maximal binary length of a string involved in the (in)equality concerned.

Comments. Informal experiments by the authors of the source reference below have shown that these axioms are in various degrees satisfied by good real-world compressors such as gzip, bzip2, and PPMZ, where the last one is best among the ones tested. Reference [M. Cebrián, M. Alfonseca, and A. Ortega, *Commun. Inform. Syst.*, 5:4(2005), 367–384] systematically investigated how far the performances of real-world compressors gzip, bzip2, and PPMZ satisfy the identity axiom $Z(xx) = Z(x)$ of a normal compressor Z. Source: R.L. Cilibrasi and P.M.B. Vitányi, *IEEE Trans. Inform. Theory*, 51:4(2005), 1523–1545.

8.4.7. [O45] The NID is not computable. The distance e_Z is a heuristic approximation that can deviate from the NID, notwithstanding the fact that it is successfully applied. Can you formulate another, better, practical theory of a normalized compression distance using a real-world compressor than in Exercise 8.4.6?

8.4.8. [31] Show that the distance e_G of Equation 8.14 on page 668 is computable, takes values primarily in $[0, 1]$ but also outside in pathological cases, and is not a metric since it violates $e_G(x, y) \neq 0$ for $x \neq y$, and $e_G(x, y) + e_G(y, z)$ can be less than $e_G(x, z)$.

Comments. Source: R.L. Cilibrasi and P.M.B. Vitányi, *IEEE Trans. Knowledge Data Engin.*, 19:3(2007), 370–383.

8.4.9. [28] Let U be the reference universal prefix machine. The cost of two-way conversion between x and y in the sense of a new distance E_5 is defined by $E_5(x, y) = \min\{l(p) : U(x, p, r) = y,\ U(y, p, q) = x\}$, the minimum taken over all p, q, r such that $l(p) + l(q) + l(r) \leq E(x, y)$. This definition separates out r as the extra information for x, and q as the extra information for y.

(a) Show that $E_5(x,y) = \min\{K(x|y), K(y|x)\}$, up to an $O(\log l(xy))$ additive term, which we will ignore.

(b) Show that $E_5(x,y)$ does not satisfy the triangle inequality.

(c) Show that $E_5(x,y)$ satisfies the density conditions in Equation 8.11 on page 646 only for x's with $K(x) \geq l(x) + O(1)$. This is perhaps not a surprise, since the sum distance $E_4(x,y)$ is equal to $E_5(x,y) + E_1(x,y)$. In the new metric E_5, 'good guys' (Kolmogorov simple objects) have even more neighbors than in E_4.

(d) Show that $E_5(x,y) = \max\{K(x), K(y)\} - I(x:y)$.

(e) Define $e_{\min}(x,y) = \min\{K(x|y), K(y|x)\}/\min\{K(x), K(y)\}$. (Thus, $e_{\min}(x,y) = E_5(x,y)/\min\{K(x), K(y)\}$.) Show that $e_{\min}(x,y) \leq e(x,y)$.

(f) Define $e'_{\min}(x,y) = E_5(x,y)/\min\{l(x), l(y)\}$. Prove the following universality statement: for every computable distance d satisfying the density condition $d(\{y : l(y) = n,\ d(x,y) \leq \delta \leq 1\}) \leq 2^{\delta n}$ we have $e'_{\min}(x,y) \leq d(x,y) + O(1/n)$ with $n = \min\{l(x), l(y)\}$.

(g) Show that $e_{\min} = 1 - (I(x:y) - \Delta)/\min\{K(x), K(y)\}$ with $\Delta = |K(x) - K(y)|$.

Comments. In question–answer systems on the Internet, as in the discussion on page 671, distances are measured with partial information; hence it is unreasonable to require the triangle inequality to hold. Also, popular concepts have a much larger neighborhood; hence the density condition in Exercise 8.3.1 on page 657 may also not hold. Furthermore, between two concepts, there is a large quantity of irrelevant information we may wish to get rid of. The definition of $e_{\min}$ above deals with this practical situation. Hint: for Item (a), use the proof ideas of Theorem 8.3.1. By Item (a) the distance E_5 equals the maximal overlap in Corollary 8.3.1 on page 645, up to an additive logarithmic term. Source for Items (a), (b), (c), (e), (f): M. Li, *Int. J. Found. Comput. Sci.*, 18:4(2007), 669–681.

8.5 Thermodynamics

Classical thermodynamics deals with describing the physical properties of substances such as gases under variations of macroscopic observables such as volume, temperature, and pressure. Certain regularities are codified on the basis of experience and formulated as axioms of the theory. The two fundamental laws of thermodynamics are as follows:

First Law. The total energy of an isolated system is invariant in time.

Second Law. No process is possible that has as its only result the transformation of heat into work.

The first law is a statement of the principle of conservation of energy for thermodynamical systems. The variation in energy of a system during any transformation is equal to the amount of energy that the system receives from its environment, minus the amount of energy the system loses to its environment. The first law arose as the result of the impossibility of constructing a machine that would create energy (*perpetuum mobile* of the first kind). It places no limitations on the possibility of transforming energy from one form into another, such as the transformation of heat into work or work into heat.

The first law still allows a machine that transforms heat into work by cooling the environment (*perpetuum mobile* of the second kind). The second law rules out this possibility. An equivalent formulation of the second law is, "no process is possible that has as its only result the transfer of heat from a cooler to a warmer body." To prove that the two formulations are equivalent, it suffices to show that a process can be created that transforms heat from a cooler body to a warmer body if a process is available that transforms heat into work, and conversely; this is not difficult.

8.5.1 Classical Entropy

The original definition of entropy was proposed in 1824 by the French engineer N.L. Sadi Carnot in the context of the so-called Carnot cycle. This is a cycle of states of thermodynamic systems such as steam engines. The existence of an entropy can be derived from the second law by reasoning that depends on the notion of *reversible processes*. The two standard examples are the following:

Example 8.5.1 We have a cylinder filled with an ideal gas, in contact with a heat reservoir at temperature T_1. Allow the gas to push out a frictionless piston very slowly. The gas will expand while staying at the same temperature and meanwhile will take up heat from the reservoir. This is (in the limit) a reversible process: by pushing in the piston again very slowly, the whole system can be brought back into exactly the same condition as it was before the expansion. ◇

Example 8.5.2 Use the same cylinder as before, but now in thermal isolation. Again pull out the piston very slowly. The temperature of the gas will fall. This is again (in the limit) a reversible process. ◇

A concatenation of reversible processes is again reversible. By combination of the two types of processes just mentioned, one can obtain a reversible *cycle*: first isothermic expansion, then adiabatic expansion, then isothermic compression, and finally adiabatic compression to the initial state. This is called a *Carnot cycle*. The whole process can be plotted in the (V, P)-plane, Figure 8.11, where V denotes volume and

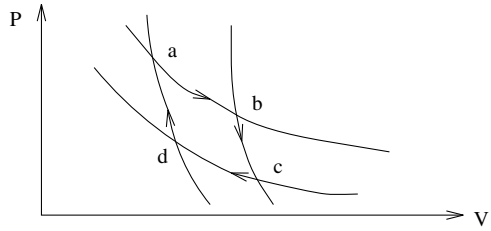


FIGURE 8.11. Carnot cycle

P denotes pressure. The curves corresponding to isothermic expansion and compression satisfy PV is constant (actually, $PV = RT$ where T is the temperature and $R > 0$ is a gas-specific constant). The curves corresponding to adiabatic expansion and compression satisfy PV^γ is constant, where $\gamma > 1$ is another gas-specific constant.

In terms of heat and work, during a Carnot cycle the gas in the cylinder consumes an amount of heat ΔQ^+ from the reservoir at temperature T_1, and delivers an amount of heat ΔQ^- to another reservoir at a lower temperature T_2. The total amount of work performed in the cycle is equal to $\int P\, dV$, which is exactly the surface area enclosed in Figure 8.11. A Carnot cycle is therefore a form of a heat engine.

According to the first law of thermodynamics, the amount of work done by a heat engine, Figure 8.12, must equal the difference between the heat extracted from the warm reservoir and the heat delivered to the cold reservoir,

$$\Delta U = \Delta Q^+ + \Delta Q^-,$$

where ΔQ^- is negative.

Assuming the process to be reversible, one can derive a further relation. Denote by a, b, c, d the four intermediate stages of the Carnot cycle,

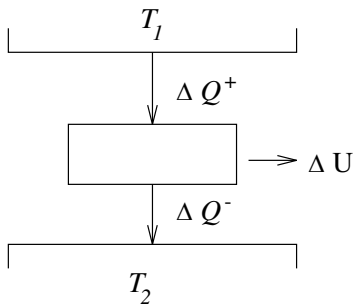


FIGURE 8.12. Heat engine

Figure 8.11. Then

$$\Delta Q^+ = \int_a^b P\, dV = \int_a^b RT_1 \frac{dV}{V} = RT_1 \log \frac{V_b}{V_a}.$$

Likewise,

$$\Delta Q^- = RT_2 \log \frac{V_d}{V_c}.$$

The relations $P_b V_b^\gamma = P_c V_c^\gamma$, $P_b V_b^\gamma = RT_1$, and $P_c V_c^\gamma = RT_2$ yield

$$T_1 V_b^{\gamma-1} = T_2 V_c^{\gamma-1},$$

and in the same way,

$$T_1 V_a^{\gamma-1} = T_2 V_d^{\gamma-1}.$$

Dividing, we obtain $V_b/V_a = V_c/V_d$ and therefore

$$\frac{\Delta Q^+}{T_1} + \frac{\Delta Q^-}{T_2} = 0.$$

Although this was derived for the Carnot cycle, it follows from the second law that this formula must be universal for reversible processes. Namely, if we had a reversible process that did not satisfy the relation above, then we could couple it to a Carnot cycle in a suitable way such as to fabricate a process that would violate the second law. For general processes, the law that we have just derived takes the following form:

$$\oint \frac{d\,Q}{T} = 0, \text{ along reversible paths.}$$

(An alternative form of the integral with the time parameter appearing explicitly is $\oint \dot{Q}(t) T^{-1}(t)\, d\,t$.) This property makes it possible to define the entropy, as a function of temperature and volume, as follows:

$$S(T_b, V_b) = S(T_a, V_a) + \int_a^b \frac{d\,Q}{T},$$

where the integral is taken along a reversible path. It follows from the above that the answer does not depend on the choice of the reversible path. This is the classical thermodynamic definition of entropy. It determines the *entropy* S only up to an additive constant. If the process is irreversible, then

$$\oint dS > 0. \tag{8.15}$$

Thus, the entropy always increases with time, except for reversible systems, where it can stay the same. This is the classic second law.

Example 8.5.3 We compute the entropy of an ideal gas at a constant temperature T, so that the entropy will be a function only of volume. Using the ideal gas law $PV = RT$, we obtain

$$S(V_b) - S(V_a) = \int_a^b \frac{d\,Q}{T} = \frac{1}{T}\Delta Q = \frac{1}{T}\int_a^b P\,dV$$
$$= \frac{1}{T}\int_a^b \frac{RT}{V} d\,V = R(\log V_b - \log V_a).$$

This result may already be related to the information-theoretic entropy, as follows: Assume that a volume is divided into cells, where each cell can contain either one molecule or no molecule. A typical gas molecule may be found with equal probabilities in one of N_a cells in volume V_a and in one of N_b cells in volume V_b. By Definition 1.11.1 on page 67, of Shannon's entropy, the difference in entropies is proportional to $\log N_b - \log N_a = \log V_b - \log V_a$. If the molecules are independent, then we can just multiply by the number of molecules to obtain the difference of the total entropies. In this simple case, the information-theoretic Definition 1.11.1 is in line with the thermodynamic case. ◇

8.5.2 Statistical Mechanics and Boltzmann Entropy

The above theory belongs to phenomenological thermodynamics. However, it is desirable to derive all laws in physics from first principles. Eventually, a gas consists of molecules that move about according to the laws of mechanics. This can be classical or quantum. But whether classical or modern, the known equations of physics are reversible in time. This implies that if a system has a certain trajectory of its evolution from state a to state b, then there also is a trajectory of evolution from state b to state a. In the classical case, it suffices to reverse all final velocities of all particles and the system will trace its trajectory backward.

This reversibility of equations seems to be in contradiction to the above considered irreversibility of some thermodynamic processes. L. Boltzmann was the first to formulate the idea that entropy measures disorderliness, and to connect this with the origin of the distinction between past and future in a nonrelativistic universe. This distinction is obvious in all our immediate experiences. We have no hesitation to identify a movie that is run backwards by, say, a broken vase becoming whole again. It would be nice if this conviction could be justified from the laws of nature. But the laws of physics (as far as known) tell a different story: if the movie run in one direction is physically appropriate, then so is the movie run in the opposite direction.

A refinement of the notions of macroscopic state and reversibility is required. Roughly speaking, the *macro state* of an isolated mechanical

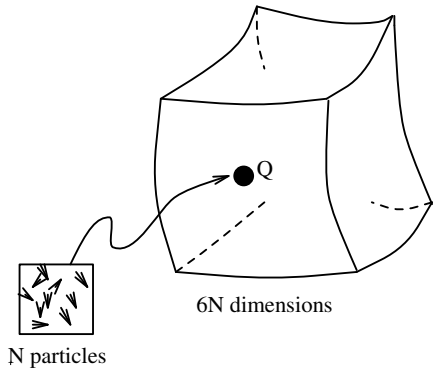


FIGURE 8.13. State space

system consists of an approximate descriptions of the values of just a few macroscopic observables such as volume, temperature, and pressure.

The *micro state* of such a system determines the behavior of the system completely, whether it is in equilibrium or not. Roughly speaking, a micro state of a physical system consists of an exhaustive description of the values of all the microscopic parameters of the particles. Statistical thermodynamics, also called statistical mechanics, tries to derive the classical laws of thermodynamics from microscopic phenomena.

Ideal gas is a convenient tool to study statistical thermodynamics. Consider a (3-dimensional) container with N identical (ideal) gas molecules. Each molecule is considered as an elastic sphere (also called ball) with no internal freedom. The behavior of the gas is completely determined if we specify the position, mass, and velocity of every molecule at some time instant. Instead of the velocity, it is more convenient to use the *impulse momentum*, which is *mass* times *velocity*, of every molecule.

The position of a molecule is specified by three space coordinates. The momentum of a molecule is a vector in 3-space and hence is also specified by three coordinates. To specify N molecules we need $6N$ parameters.

Definition 8.5.1 Since $N \approx 6 \times 10^{23}$ (Avogadro's number) for 1 *mole* of gas (1 mole of hydrogen is 2 grams), such a system is hard to analyze. It is convenient to use a fictional space X, the so-called *state space* of the system (also called *phase space*), of $6N$ dimensions. There is a one-to-one correspondence between the dimensions and the coordinates of the molecules. A *micro state* of the entire system of N molecules corresponds to just one point,

$$Q = (q_1, \ldots, q_{3N}, p_1, \ldots, p_{3N}),$$

in state space, where the q_i's specify the coordinates of the positions of N molecules and the p_i's specify their momenta, Figure 8.13.

Each coordinate is a real number. If $(\omega^1, \ldots, \omega^{6N})$ is a micro state, then it can be encoded as a single infinite binary sequence ω.

An encoding such as representing the ith bit of ω^j by the $(j+6(i-1)N)$th bit of ω is improper. Since typically the number of molecules is about 6×10^{23}, the first 10^{23} bits of ω give little information about the state of the system. Our encoding should have the property that the consecutive bits contain information about the micro state in decreasing order of importance from a macroscopic point of view. For example, the first few bits may describe, to a reasonable degree of precision, the amount of gas in each half of the container, the next few bits may describe the amounts in each quarter, the next few bits may describe the temperature in each half, the next few bits may describe again the amount of gas in each half, but now to more precision, and so on. We can now divide our space into cells of different grain size.

We shall assume that the micro state moves around in a closed volume of the state space, of a standard number of units in each dimension. Then for each dimension we can assume the position of the decimal point known, and we ignore it in the binary expansion of the coordinates. The set of micro states of the state space is denoted by $X \subseteq \{0,1\}^\infty$.

The total volume of the space need not measure to 1. For example, if λ denotes the Lebesgue measure (or uniform measure), and our state space is a $6N$-dimensional hypercube R with each side 2 units, then $\lambda(R) = 2^{6N}$. We restrict ourselves to measures that are finite over the total volume, so that the difference with a probability measure is just a matter of multiplication by a scaling constant.

Notation 8.5.1 In Notation 4.2.1 on page 263 we introduced the shorthand notation of $\mu(x)$ for the μ-measure $\mu(\Gamma_x)$ of a cylinder set Γ_x. In this section we use the common standard notation $\mu(A)$ for the μ-measure of a set A.

Now suppose there are m macroscopic parameters $u_1, \ldots, u_m$. We also will assume that each such parameter takes only a finite number of values: a macroscopic parameter that is a real number will be taken up only to some reasonable precision (four digits are probably more than sufficient). In this way we obtain a partition of the state space X into cells

$$X = \bigcup_x \Gamma_x,$$

where x runs over all possible values of parameters $u_1, \ldots, u_m$. Here each Γ_x consists of the set of all micro states that can occur under macroscopic constraints x.

Definition 8.5.2 Let x be as above. A *macroscopic state* of the system, or *macro state*, is a cell Γ_x. Each cell is identified with the corresponding set of micro states. The partition interpretation of a macro state is called *coarse-graining*.

Example 8.5.4 Suppose we subdivide a container into $10 \times 10 \times 10$ parts. Determine the pressure in each part to within 0.1 atmosphere. It is important to note that the state-space partition according to the different values of the macro description will have hugely different cell sizes. Therefore, the cell volumes will depend on the partition, but these differences are for reasonable partitions negligible in comparison to the cell-size differences arising already within a fixed partition. ◇

According to the laws of mechanics, an isolated system undergoes an evolution described by a transformation group U^t. If at time t_0, the system was in state ω, then at time $t+t_0$ it will be in state $U^t\omega$. The group U^t is generally given by a system of differential equations that, at least in the example of ideal gas given with coordinates and impulses, are called the *Hamiltonian equations*. For this case, *Liouville's theorem* holds, saying that the volume of a domain remains invariant under transformation under U^t. In most other cases also, a natural measure is found on X that remains invariant under U^t, and we will call this measure the *volume*, and denote the volume of a set A by $L(A)$.

The law of energy conservation means that during an evolution of an isolated system, it is confined to a surface of the state space determined by the requirement that the energy be equal to a certain value. Therefore the volume measure to use will be actually obtained by restricting the original volume measure to a thin layer determined by the requirement that the value of energy be in a certain small interval, and normalizing.

When we say that the transformation of a macroscopic description x to macroscopic description y is reversible, then this statement refers to the macro states Γ_x, Γ_y, and what is meant is that the existence of a reverse transformation follows already from the properties of macro states. The question arises what reversibility means in terms of micro states.

Since macro state x contains many micro states, the question is whether this means that a reverse transformation exists for all elements of Γ_y or only for some. It is easy to see that it is too much to require that the reverse transformation exist for all micro states in Γ_y, and too little that it exist only for some. The answer is that the transformation is reversible if the reverse transformation exists for most (by volume) micro states in Γ_y. Boltzmann considered as his greatest achievement to have found a microscopic expression for entropy.

Definition 8.5.3 The *Boltzmann entropy* of a system with macroscopic description x is proportional to the logarithm of the volume of Γ_x,

$$S_B(x) = (k \ln 2) \log L(\Gamma_x),$$

where $k = 1.38 \times 10^{-23}$ joules/kelvin is called the *Boltzmann constant*, and Γ_x is the cell in state space corresponding to the macroscopic description x of the system, and $L(\Gamma_x)$ is its volume.

Example 8.5.5 For the special case of a discrete state space with L the counting measure defined as $L(\Gamma_x) = d(\Gamma_x)$, that is, the number of elements in the set Γ_x,

$$S_B(x) = (k \ln 2) \log d(\Gamma_x)$$

is the familiar form of the Boltzmann entropy. ◇

Example 8.5.6 Consider a container of ideal gas consisting of n identical molecules. Let the container be partitioned into m compartments $C_1, \ldots, C_m$, where m is much smaller than n. For simplicity, we ignore the velocities. Let the macro variable n_i give the number of molecules in compartment C_i. Let $n = (n_1, \ldots, n_m)$. A second set of variables, the numbers $i_1, \ldots, i_n$, indicates that molecule j is in compartment i_j.

The description $i = (i_1, \ldots, i_n)$ is, of course, more detailed than the description n, which is a function $n(i)$ of i. Therefore $\Gamma_n = \bigcup_{n=n(i)} \Gamma_i$.

Since the molecules are identical, the dynamics, therefore the measure L, will certainly be invariant with respect to the exchange of molecules. Therefore if $n(i) = n(j)$, then $L(\Gamma_i) = L(\Gamma_j)$. Assume that molecules are distinguishable. Then $L(\Gamma_n) = N(n)p_n$, where p_n is the volume of each Γ_i, with $n(i) = n$ and

$$N(n) = \frac{n!}{n_1! \cdots n_m!}.$$

If we also assume that all the compartments C_i have the same shape and size, and hence have the same volume V, then p_n does not depend on n. Indeed, we have $p_n = V^n$, the nth power of the volume of a single compartment, since with i fixed only the exact position of each molecule in its compartment is undetermined. The Boltzmann entropy of state n is equal to

$$S_B = (k \ln 2) \log N(n) \approx (k \ln 2) n \left(\sum_i f_i \log \frac{1}{f_i} + \log V \right),$$

where $f_i = n_i/n$. If the molecules are indistinguishable, then it is necessary to subtract $\log n!$. ◇

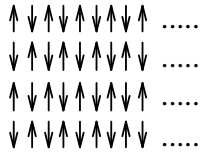


FIGURE 8.14. Atomic spin in CuO_2 at low temperature

Example 8.5.7 In high-temperature superconductivity research, a material like CuO_2 loses magnetic moment below a critical temperature. In such a state, the nuclear spins all line up as in Figure 8.14. For Boltzmann entropy, one must agree on some standard partition (coarse-graining) of the space into subsets, and the entropy of x is the negative logarithm of the measure of the subset that x is in. There is much arbitrariness in the choice of the partition, but the differences caused by different reasonable choices of partition are negligible compared to the size of the entropy. It is important that the partition be a reasonable one.

A *reasonable partition* is a discretization of the state space in which the values of the most important quantities come first. In the case of the ferromagnetic example, the sequence $x_1, x_2, \ldots$ would not be simply the sequence of spins but would begin with various global statistical quantities, since presumably these can be measured. One may put first the total number of 'up' spins, at least to some precision, then the total number of 'up' spins in the left half, and so on. Then, some time later, one may put the total number of neighborhoods of the form as in Figure 8.15.

Let us call this last number z. If the total number of atoms is N, then in Figure 8.14 we have approximately $z = N/2$. On the other hand, there is only a very small number of configurations with $z = N/2$, or even in which z is almost $N/2$, since they must all essentially look like this. Therefore, the Boltzmann entropy of this configuration is very low.

But suppose that the ups and downs encode the bits of $3.1415\ldots$. Then in the algorithmic sense the complexity is small, but this will be seen only when the size of the neighborhoods is chosen as large as N. Considering only local neighborhoods, the statistical properties suggest randomness. In physics, it is assumed that we never look at other than global statistical (macroscopic) quantities; hence, when the ferromagnet encodes the digits of π, this will forever be its secret. ◇

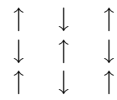


FIGURE 8.15. Regular 'up' and 'down' spins

The most obvious problem with this definition of Bolzmann entropy seems to be its dependence on the particular choice and number of macroscopic variables and the precision with which we want to determine them. Indeed, another digit of precision will decrease the Boltzmann entropy of most states by about $\log 10$. The volumes in question are, however, typically so large that such a small difference is negligible (especially if we take into consideration that the actual definition of $S_B(x)$ involves multiplication by the very small Boltzmann constant k).

Example 8.5.8 Boltzmann entropy possesses the entropy increase properties as required by the second law. This is due to the enormous differences in cell sizes irrespective of a small number of cells. Classical statistical-mechanical systems are distinguished by the fact that every reasonable canonical cell division will have this property.

For instance, suppose we are dealing with a vessel containing a mole of gas, Definition 8.5.1 on page 679, and our macroscopic variable x is the binary sequence x_1x_2 giving approximately the relative quantity of gas in the left half of the container. Then the cells Γ_{00} and Γ_{11} are absolutely negligible in volume compared to the cells Γ_{01} and Γ_{10}.

Since entropy measures the amount of state space occupied by a macro state, it sounds plausible that a system tends to be in states with high entropy. According to one formulation of the second law of thermodynamics, entropy in isolated systems cannot decrease. There are some other related entropy increase properties, such as

- an isolated system undergoes an irreversible transformation exactly when entropy actually increases;
- an equilibrium state of an isolated system is a maximum (or at least a strongly pronounced local maximum) of entropy.

It is most important that Boltzmann entropy can be shown to increase in time until equilibrium is reached. Since there are huge differences in the cell volumes $L(\Gamma_x)$ for different macro states Γ_x, typically U^t takes a point from a small cell to a bigger cell. Just as the precise notion of reversibility had to be formulated statistically, the same can be expected for these entropy increase properties for Boltzmann entropy.

It turns out that for certain special cases one can prove two properties that together imply the desired entropy increase properties for Boltzmann entropy. The first property states that the union of all small cells taken together is small. The second property states that if the system starts from a state in a not very small cell then after a time t, it is unlikely to end up in any small union of cells. ◇

Example 8.5.9 Let us go back to Example 8.5.7 on page 683. The state of CuO_2 in Figure 8.14 can be described by a program of a few bits:

repeat forever : **print** $\uparrow$; **print** $\downarrow$.

Thus, the complexity term in the algorithmic entropy almost vanishes. Similarly, the sequence of arrows that encodes 3.1415... has a very low complexity term, and corresponding low algorithmic entropy. ◇

Example 8.5.10 Adiabatic demagnetization is an important technique that has been used with great success to achieve record low temperatures (near zero kelvin). A sample such as a chrome-alum salt (whose molecules may be considered as tiny magnets) is placed in a thermally insulating (adiabatic) enclosure at the lowest attainable temperature during cooling. A strong magnetic field is applied by an external magnet so that the tiny atomic magnets (spins) line up, forming a very ordered state, as Figure 8.16(a) shows. Then the magnet is removed so that its field is no longer present.

Redistributing the energy evenly among all the degrees of freedom lowers the temperature of the specimen, since the temperature is derived only from some of these degrees of freedom (the translational velocities). But the state of having the same amount of energy in each degree of freedom has the highest entropy. This clearly includes a significant increase of Kolmogorov complexity of the spins. ◇

8.5.3 Gibbs Entropy

A third definition of entropy is due to J.W. Gibbs. Consider some thermodynamic system. Another possible interpretation of a macro state is as a certain distribution over microscopic states. An *ensemble* is some measure with density $p(\omega)$ over the state space X interpreted as a distribution of individual points satisfying $\int p(\omega)L(d\omega) = 1$. The ensemble $p_\Gamma(\omega)$ corresponding to a macro state Γ is defined as

$$p_\Gamma(\omega) = \begin{cases} 1/L(\Gamma), & \omega \in \Gamma, \\ 0, & \text{otherwise.} \end{cases}$$

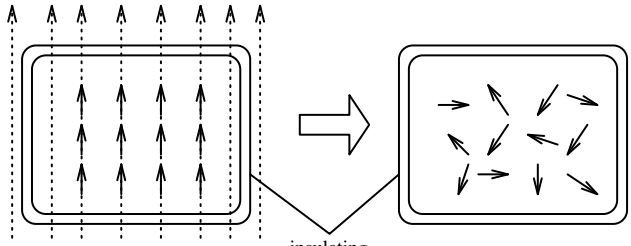


FIGURE 8.16. Adiabatic demagnetization to achieve low temperature

Definition 8.5.4 The *Gibbs entropy* of a cell Γ is defined as

$$S_G(p) = (k \ln 2) \int_\Gamma p(\omega) \log \frac{1}{p(\omega)} L(d\,\omega). \tag{8.16}$$

Example 8.5.11 Let X be the state space of a thermodynamic system, and U the reversible transformation satisfying Liouville's theorem on page 681, meaning that U is volume-preserving. The ensemble is imagined to consist of points ω moving individually so that point ω is transformed to $U^t\omega$. For a cell Γ we find that $S_G(p_\Gamma) = (k \ln 2) \log L(\Gamma)$. That is, the Gibbs entropy is the same as the Boltzmann entropy.

What is the probability density of finding a state ω at time $t+t_0$? We call the new ensemble p^t. Liouville's theorem says that the volume L is invariant under the transformation U^t. This implies that $p^t(U^t\omega) = p(\omega)$, from which one can infer that $S_G(p^t) = S_G(p)$—the Gibbs entropy of an ensemble does not change at all in an isolated system during evolution. This shows that in the case of the evolution of isolated nonequilibrium systems, the evolution of a Gibbs ensemble does not express adequately what we consider thermodynamic behavior. The problem is that even if at the starting time t_0 the Gibbs ensemble was something simple, it can develop in time t into a very complicated density function that does not correspond to any reasonable macroscopic description. ◇

Example 8.5.12 If the ensemble is a discrete state space X with a probability density function $P(x)$, then the Gibbs entropy reduces to

$$S_G(P) = (k \ln 2) \sum_{x \in X} P(x) \log \frac{1}{P(x)}.$$

This is proportional to Shannon's information-theoretic entropy $H(P) = \sum_{x \in X} P(x) \log 1/P(x)$ of Definition 1.11.1 on page 67. ◇

8.6 Entropy Revisited

The previous approaches treated entropy as a probabilistic notion; in particular, each individual micro state of a system has entropy equal to 0. However, it is desirable to find a notion of entropy that assigns a nonzero entropy to each micro state of the system, as a measure of its individual disorder.

We will first give a naive form. Then we give a more sophisticated algorithmic form possessing properties similar to Boltzmann entropy, and in which a version of the second law can be proved.

Assume that the set of micro states of a thermodynamic system is discrete. For convenience we identify them with the natural numbers.

Definition 8.6.1 The *complexity entropy* of a micro state x of a system is defined as $K(x)$, where K is the prefix complexity of Chapter 3.

A single regular micro state like the state of CuO_2 in Figure 8.14 has low complexity entropy, as opposed to a complex state. Such a micro state looks regular and therefore special, only because it has low complexity. Such nonrandom states have complexity entropy close to zero.

The new definition can even be justified to a certain degree by Theorem 8.1.1 on page 604. Let X be the set of micro states of a thermodynamic system. We prove that a discrete version of Gibbs entropy and the expected complexity entropy are quantitatively about the same under some mild assumptions. Consider a discrete ensemble X, with $P(x)$ the probability density function that the system is in micro state $x \in X$. Then, $S_G(P) = (k \ln 2) \sum_{x \in X} P(x) \log 1/P(x)$ is the Gibbs entropy. If P is computable, then by Theorem 8.1.1 on page 604 we have

$$S_G(P) = (k \ln 2) \sum_{x \in X} P(x)K(x) + O(1).$$

In this sense, the complexity entropy inherits the successes and failures of Gibbs entropy. A serious problem is the choice of precision in describing micro state x. Our system will certainly have simpler micro states if we have units on the scale of 1 kilometer, rather than of scale 5×10^{-11} meters (the diameter of a hydrogen atom).

8.6.1 Algorithmic Entropy

In the previous approaches the macroscopic parameters were not accounted for and appeared out of the blue. Considering a system from an observer's viewpoint occasions a modified approach. Suppose we have determined the macroscopic parameters describing the classical entropy of a thermodynamic system. Truncate these parameters to the required precision (e.g., four decimal places) and encode them in an integer x.

Definition 8.6.2 Assume the discussion above, and suppose that the system in equilibrium is described by the encoding x of the approximated macroscopic parameters. The *algorithmic entropy* of the macro state of a system is given by $K(x) + H_x$, where $K(x)$ is the prefix complexity of x, and $H_x = S_B(x)/(k \ln 2)$. Here $S_B(x)$ is the Boltzmann entropy of the system constrained by macroscopic parameters x, and k is Boltzmann's constant. The physical version of algorithmic entropy is defined as $S_A(x) = (k \ln 2)(K(x) + H_x)$.

The second term, H_x, denotes our ignorance about the micro state, given x. The notion of algorithmic entropy reflects the fact that measurements can increase our knowledge about a system. Initially, we have no knowledge about the state of system, and therefore the algorithmic entropy reduces to Boltzmann entropy.

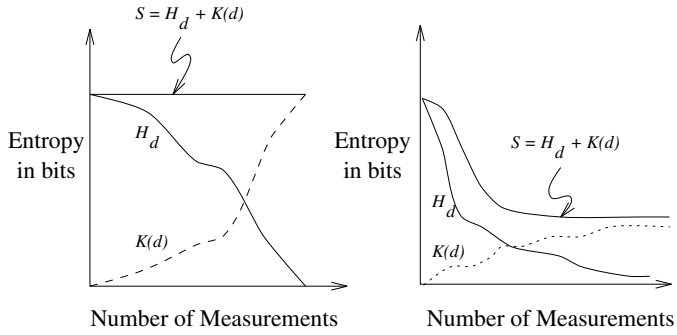


FIGURE 8.17. Algorithmic entropy: left a random micro state, right a regular micro state

If the system is in a nonrandom micro state ω, then the H_x term decreases significantly as we make more and more measurements, while the $K(x)$ term rises very slowly. Overall, the algorithmic entropy decreases. If a micro state ω is random, then we cannot achieve decrease of algorithmic entropy by making more and more measurements. We simply exchange decreased ignorance in H_x for increased knowledge in $K(x)$, for x is an increasing initial segment of ω. The two pictures in Figure 8.17 describe these situations.

Example 8.6.1 **(Maxwell's demon)** In 1867, J.C. Maxwell described in a letter a thought experiment that has received widespread attention ever since:

"If we conceive a being whose faculties are so sharpened that he can follow every molecule in its course, such a being, whose attributes are still as essentially finite as our own, would be able to do what is at present impossible to us. For we have seen that the molecules in a vessel full of air at uniform temperature are moving with velocities by no means uniform [...]. Now let us suppose that such a vessel is divided into two portions, A and B, by a division in which there is a small hole, and that a being, who can see the individual molecule, opens and closes this hole, so as to allow only the swifter molecules to pass from A to B and only the slower ones to pass from B to A. He will thus, without expenditure of work, raise the temperature of B and lower that of A, in contradiction to the second law of thermodynamics." [J.C. Maxwell, *Theory of Heat*, 1871.]

Several solutions to this problem have been proposed and subsequently been disposed of. L. Szilard in 1929 suggested that the required information gathering by the demon saves the second law; and C.H. Bennett in 1982 proposed the current interpretation that it is the destruction of old information that must dissipate energy.

In Figure 8.18 we describe this version of Maxwell's demon: the Szilard engine. The engine runs on a one-molecule gas and is submerged in a

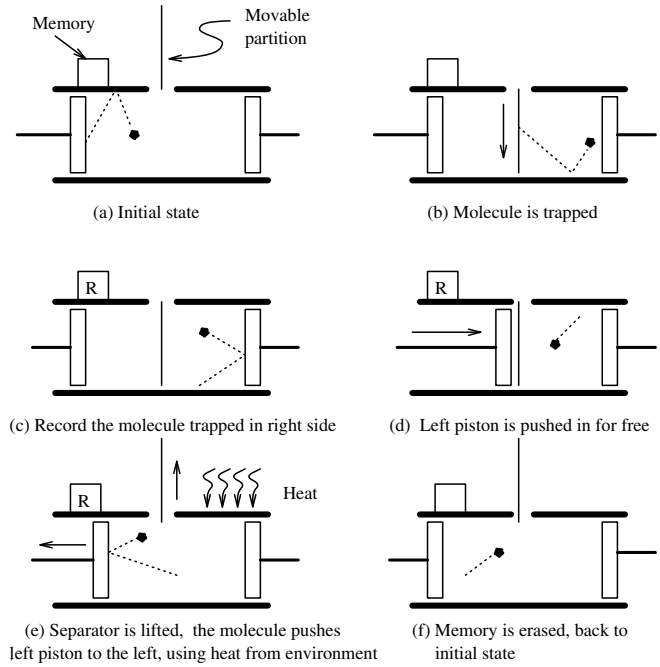


FIGURE 8.18. Szilard engine

heat bath at temperature T. The initial state of the system is that the molecule is located in either the left or the right chamber of the device and the trap door is up. The demon lowers the trap door, trapping the molecule in either the left or right chamber. The demon records where it is in its memory. It then pushes in the frictionless and massless piston in the chamber that does not contain the molecule. This does not cost work, since there is no pressure (bouncing molecule) to overcome. Since the engine is submerged in a heat bath, the molecule bounces around vigorously. If we raise the trap door, then the molecule will push the piston back to its original position, performing work. At the end of the cycle, we erase memory to resume the initial state.

Common objections against the demon include the idea that opening and closing the door must dissipate energy; the measurement of the molecule's position must dissipate energy (for example, the demon must send at least one photon to the molecule in order to see it), and so on. Such arguments have been found invalid, at least from the theoretical point of view.

However, the demon operating the engine must either store increasingly more information, or it must irreversibly erase some of it. Irreversible erasure of information will dissipate energy by Sections 8.2.1 and 8.2.4.

If the demon does not erase its memory each cycle, but keeps on adding new information, then this is not a true cycle. The engine increases the entropy of its memory in order to decrease the entropy of its environment.

The demon can, of course, wait for a while, accumulating information, and then try to guess the shortest program that generates it, and subsequently erase all information by irreversibly erasing only the shortest program for it.

We can obtain some insight into what happens using the algorithmic entropy of Definition 8.6.2.

For the sake of the present discussion we make the simplifying assumption that the system consisting of the engine coupled with the computerized demon has entropy $S_A = (k \ln 2)(K(x) + H_y)$, where k is Boltzmann's constant, $K(x)$ is the complexity of the data in the demon's memory, and $S_B(y) = (k \ln 2)H_y$ is the Boltzmann entropy of the engine. That is, the demon has no Boltzmann entropy part and the engine has no complexity part.

We make another important assumption by setting the total work performed to the sum of the work needed to change the entropy of the machine and the work needed to change the entropy of the demon. This seems a possibly unrealistic assumption, since it is not obvious that the joint effect cannot be achieved with less work. Think of the triangle inequality. The work to pull something north 1 meter and east 1 meter is more than the work needed to pull it to its final place in a straight line.

Claim 8.6.1 Assume that the entropy of the system behaves as described above. Then the net heat gained by the engine, coupled with a computerized demon that can perform measurements and control the operation of the engine, is no more than

$$\Delta Q = (S_A^f - S_A^i)T,$$

where T is the temperature in kelvins, and S_A^i and S_A^f are the initial and final algorithmic entropies of the system, respectively.

The idea is as follows: The system is operating at temperature T. Then the heat gained due to the change of Boltzmann entropy, denoted by S_B, is given by the usual formula

$$\Delta Q^+ = (S_B^f - S_B^i)T.$$

Let $K(i)$ and $K(f)$ be the complexity of the demon's initial memory state i and final memory state f, respectively. In Section 8.2 we have analyzed the absolute minimum number of bits that need to be erased in

a computation from i to f, leaving nothing behind except the record f. This is at least the number of garbage bits left after computing f from i by a reversible machine. These garbage bits constitute a program to compute f from i (executing the computation from i to f in reverse). The length of a self-delimiting program for this purpose is by definition $K(i|f)$. By Theorem 3.9.1 on page 247, up to an additive constant,

$$K(i,f) = K(i) + K(f|\langle i, K(i)\rangle) = K(f) + K(i|\langle f, K(f)\rangle).$$

Trivially, $K(i|f) \geq K(i|\langle f, K(f)\rangle) + \mathrm{O}(1)$. Therefore, $K(i|f) \geq K(i) - K(f + O(1))$. (See also Section 8.2.4.) The heat loss to update the demon's memory is the negative of the number of erased bits:

$$\Delta Q^- = (K(f) - K(i))kT \ln 2.$$

Thus, the total heat ΔQ gained by the demon is given by

$$\begin{aligned}\Delta Q^+ + \Delta Q^- &= \left[(S_B^f + (k \ln 2)K(f)) - (S_B^i + (k \ln 2)K(i))\right] T \\ &= (S_A^f - S_A^i)T.\end{aligned}$$

With S_A subject to the second law, ΔQ in the above claim must be less than or equal to 0. ◇

8.6.2 Algorithmic Entropy and Randomness Tests

Algorithmic entropy of the physics variety, Definition 8.6.2 on page 687, can be based roughly on the profound idea of the universal randomness test as developed in Section 2.5 and especially Section 4.5. We will call the resulting mathematical notion 'algorithmic entropy' also.

Consider the molecules of an ideal gas as points that move in 3-dimensional space $\mathcal{R}^3$, where $\mathcal{R}$ is the set of real numbers, and use the model of statistical mechanics as described in Definition 8.5.1 on page 679 and the following pages.

Definition 8.6.3 Let x be a binary string of length n. An *n-cell* is the set Γ_x of all infinite binary sequences (micro states) with finite initial segment x, defined by $\Gamma_x = \{\omega : \omega = x\zeta,\ \zeta \in \{0,1\}^\infty\}$.

The sets Γ_x are the well-known cylinder sets of measure theory (Section 1.6). The area of state space we consider is divided into n-cells. For $n = 1, 2, \ldots$, this division becomes progressively finer-grained and discerns more detail. To say that a micro state ω is in n-cell Γ_x means that x is a prefix of ω.

In fact, ω is the limit point to which the infinite nested sequence of n-cells that contain ω converges with increasing n.

Definition 8.6.4 Assume the notation above and recall Notation 8.5.1 on page 680. The *algorithmic entropy* of an n-cell Γ_x with respect to μ is defined as

$$H_\mu(\Gamma_x) = K(x|\mu) + \log \mu(\Gamma_x).$$

In terms of Definition 8.6.2, page 687, the algorithmic entropy of an n-cell Γ_x consists of two parts, where the first part is the prefix complexity of x (the macroscopic description of the n-cell) and the second part is the logarithm of the measure of volume Γ_x (our lack of knowledge about a member of the n-cell). This definition can serve to define successively closer approximations to the entropy of a micro state as the minimum of the entropies of successively smaller n-cells containing it.

For computable measures μ, we have $K(x|\mu) \geq K(x) - K(\mu)$ for all x, with $K(\mu)$ a constant independent of x that is the length of the shortest program to compute μ. Since by definition $K(x|\mu) \leq K(x) + O(1)$, we have $K(x|\mu) = K(x) \pm O(K(\mu))$.

Definition 8.6.5 Let μ be a computable measure. The *algorithmic entropy* of $\omega \in X$ with respect to μ is defined by

$$H_\mu(\omega) = \inf_n \left\{ K(\omega_{1:n}|\mu) + \log \mu(\Gamma_{\omega_{1:n}}) \right\}.$$

This definition sets $H_\mu(\omega)$ equal to $-\rho_0(\omega|\mu)$, the universal integral μ-test of Corollary 4.5.2 on page 315. (Recall Notation 4.2.1.) For the special case that μ is the uniform measure, see Corollary 3.6.1 on page 223.

Example 8.6.2 **(Generalized prefix complexity)** Let $X = \{0, 1, \ldots\}$, and for every $x \in X$, set $\mu(\{x\}) = 1$. Then $H_\mu(x) = K(x) + O(1)$. Thus, the algorithmic entropy H_μ, defined by way of randomness tests, is a generalization of the prefix complexity K. ◇

Example 8.6.3 How large can $H_\mu(\omega)$ be? The higher it is, the more random is ω. For a finite measure μ ($\mu(X) \leq \infty$), the function H_μ is bounded above. Such a bound is the ultimate maximum, or equilibrium, of algorithmic entropy in such a system, and it is reached when the system has evolved to total disorder. For infinite measures, it can take arbitrarily large positive values; but it will never be $+\infty$. By the theory of testing, ω is random in the sense of Martin-Löf (Sections 2.5, 4.5.6) iff $-H_\mu(\omega) = \rho_0(\omega|\mu) < \infty$. In other words, a micro state ω is random iff $H_\mu(\omega) > -\infty$.

To give some idea how $H_\mu(\omega)$ depends on ω and μ we give an upper bound. Let $m(x) = \mu(\Gamma_x)$. For $\omega \in \Gamma_x - \Gamma_{x0}$, we have $H_\mu(\omega) < \log(m(x) + 1) + 2\log(\log(m(x) + 1) + 1) + O(1)$. The right-hand side is an upper bound on the amount by which $K(x|\mu)$ can exceed $\log \mu(\Gamma_x)$. ◇

Example 8.6.4 The sequence $\eta = 00\ldots$ is nonrandom in the uniform measure λ, and hence (Sections 2.5, 4.5.6) $\rho_0(\eta|\lambda) = \infty$. This means that the algorithmic entropy $H_\lambda(\eta)$ is $-\infty$. The average ω satisfies $\rho_0(\omega|\lambda) < \infty$, and therefore $H_\lambda(\omega) > -\infty$. (This is the case for all random sequences, and the set of these sequences has λ-measure 1, by Theorem 2.5.3 on page 151.)

But some $\eta \in \{0,1\}^\infty$ have $H_\lambda(\eta) = -\infty$. These are precisely the sequences that are not λ-random. ◇

Example 8.6.5 For some computable measures μ there are n-cells Γ_x such that all sequences in Γ_x are not μ-random. Namely, if $\mu(\Gamma_x) = 0$, then $\log \mu(\Gamma_x) = -\infty$, while $K(x|\mu)$ is finite. It follows from the definition of algorithmic entropy that then $H_\mu(\omega) = -\infty$ for all $\omega \in \Gamma_x$. (By the way, existence of such a Γ_x means that μ is not the uniform measure.)

If $H_\mu(\omega) = -\infty$, then the system is in an orderly state. For example, all molecules of a gas are in the left half of the containing vessel. The set of all such orderly states has zero probability in the sense of measure μ. ◇

Example 8.6.6 We show a connection between H_μ and the Gibbs entropy S_G. Consider an ensemble X with computable probability density function p and measure μ. Define $\nu(d\,\omega) = p(\omega)\mu(d\,\omega)$. Then it can be shown that

$$\frac{S_G(p)}{k \ln 2} = \int p(\omega) \log \frac{1}{p(\omega)} \mu(d\,\omega) = \int H_\mu(\omega)\nu(d\,\omega) + O(1).$$

This means that the Gibbs entropy is essentially the average algorithmic entropy, while the latter measures the randomness of ω with respect to μ. The higher it is, the more random is ω. ◇

The H_μ measure is important, but it does not satisfy the second law of thermodynamics, since the source reference paper of Exercise 8.6.1 on page 696 shows that it has a no strong increase property. We solve this problem by considering coarse-grained entropy.

Definition 8.6.6 Assume the above notation. The nth approximation of $H_\mu(\omega)$, the *coarse-grained algorithmic entropy*, is given by $H^n_\mu(\omega)$ defined as

$$H^n_\mu(\omega) = \inf_{i \le n} \left\{ H_\mu(\Gamma_{\omega_{1:i}}) \right\}.$$

Example 8.6.7 If the notion of temperature is meaningful in our system, then the physical version of algorithmic entropy is to be defined as

$$S_A(\omega) = (k \ln 2) H_\mu(\omega), \quad S^n_A(\omega) = (k \ln 2) H^n_\mu(\omega),$$

where k is the very small Boltzmann's constant. For appropriate fixed n, this new quantity $(k \ln 2) H^n_\mu(\cdot)$ is our corrected version of coarse-grained Bolzmann entropy $S_B(\cdot)$. ◇

The coarse-grained algorithmic entropy H^n_μ is important, since the source reference paper of Exercise 8.6.1 on page 696 shows that it satisfies the second law, by giving the required strong growth of entropy, similar to Boltzmann entropy. The quantity H^n_μ is mathematically objective and is related to the observation-dependent term H_x in Definition 8.6.2.

Notation 8.6.1 The subscript μ is usually understood from the context. We dispense with the subscript from now on, using it only when there is risk of confusion.

The relations between the algorithmic entropy of a micro state, the coarse-grained algorithmic entropy, and the n-cell containing it are as follows:

Theorem 8.6.1 *Assume the notation above. Then*

$$H(\omega) = \lim_{n\to\infty} H^n(\omega) = \inf_{x\in\{0,1\}^*} \{H(\Gamma_x) : \omega \in \Gamma_x\} \ . \tag{8.17}$$

Proof. This follows directly from the definitions. □

The formula for $H(\omega)$ says that to determine its value we should consider ever finer-grained n-cells (test more bits of $\omega_{1:n}$), but only as long as the complexity increase of $K(\cdot)$ buys us an even greater decrease in $\log \mu(\Gamma_{\omega_{1:n}})$. For ω that are not μ-random, the latter quantity goes down all the way to $-\infty$ with growing n.

For μ-random ω, the value of $H^n(\omega)$ goes down to at most a finite (possibly negative) value with n. Hence, there is a value n_0 such that $H^n(\omega)$ reaches its infimum for $n = n_0$. This is the optimal granularity of description for this micro state that yields the final algorithmic entropy. The term $\log \mu(\Gamma_x)$ represents our a priori uncertainty about ω as an element of its n-cell Γ_x. This corresponds to the second term, H_x, in Definition 8.6.2.

Example 8.6.8 For the systems of interest in physics, and for those precisions in the macroscopic parameters for which Boltzmann's entropy is generally considered, we expect the additive term $K(\omega_{1:n})$, which is at most $n+2\log n$, to be negligible compared to the second term of algorithmic entropy. This log-volume term is usually very large.

For example, consider a mole of gas (about 6×10^{23} molecules, Definition 8.5.1 on page 679) in a container. Let us be extremely generous and assume that each atom can assume only two states (or each of the $6N$ dimensions is of size 2 in the continuous case). Then the log-volume is about 10^{23} bits in order to describe the complete state of a few grams of matter. This is the order of magnitude of the total entropy.

The macroscopic information given to us in $\omega_{1:n}$ is generally just the values of some macroscopic parameters. We may, for example, subdivide the piece of magnet under study into 100 parts and measure the magnetization of each to an unlikely precision of 10 bits: this is still only 1000 bits out of 10^{23} bits. Therefore, $K(\omega_{1:n})$ is negligible compared to the second log-volume term in such cases. Hence, $H(\omega)$ is not appreciably larger than its log-volume term when expressed this way, and can be interpreted as an approximation to the Boltzmann entropy. ◇

Example 8.6.9 Often, the first term $K(\cdot|\mu)$ of the algorithmic entropy $\inf_n\{K(\omega_{1:n}|\mu) + \log\mu(\Gamma_{\omega_{1:n}})\}$ is insignificant compared to the second term, which measures the Boltzmann entropy. An example of a system for which also the complexity term of algorithmic entropy is needed is a computer memory.

The memory cells are not individual atoms but they are rather tiny. The total number of bits in a modern computer is already measured in gigabytes. It is natural to view all the information in the computer memory as macroscopic when we describe this system. However, as the memory cells get smaller, the boundary between macroscopic and microscopic information becomes somewhat blurred, and when describing the whole system one will probably want to use the whole formula. All memory states of the computer would have about the same classical Boltzmann entropy, but more complex memory states would have larger algorithmic entropy. Thus, the tendency of noise to increase the complexity of memory could be considered the same phenomenon as the entropy increase in classical physical systems.

Such computer systems are typical of the systems considered by Landauer. Interpret "erasing a bit," done by irreversible computation as in Section 8.2, as decreasing the entropy by 1. Suppose that all of the memory together contains the string x. Then we may say that $H(\omega) = K(x) + \log\mu(\Gamma_x)$, where $\omega \in \Gamma_x$, and Γ_x is a subset of the memory states that can be described by saying what bits are in the memory. This means that $K(\Gamma_x) = K(x)$.

Erasing the information means putting 0's everywhere into the memory. Denote the micro state of the environment by ζ. Then the development of the system consisting of memory and environment in time is denoted by $U^t(\omega,\zeta) = (\omega^t,\zeta^t)$. Arguably, we will have $H(\omega^t) = K(y) + \log\mu(\Gamma_y)$, where $\omega^t \in \Gamma_y$. Here, Γ_y just says that the memory contains all 0's; hence $K(\Gamma_y) = O(1)$.

If we require that the memory have approximately the same amount of Boltzmann entropy $\log\mu(\Gamma_x)$ no matter what bits it holds, then this means that $\log\mu(\Gamma_x) = \log\mu(\Gamma_y)$. In this interpretation, the erasure indeed means $\Delta H(\omega) = K(x)$, which will be balanced by a similar increase in $H(\zeta)$.

In order to argue, on the basis of Boltzmann entropy alone, that turning all these bits to 0 results in heat dissipation, we need to consider this operation of turning all bits to 0 as some kind of general operation that does something similar with all possible memory contents, that is, decreases the volume of the whole state space. This artificiality is avoided here. ◇

Exercises

8.6.1. [33] Let $n = l(x)$. Show that for some constant c,

$$\mu\{\omega \in \Gamma_x : H(\omega) < H(\Gamma_x) - K(n) - m\} < c\, 2^{-m}\mu(\Gamma_x).$$

Comments. For most elements ω of an n-cell Γ_x, the value of $H(\omega)$ cannot be much less than $H(\Gamma_x)$. That is, if some elements of a cell are (sufficiently) random, then most of them are (sufficiently) random. Recall that $K(n) \leq \log n + 2\log\log n + O(1)$, for all n. Source: P. Gács, *Proc. IEEE Workshop Physics and Computation*, Dallas, USA, 1994, 209–216.

8.7 Quantum Kolmogorov Complexity

Quantum information theory, the quantum-mechanical analogue of classical information theory, is experiencing a renaissance due to the rising interest in the notion of quantum computation and the possibility of realizing a quantum computer. While Kolmogorov complexity is the accepted absolute measure of information content in an individual classical finite object, a similar absolute notion is needed for the information content of an individual pure quantum state. We base quantum Kolmogorov complexity on quantum Turing machines.

A pure quantum state ϕ, represented as a unit-length vector in a Hilbert space, is denoted by $|\phi\rangle$, and the corresponding element of the dual space (the conjugate transpose) is written as $\phi^\dagger$ or $\langle\phi|$. The inner product of $\langle\phi|$ and $|\psi\rangle$ is written in physics notation as $\langle\phi \mid \psi\rangle$ and in mathematics notation as $\phi^\dagger\psi$. The bra-ket notation is due to P. Dirac and is the standard quantum-mechanics notation. The bra $\langle x|$ denotes a row vector with complex entries, and the ket $|x\rangle$ is the column vector consisting of the conjugate transpose of $\langle x|$ (columns interchanged with rows and the imaginary part of the entries negated, that is, $\sqrt{-1}$ is replaced by $-\sqrt{-1}$).

For every N, the finite-dimensional Hilbert space $\mathcal{H}_N$ has a canonical basis $|e_0\rangle, \ldots, |e_{N-1}\rangle$. Assume that the canonical basis of $\mathcal{H}_N$ is also the beginning of the canonical basis of $\mathcal{H}_{N+1}$. The m-fold tensor product $\otimes_{i=1}^m \mathcal{H}$ of a Hilbert space $\mathcal{H}$ is denoted by $\mathcal{H}^{\otimes m}$.

Of special importance is the two-dimensional Hilbert space $\mathcal{C}^2$, where $\mathcal{C}$ is the set of complex real numbers, and $|0\rangle$, $|1\rangle$ is its canonical orthonormal

basis. An element of $\mathcal{C}^2$ is called a qubit ('quantum bit' in analogy with an element of $\{0,1\}$ which is called a bit for 'binary digit'). To generalize this to strings of n qubits, we consider the quantum state space $\mathcal{C}^N$ with $N = 2^n$. The basis vectors $|e_0\rangle, \ldots, |e_{N-1}\rangle$ of this space are parametrized by binary strings of length n, so that $|e_0\rangle$ is shorthand for $|e_{0\ldots0}\rangle$ and $|e_{N-1}\rangle$ is shorthand for $|e_{1\ldots1}\rangle$. Mathematically, $\mathcal{C}^N$ is decomposed into a tensor product of n copies of $\mathcal{C}^2$, written as $(\mathcal{C}^2)^{\otimes n}$, and an n-qubit state $|a_1 \ldots a_n\rangle$ in bra-ket notation can also be written as the tensor product $|a_1\rangle \otimes \cdots \otimes |a_n\rangle$, or shorthand as $|a_1\rangle \cdots |a_n\rangle$, a string of n qubits, the qubits being distinguished by position.

8.7.1 Quantum Computation

A quantum Turing machine can be viewed as a generalization of a probabilistic Turing machine. Consider the same computation tree where every node has two children.. In the probabilistic computation there is a probability $p_i \geq 0$ associated with each node i (state of the system) at the same level in the tree, such that $\sum p_i = 1$, summed over the nodes at the same level. In a quantum-mechanical computation there is a *probability amplitude* α_i associated with each basis state $|i\rangle$ of the system. In the probabilistic case the states of the nodes i at the mth level of the computation tree run through the classical values 0 through $2^m - 1$. In the quantum case the states are represented by the orthonormal basis m-qubit states $|00\ldots0\rangle$ through $|11\ldots1\rangle$. The nodes at level m are in a superposition $|\psi\rangle = \sum_{i\in\{0,1\}^m} \alpha_i |i\rangle$ with the probability amplitudes satisfying $\sum_{i\in\{0,1\}^m} ||\alpha_i||^2 = 1$.

Generally, the probability amplitudes are complex numbers satisfying $\sum ||\alpha_i||^2 = 1$, where if $\alpha_i = a + b\sqrt{-1}$ then $||\alpha_i|| = \sqrt{a^2 + b^2}$, and the summation is taken over all distinct states of the observable at a particular instant. We say distinct states since the quantum-mechanical calculus dictates that equal states be grouped together: If state $|\phi\rangle$ of probability amplitude α equals state $|\psi\rangle$ of probability amplitude β, then their combined contribution in the sum is $||\alpha + \beta||^2 |\phi\rangle$. The transitions are governed by a matrix U that represents the program being executed. Such a program has to satisfy the following constraints. Denote the set of possible configurations of the Turing machine by X, where X is the set of m-bit column vectors (the basis states) for simplicity. Then U maps the column vector $\underline{\alpha} = (\alpha_x)_{x\in X}$ to $U\underline{\alpha}$. Here $\underline{\alpha}$ is a (2^m)-element complex vector of amplitudes of the quantum superposition of the 2^m basis states before the step, and $U\underline{\alpha}$ the same after the step concerned. The special property that U needs to satisfy in quantum mechanics is that it be *unitary*, that is, $U^\dagger U = I$, where I is the identity matrix and $U^\dagger$ is the conjugate transpose of U (as with the bra-ket, 'conjugate' means that all $\sqrt{-1}$'s are replaced by $-\sqrt{-1}$'s, and 'transpose' means that the rows and columns are interchanged). In other words, U is unitary iff $U^\dagger = U^{-1}$.

The unitary constraint on the evolution of the computation enforces two facts:

1. If $U^0\underline{\alpha} = \underline{\alpha}$ and $U^t = UU^{t-1}$ then $\sum_{x\in X} ||(U^t\underline{\alpha})_x||^2 = 1$ for all t (discretizing time for convenience).
2. A quantum computation is reversible (replace U by $U^\dagger = U^{-1}$). In particular, this means that a computation $U^t\underline{\alpha_0} = \underline{\alpha_t}$ is undone by running the computation stepwise in reverse: ${U^\dagger}^t\underline{\alpha_t} = \underline{\alpha_0}$.

Definition 8.7.1 A *qubit* $|\psi\rangle$ is the quantum-mechanical version of a single classical bit, defined by $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$ such that $||\alpha||^2 + ||\beta||^2 = 1$.

A qubit consists of partially the basis state $|0\rangle$ and partially the basis state $|1\rangle$. The states are denoted by the column vectors of the appropriate complex probability amplitudes. For the basis states the vector notations are $|0\rangle = \binom{1}{0}$ (that is, $\alpha = 1$ and $\beta = 0$), and $|1\rangle = \binom{0}{1}$ (that is, $\alpha = 0$ and $\beta = 1$). We also write $|\phi\rangle$ as the column vector $\underline{\phi} = \binom{\alpha}{\beta}$.

Example 8.7.1 Physically, for example, the state $|\psi\rangle$ can be the state of a polarized photon, and the basis states are horizontal or vertical polarization, respectively. Upon measuring according to the basis states, that is, passing the photon through a medium that is polarized either in the horizontal or vertical orientation, the photon is observed with probability $||\alpha||^2$ or probability $||\beta||^2$, respectively. Consider a sample computation on a one-bit computer executing the unitary operator

$$S = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix}. \tag{8.18}$$

It is easy to verify, using common matrix calculation, that

$$S|0\rangle = \frac{1}{\sqrt{2}}\,|0\rangle - \frac{1}{\sqrt{2}}\,|1\rangle,\ S|1\rangle = \frac{1}{\sqrt{2}}\,|0\rangle + \frac{1}{\sqrt{2}}\,|1\rangle,$$
$$S^2|0\rangle = 0\,|0\rangle - 1\,|1\rangle = -|1\rangle,\ S^2|1\rangle = 1\,|0\rangle + 0\,|1\rangle = |0\rangle.$$

If we observe the computer in state $S|0\rangle$, then the probability of observing state $|0\rangle$ is $(\frac{1}{\sqrt{2}})^2 = \frac{1}{2}$, and the probability to observe $|1\rangle$ is $(-\frac{1}{\sqrt{2}})^2 = \frac{1}{2}$. However, if we observe the computer in state $S^2|0\rangle$, then the probability of observing state $|0\rangle$ is 0, and the probability to observe $|1\rangle$ is 1. Similarly, if we observe the computer in state $S|1\rangle$, then the probability of observing state $|0\rangle$ is $(\frac{1}{\sqrt{2}})^2 = \frac{1}{2}$, and the probability to observe $|1\rangle$ is $(\frac{1}{\sqrt{2}})^2 = \frac{1}{2}$. If we observe the computer in state $S^2|1\rangle$, then

the probability of observing state $|0\rangle$ is 1, and the probability to observe $|1\rangle$ is 0. Therefore, the operator S inverts a bit when it is applied twice in a row, and hence has acquired the charming name *square root of 'not'*.

In contrast, with the analogous probabilistic calculation, flipping a coin two times in a row, we would have found that the probability of each computation path in the complete binary computation tree of depth 2 was $\frac{1}{4}$, and the states at the four leaves of the tree were $|0\rangle, |1\rangle, |0\rangle, |1\rangle$, resulting in a total probability of observing $|0\rangle$ being $\frac{1}{2}$, and the total probability of observing $|1\rangle$ being $\frac{1}{2}$ as well.

The quantum principle involved in the above example is called *interference*, similar to the related light phenomenon in the seminal two-slit experiment: Shining light through a screen with *two* small holes, we observe a diffraction pattern of bright and dark stripes due to interference. Namely, the light hits every point on the screen via two different routes (through the two different holes). If the two routes differ by an even number of half wavelengths, then the wave amplitudes at the target are added, resulting in twice the amplitude and a bright spot, and if they differ by an odd number of half wavelengths then the wave amplitudes are in opposite phase and are subtracted, resulting in zero and a dark spot. ◇

We are now in a position to explain the quantum equivalent of a probabilistic coin flip. This is a main trick for enhancing the power of quantum computation. A sequence of n fair coin flips corresponds to a sequence H_n of n one-qubit unitary operations, the Hadamard transform

$$H \;=\; \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

on the successive bits of a register of n bits originally in the all-0 state $|\psi\rangle = |00\ldots0\rangle$. The result is a superposition

$$H_n|\psi\rangle = \sum_{x\in\{0,1\}^n} 2^{-n/2}\ |x\rangle$$

of all the 2^n possible states of the register, each with amplitude $2^{-n/2}$ (and hence probability of being observed of 2^{-n}).

The Hadamard transform is ubiquitous in quantum computing; its singlefold action is similar to that of the transform S of Equation 8.18 with the roles of 0 and 1 partly interchanged. In contrast to S^2, which implements the logical 'not,' we have $H^2 = I$ with I the identity matrix.

Example 8.7.2 If A is a classical algorithm for computing some function f, possibly even irreversible such as $f(x) \equiv x \bmod 2$, then we can turn it into a

unitary transformation that maps classical state $|x, 0\rangle$ to $|x, f(x)\rangle$. Note that we can apply A to a superposition of all 2^n inputs:

$$A\left(2^{-n/2}\sum_x |x,0\rangle\right) = 2^{-n/2}\sum_x |x, f(x)\rangle.$$

In some sense this state contains the results of computing f for *all* possible inputs x, but we have applied A only once to obtain it. This effect together with the interference phenomenon is responsible for one of the advantages of quantum over classical randomized computing and is called *quantum parallelism*. ◇

8.7.2 Quantum Turing Machine

A quantum Turing machine is a Turing machine operating on strings of quantum bits, using specific quantum-mechanical operations like those described above. To define quantum Kolmogorov complexity by way of quantum Turing machines leaves essentially two options:

1. We want to describe every quantum superposition exactly; or
2. we want to take into account the number of bits/qubits in the specification as well the accuracy of the quantum state produced.

We have to deal with three problems:

- There are continuously many qubit descriptions.
- There are continuously many quantum Turing machines;
- There are continuously many pure quantum states;

Definition 8.7.2 The *quantum Turing machine* is equipped with an input tape that is one-way infinite with the classical input (the program) in binary left adjusted from the beginning. The input tape is read-only from left to right without backing up. Additionally, the machine contains a one-way infinite work tape containing qubits, a one-way infinite auxiliary tape containing qubits, and a one-way infinite output tape containing qubits. Initially, the input tape contains a classical binary program p, and all (qu)bits of the work tape, auxiliary tape, and output tape are set to $|0\rangle$. In case the Turing machine has an auxiliary input (classical or quantum) then initially the leftmost qubits of the auxiliary tape contain this input. A quantum Turing machine Q with classical program p and auxiliary input y computes until it halts with output $Q(p, y)$ on its output tape or it computes forever.

Since quantum Turing machines are reversible, this means that there must be an ongoing evolution with nonrepeating configurations.

Definition 8.7.3 Define the *output* $Q(p, y)$ of a quantum Turing machine Q with classical program p and auxiliary input y as the pure quantum state $|\psi\rangle$ resulting from Q computing until it halts with output $|\psi\rangle$ on its ouput tape. We write $Q(p, y) < \infty$. The halting convention is a complex issue, which we ignore here. If there is no such $|\psi\rangle$ then $Q(p, y)$ is undefined and we write $Q(p, y) = \infty$. By definition the input tape is read-only from left to right without backing up. Therefore, the set of *halting programs* $\mathcal{P}_y = \{p : Q(p, y) < \infty\}$ is *prefix-free*: no program in $\mathcal{P}_y$ is a proper prefix of another program in $\mathcal{P}_y$. Put differently, the Turing machine scans all of a halting program p but never scans the bit following the last bit of p: it is *self-delimiting*.

There are uncountably many quantum Turing machines only if we allow arbitrary quantum-mechanical operations (these are called rotations) in the definition of the machines. Then, a quantum Turing machine can be universal only in the sense that it can approximate the computation of an arbitrary machine. In descriptions using universal quantum Turing machines we would have to account for the closeness of approximation, the number of steps required to get this precision, and the like. In contrast, if we fix the rotation of all contemplated machines to a single primitive rotation θ with $\cos\theta = \frac{3}{5}$ and $\sin\theta = \frac{4}{5}$, then there are only countably many Turing machines, and the universal machine simulates the others exactly. Exact simulation of arbitrary quantum Turing machines by a fixed universal quantum Turing machine is impossible, since there are continuously many of the former. Close approximations are possible by Turing machines using a fixed rotation such as θ, Exercise 8.7.3 on page 709. There are only countably many such Turing machines. Using a standard ordering, we fix $Q_1, Q_2, \ldots$ as a standard enumeration of quantum Turing machines using only rotation θ. There is a universal machine U in this enumeration that simulates the others exactly: $U(1^i 0 p, y) = Q_i(p, y)$, for all i, p, y.

Since the computation time of the machine is not limited in the theory of description complexity as developed here, a quantum computer can be simulated by a classical computer to every desired degree of precision. We can rephrase everything in terms of the standard enumeration of $T_1, T_2, \ldots$ of classical Turing machines. Let $|x\rangle = \sum_{i=0}^{N-1} \alpha_i |e_i\rangle$ $(N = 2^n)$ be an n-qubit state. We can write $T(p) = |x\rangle$ if T either outputs

(i) algebraic definitions of the coefficients of $|x\rangle$ (in case these are algebraic), or

(ii) a sequence of approximations $(\alpha_{0,k}, \ldots, \alpha_{N-1,k})$ for $k = 1, 2, \ldots$, where $\alpha_{i,k}$ is an algebraic approximation of α_i to within 2^{-k}.

8.7.3 Classical Descriptions of Pure Quantum States

The next question is whether we want programs (descriptions) to be in classical bits or in qubits. The intuitive notion of computability requires the programs to be classical. Namely, to prepare a quantum state requires a physical apparatus that computes this quantum state from classical specifications. Since such specifications have effective descriptions, every quantum state that can be prepared can be described effectively in descriptions consisting of classical bits. Descriptions consisting of arbitrary pure quantum states allows incomputable (or hard to compute) information to be hidden in the bits of the amplitudes. In Definition 8.7.5 we call a pure quantum state *directly computable* if there is a (classical) program such that the universal quantum Turing machine computes that state from the program and then halts in an appropriate fashion.

The complex quantity $\langle x \mid z \rangle$ is the inner product of vectors $\langle x|$ and $|z\rangle$. Since pure quantum states $|x\rangle, |z\rangle$ have unit length, $||\langle x \mid z\rangle|| = |\cos\theta|$, where θ is the angle between vectors $|x\rangle$ and $|z\rangle$. The quantity $||\langle x \mid z\rangle||^2$, the *fidelity* between $|x\rangle$ and $|z\rangle$, is a measure of how close or confusable the vectors $|x\rangle$ and $|z\rangle$ are. It is the probability of outcome $|x\rangle$ being measured from state $|z\rangle$ that is actually there. Essentially, we project $|z\rangle$ on outcome $|x\rangle$ using projection $|x\rangle\langle x|$, resulting in $\langle x \mid z\rangle|x\rangle$.

Definition 8.7.4 The (*self-delimiting*) *complexity* of $|x\rangle$ with respect to quantum Turing machine Q with y as conditional input given for free is

$$KQ_Q(|x\rangle|y) = \min_p \left\{ l(p) + \left\lceil \log \frac{1}{||\langle z \mid x\rangle||^2} \right\rceil : Q(p,y) = |z\rangle \right\}, \quad (8.19)$$

where $l(p)$ is the number of bits in the program p, auxiliary y is an input (possibly quantum) state, and $|x\rangle$ is the target state that one is trying to describe.

Definition 8.7.5 A pure quantum state $|x\rangle$ is *computable* if $KQ(|x\rangle) < \infty$. Hence all finite-dimensional pure quantum states are computable. We call a pure quantum state *directly computable* if there is a program p such that $U(p) = |x\rangle$.

Example 8.7.3 Note that $|z\rangle$ is the quantum state produced by the computation $Q(p, y)$, and therefore, given Q and y, completely determined by p. Therefore, we obtain the minimum of the right-hand side of the equality by minimizing over p only. We call the $|z\rangle$ that minimizes the right-hand side the *directly computed part* of $|x\rangle$, while $\lceil \log 1/||\langle z \mid x\rangle||^2 \rceil$ is the *approximation part*. The quantity $P(x) = ||\langle z \mid x\rangle||^2$ is the probability that $|z\rangle$ passes a test for $|x\rangle$, and vice versa. The term $\lceil \log 1/||\langle z \mid x\rangle||^2 \rceil$ can be viewed as the code-word length to redescribe $|x\rangle$, given $|z\rangle$ and an orthonormal

basis with $|x\rangle$ as one of the basis vectors, using the Shannon–Fano prefix code. This works as follows: Write $N = 2^n$. For every state $|z\rangle$ in N-dimensional Hilbert space with basis vectors $\mathcal{B} = \{|e_0\rangle, \ldots, |e_{N-1}\rangle\}$ we have $\sum_{i=0}^{N-1} ||\langle e_i \mid z\rangle||^2 = 1$. If the basis has $|x\rangle$ as one of the basis vectors, then we can consider $|z\rangle$ as a random variable that assumes value $|x\rangle$ with probability $||\langle x \mid z\rangle||^2$. The Shannon–Fano code word for $|x\rangle$ in the probabilistic ensemble $\left(\mathcal{B}, (||\langle e_i \mid z\rangle||^2)_i\right)$ is based on the probability $||\langle x \mid z\rangle||^2$ of $|x\rangle$, given $|z\rangle$, and has length $\lceil \log 1/||\langle x \mid z\rangle||^2 \rceil$. Considering a canonical method of constructing an orthonormal basis $\mathcal{B} = |e_0\rangle, \ldots, |e_{N-1}\rangle$ from a given basis vector, we can choose $\mathcal{B}$ such that $KQ(\mathcal{B}) = \min_i\{KQ(|e_i\rangle)\} + O(1)$. The Shannon–Fano code is appropriate for our purpose, since it is optimal in that it achieves the least expected code-word length—the expectation taken over the probability of the source words—up to 1 bit by the noiseless coding theorem, Theorem 1.11.2. As in the classical case, the quantum Kolmogorov complexity is an integral number, and there is an invariance theorem. ◇

Theorem 8.7.1 *There is a universal quantum Turing machine U such that for every quantum Turing machine Q, there is a constant c_Q (the length of the description of the index of Q in the enumeration) such that for all quantum states $|x\rangle$ and all auxiliary inputs y we have*

$$KQ_U(|x\rangle|y) \leq KQ_Q(|x\rangle|y) + c_Q.$$

Proof. Assume that the program p that minimizes the right-hand side of (8.19) is p_0 and the computed $|z\rangle$ is $|z_0\rangle$:

$$KQ_Q(|x\rangle|y) = l(p_0) + \left\lceil \log \frac{1}{||\langle z_0 \mid x\rangle||^2} \right\rceil.$$

There is a universal quantum Turing machine U in the standard enumeration $Q_1, Q_2, \ldots$ such that for every quantum Turing machine Q in the enumeration there is a self-delimiting program i_Q (the index of Q) and $U(i_Q p, y) = Q(p, y)$ for all p, y: if $Q(p, y) = |z\rangle$ then $U(i_Q p, y) = |z\rangle$. In particular, this holds for p_0 such that Q with auxiliary input y halts with output $|z_0\rangle$. But U with auxiliary input y halts on input $i_Q p_0$ also with output $|z_0\rangle$. Consequently, the program q that minimizes the right-hand side of (8.19) with U substituted for Q, and computes $U(q, y) = |u\rangle$ for some state $|u\rangle$ possibly different from $|z\rangle$, satisfies

$$\begin{aligned} KQ_U(|x\rangle|y) &= l(q) + \left\lceil \log \frac{1}{||\langle u \mid x\rangle||^2} \right\rceil \\ &\leq l(i_Q p_0) + \left\lceil \log \frac{1}{||\langle z_0 \mid x\rangle||^2} \right\rceil. \end{aligned}$$

Combining the two displayed inequalities, and setting $c_Q = l(i_Q)$, proves the theorem. □

For every pair U, U' of universal Turing machines as in the proof of Theorem 8.7.1, there is a fixed constant $c_{U,U'}$, depending only on U and U', such that for all $|x\rangle, y$ we have

$$|KQ_U(|x\rangle|y) - KQ_{U'}(|x\rangle|y)| \leq c_{U,U'}.$$

To see this, substitute U' for Q in (8.19), and, conversely, substitute U' for U and U for Q in (8.19), and combine the two resulting inequalities.

Definition 8.7.6 We fix once and for all a *reference universal quantum Turing machine* U and define the *quantum Kolmogorov complexity* as

$$KQ(|x\rangle|y) = KQ_U(|x\rangle|y),$$
$$KQ(|x\rangle) = KQ_U(|x\rangle|\epsilon),$$

where ϵ denotes the absence of conditional information.

Example 8.7.4 The definition is continuous in the following sense: If two quantum states are very close then their quantum Kolmogorov complexities are very close. Furthermore, since we can approximate every (pure quantum) state $|x\rangle$ to arbitrary closeness, then in particular, for every constant $\epsilon > 0$ we can compute a (pure quantum) state $|z\rangle$ such that $||\langle z \mid x\rangle||^2 > 1 - \epsilon$. One can view this as the probability of obtaining the possibly incomputable outcome $|x\rangle$ when executing projection $|x\rangle\langle x|$ on $|z\rangle$ and measuring outcome $|x\rangle$. ◇

8.7.4 Properties

Our proposal would not be useful if for a directly computable object the complexity were less than the shortest program to compute that object. This would imply that the code corresponding to the probabilistic component in the description is possibly shorter than the difference in program lengths for programs for an approximation of the object and the object itself. This would penalize definite description compared to probabilistic description and in case of classical objects would make quantum Kolmogorov complexity less than classical Kolmogorov complexity. This may be called *consistency*.

Theorem 8.7.2 *Let U be the reference universal quantum Turing machine and let $|x\rangle$ be a basis vector in a directly computable orthonormal basis $\mathcal{B}$, given y, and let there be a program p such that $U(p, y) = |x\rangle$. Then $KQ(|x\rangle|y) = \min_p\{l(p) : U(p, y) = |x\rangle\}$ up to $= KQ(\mathcal{B}|y) + O(1)$.*

Proof. Let $|z\rangle$ be such that

$$KQ(|x\rangle|y) = \min_q \left\{ l(q) + \left\lceil \log \frac{1}{||\langle z \mid x\rangle||^2} \right\rceil : U(q,y) = |z\rangle \right\}.$$

Denote the program q that minimizes the right-hand side by $q_{\min}$ and the program p that minimizes the expression in the statement of the theorem by $p_{\min}$.

A *dovetailed* computation is a method related to Cantor's celebrated diagonalization method: run all programs alternatingly in such a way that every program eventually makes progress. On a list of programs $p_1, p_2, \ldots$ one divides the overall computation into stages $k = 1, 2, \ldots$. In stage k of the overall computation one executes the ith computation step of every program p_{k-i+1} for $i = 1, \ldots, k$.

By running U on all binary strings (candidate programs) simultaneously dovetailed-fashion, one can enumerate all objects that are directly computable, given y, in order of their halting programs. Assume that U is also given a program b of length $KQ(\mathcal{B}|y)$ such that b given y computes $\mathcal{B}$—that is, enumerates the basis vectors in $\mathcal{B}$. In this way, $q_{\min}$ computes $|z\rangle$, and the program b computes $\mathcal{B}$. Now since the vectors of $\mathcal{B}$ are mutually orthogonal,

$$\sum_{|e\rangle \in \mathcal{B}} ||\langle z \mid e\rangle||^2 = 1.$$

Since $|x\rangle$ is one of the basis vectors, we have that $\log 1/||\langle z \mid x\rangle||^2$ is the length of a prefix code (the Shannon–Fano code) to compute $|x\rangle$ from $|z\rangle$ and $\mathcal{B}$. Denoting this code word by r, we have that the concatenation $q_{\min} b r$ is a program to compute $|x\rangle$: parse it into $q_{\min}, b$, and r using the self-delimiting property of $q_{\min}$ and b. Use $q_{\min}$ to compute $|z\rangle$ and use b to compute $\mathcal{B}$; determine the probabilities $||\langle z \mid e\rangle||^2$ for all basis vectors $|e\rangle$ in $\mathcal{B}$. Determine the Shannon–Fano code words for all the basis vectors from these probabilities. Since r is the code word for $|x\rangle$, we can now decode $|x\rangle$. Therefore,

$$l(q_{\min}) + \left\lceil \log \frac{1}{||\langle z \mid x\rangle||^2} \right\rceil \geq l(p_{\min}) - KQ(\mathcal{B}|y) + O(1),$$

which was what we had to prove. □

Corollary 8.7.1 On classical objects (that is, the natural numbers or finite binary strings that are all directly computable) the quantum Kolmogorov complexity

coincides up to a fixed additive constant with the self-delimiting Kolmogorov complexity, since $KQ(\mathcal{B}|n) = O(1)$ for the standard classical basis $\mathcal{B} = \{0,1\}^n$. (We assume that the information about the dimensionality of the Hilbert space is given conditionally.)

Theorem 8.7.3 *For every n-qubit quantum state $|x\rangle$ we have $KQ(|x\rangle|n) \leq 2n + O(1)$.*

Proof. Write $N = 2^n$. For every state $|x\rangle$ in N-dimensional Hilbert space with basis vectors $|e_0\rangle, \ldots, |e_{N-1}\rangle$ we have $\sum_{i=0}^{N-1} ||\langle e_i \mid x\rangle||^2 = 1$. Hence there is an i such that $||\langle e_i \mid x\rangle||^2 \geq 1/N$. Let p be a $(KQ(i|n) + O(1))$-bit program to construct a basis state $|e_i\rangle$ given n. Then $l(p) \leq n + O(1)$. Then $KQ(|x\rangle|n) \leq l(p) + \log N \leq 2n + O(1)$. □

Lemma 8.7.1 *Let $N = 2^n$. There is a particular (possibly nonclassical) orthonormal basis of the N-dimensional Hilbert space $\mathcal{H}_N$, computed from the directly computed pure quantum states, such that at least $2^n(1-2^{-c})$ basis vectors $|e_i\rangle$ satisfy $KQ(|e_i\rangle|n) \geq n - c$.*

Proof. Every orthonormal basis of $\mathcal{H}_N$ has 2^n basis vectors and there are at most $m \leq \sum_{i=0}^{n-c-1} 2^i = 2^{n-c} - 1$ programs of length less than $n - c$. Hence there are at most m programs of length less than $n - c$ available to approximate the basis vectors. We construct an orthonormal basis satisfying the lemma, as follows. The set of directly computed pure quantum states $|x_0\rangle, \ldots, |x_{m-1}\rangle$ spans an m'-dimensional subspace $\mathcal{A}$ with $m' \leq m$ in the N-dimensional Hilbert space $\mathcal{H}_N$ such that $\mathcal{H}_N = \mathcal{A} \oplus \mathcal{A}^\perp$. Here $\mathcal{A}^\perp$ is a $(2^n - m')$-dimensional subspace of $\mathcal{H}_N$ such that every vector in it is perpendicular to every vector in $\mathcal{A}$. We can write every element $|x\rangle \in \mathcal{H}_N$ as

$$\sum_{i=0}^{m'-1} \alpha_i |a_i\rangle + \sum_{i=0}^{2^n - m' - 1} \beta_i |b_i\rangle \,,$$

where the $|a_i\rangle$'s form an orthonormal basis of $\mathcal{A}$ and the $|b_i\rangle$'s form an orthonormal basis of $\mathcal{A}^\perp$, so that the $|a_i\rangle$'s and $|b_i\rangle$'s form an orthonormal basis for $\mathcal{H}_N$. For every state $|x_j\rangle \in \mathcal{A}$, directly computed by a program x_j^* given n, and basis vector $|b_i\rangle \in \mathcal{A}^\perp$, we have $||\langle x_j \mid b_i\rangle||^2 = 0$. Therefore, $KQ(|b_i\rangle|n) \geq l(x_j^*) + \log 1/||\langle x_j \mid b_i\rangle||^2 + O(1) = \infty > n - c$ $(0 \leq j < m,\ 0 \leq i < 2^n - m')$. This proves the lemma. □

Theorem 8.7.4 *With* Pr *the uniform probability we have*

$$\Pr\{|x\rangle : l(|x\rangle) = n,\ \ KQ(|x\rangle|n) \geq n - c\} \geq 1 - \frac{1}{2^c}.$$

Proof. The theorem follows immediately by generalizing Lemma 8.7.1 to arbitrary orthonormal bases:

Claim 8.7.1 Let $N = 2^n$. Every orthonormal basis $|e_0\rangle, \ldots, |e_{2^n-1}\rangle$ of the N-dimensional Hilbert space $\mathcal{H}_N$ has at least $2^n(1 - 2^{-c})$ basis vectors $|e_i\rangle$ that satisfy $KQ(|e_i\rangle|n) \geq n - c$.

Proof. Use the notation of the proof of Lemma 8.7.1. Let A be a set initially containing the programs of length less than $n - c$, and let B be a set initially containing the set of basis vectors $|e_i\rangle$ with $KQ(|e_i\rangle|n) < n-c$. Assume to the contrary that $|B| > 2^{n-c}$. Then at least two of them, say $|e_0\rangle$ and $|e_1\rangle$, and some pure quantum state $|x\rangle$ directly computed from a program of length less than $(n - c)$ satisfy

$$KQ(|e_i\rangle|n) = KQ(|x\rangle|n) + \left\lceil \log \frac{1}{||\langle e_i \mid x\rangle||^2} \right\rceil, \tag{8.20}$$

with $|x\rangle$ being the directly computed part of both $|e_i\rangle$, $i = 0, 1$. This means that $KQ(|x\rangle|n) < n - c - 1$, since not both $|e_0\rangle$ and $|e_1\rangle$ can be equal to $|x\rangle$. Hence for every directly computed pure quantum state of complexity $n-c-1$ there is at most one basis state, say $|e\rangle$, of the same complexity. (In fact only if that basis state is identical to the directly computed state.) Now eliminate every directly computed pure quantum state $|x\rangle$ of complexity $n - c - 1$ from the set A, and the basis state $|e\rangle$ as above (if it exists) from B. We are now left with $|B| > 2^{n-c} - 1$ basis states of which the directly computed parts are included in A with $|A| \leq 2^{n-c-1} - 1$ with every element in A of complexity $\leq n - c - 2$. Repeating the same argument, we end up with $|A| > 1$ basis vectors of which the directly computed parts are elements of the empty set B, which is impossible. □ □

Example 8.7.5 It may be instructive to check the behavior of the approximation part $\log 1/||\langle x \mid z\rangle||^2$ in Definition 8.7.4 on a nontrivial example. Let x be a random classical string with $K(x) \geq l(x)$ and let y be a string obtained from x by complementing one bit, say in position j. By Exercise 2.2.8 on page 123, for every such x of length n there is such a y with complexity $K(y|n) = n - \log n + O(1)$. Since $K(x|n) \leq K(y|n) + K(j|n) + O(1)$, we have $K(j|n) \geq \log n + O(1)$ (and since $j \leq n$, we also have $K(j|n) \leq \log n + O(1)$). Now let $|z\rangle$ be a pure quantum state that has classical bits except that the difference qubit between x and y that has equal probabilities of being observed as 1 and as 0. We can prepare $|z\rangle$ by giving y and the position of the difference qubit (in $\log n$ bits) and therefore $KQ(|z\rangle|n) \leq n + O(1)$.

From $|z\rangle$ we have probability $\frac{1}{2}$ of obtaining x by observing the difference qubit; it follows $KQ(x|n) \leq KQ(|z\rangle|n,j)+O(1)$, and since $KQ(|z\rangle|n) \geq KQ(|z\rangle|n,j)+O(1)$, we have $KQ(|z\rangle|n) \geq n+O(1)$.

From $|z\rangle$ we also have probability $\frac{1}{2}$ of obtaining y by observing the difference qubit, which yields that $K(y|n) \leq KQ(|z\rangle|n,j)+O(1)$. Since also $KQ(|z\rangle|n) \geq KQ(|z\rangle|n,j)+O(1) \geq KQ(|z\rangle|n)-K(j|n)+O(1) = KQ(|z\rangle|n)+O(1) = n-\log n+O(1)$, we obtain $n-\log n \leq K(y|n)+O(1) \leq n+O(1)$. This is the strongest conclusion we can draw about y constructed as above from the fact that it is the result of observing one qubit of a high complexity $|z\rangle$ constructed as above. That is, if we flip an ith bit of x with complexity $K(i|n) = \log n+O(1)$, then this will not necessarily result in a string of complexity $n-\log n+O(1)$ (take for example $i = j/2$ with j as above). ◇

Example 8.7.6 Theorem 8.7.3 states an upper bound of $2n$ on $KQ(|x\rangle|n)$. This leaves a relatively large gap with the lower bound of n established in Theorem 8.7.4. Exercise 8.7.1 shows that there are states $|x\rangle$ with $KQ(|x\rangle|n) \geq 2n-2\log n-O(1)$; in fact, most states satisfy this. The proof supports about the same incompressibility results as in this section, with n replaced by $2n-2\log n$. ◇

Example 8.7.7 For classical complexity we have $KQ(x,x) = KQ(x)+O(1)$, since a classical program to compute x can be used twice; indeed, it can be used many times. In the quantum world things are not so easy: the no-cloning property of quantum mechanics prevents cloning an unknown pure state $|x\rangle$ perfectly to obtain $|x\rangle|x\rangle$: that is, $KQ(|x\rangle) < KQ(|x\rangle|x\rangle) \leq 2KQ(|x\rangle)+O(1)$. ◇

Exercises

8.7.1. [34] Show that all but an exponentially vanishing fraction of states $|x\rangle$ have $KQ(|x\rangle|n) \geq 2n-2\log n-O(1)$.

Comments. This shows that most pure quantum states are maximally incompressible, in line with the classical case. Hint: Since the quantum Kolmogorov complexity of an n-qubit state is $\leq 2n+O(1)$, the set of directly computable pure n-qubit states has cardinality $A \leq 2^{2n+O(1)}$. The set of unit vectors in $\mathcal{H}_N$ forms the surface of the N-dimensional ball with unit radius in Hilbert space. Every vector can be described by a directly computable vector closest to it and the fidelity part. Source: P. Gács, *J. Phys. A: Math. Gen.*, 34(2001), 6859–6880.

8.7.2. [37] Define the notion of a quantum Turing machine, and show that there is a universal quantum Turing machine that can simulate t steps of every other quantum Turing machine on input x up to precision ϵ (in the fidelity sense) in time polynomial in t, $l(x)$, and $1/\epsilon$.

Comments. Source: E. Bernstein and U. Vazirani, *SIAM J. Comput.*, 26:5(1997), 1411–1473.

8.7.3. [34] Show that every quantum Turing machine can be simulated to every degree of precision by a quantum Turing machine that has a single primitive rotation θ with $\cos\theta = \frac{3}{5}$ and $\sin\theta = \frac{4}{5}$.

Comments. Hence, we can restrict ourselves to such quantum Turing machines in the definition of KQ. Then, there are only countably many Turing machines, they can be enumerated, and the universal machine simulates the others exactly. L.M. Adleman, J. DeMarrais, and M.-D.A. Huang *SIAM J. Comput.*, 26:5(1997), 1524–1540.

8.7.4. [35] Let $|x\rangle$ be a pure quantum state. Let $QC^{\epsilon}(|x\rangle)$ be the length k of the smallest qubit program that when given $|x\rangle$ as input together with ϵ, results in an output density matrix σ such that $\langle x \mid \sigma \mid x\rangle \geq 1-\epsilon$. (This requires the proper definition of a quantum Turing machine that takes qubit strings as input.) We write $QC(|x\rangle) \leq k$ if there is a k-qubit state $|p\rangle$ such that for all ϵ of the form $1/m$, if $|p\rangle$ and ϵ are given as input to the reference quantum Turing machine, we obtain an output density matrix σ with $\langle x \mid \sigma \mid x\rangle$ with probability at least $1-\epsilon$. Show that:

(a) For classical strings x we have $QC(x) = C(x) \pm O(1)$.

(b) For every n, there is a classical x of length n such that $QC(x) \geq n$, and at least $2^n - 2^{n-c} + 1$ mutually orthogonal qubit strings of length n have complexity at least $n - c$.

(c) For every m and n there is an n-bit qubit state $|x\rangle$ such that with $N = 2^n$, the complexity of multiples $|x\rangle^{\otimes m}$ consisting of m copies of $|x\rangle$ satisfies

$$\log \binom{m+N-1}{m} \leq QC(|x\rangle^{\otimes m}).$$

(d) For every m and n, every n-bit qubit state $|x\rangle$ satisfies, with $N = 2^n$,

$$QC(|x\rangle^{\otimes m}) \leq \log \binom{m + 2^{QC(|x\rangle)-1}}{2^{QC(|x\rangle)-1}} + O(\log m).$$

(e) For every pair of qubit strings $|x\rangle$ and $|y\rangle$ we have

$$QC(|x\rangle, |y\rangle) \leq QC(|x\rangle, |x\rangle) + QC(|y\rangle||x\rangle) + O(\log QC(|x\rangle)).$$

Comments. This is the quantum Kolmogorov complexity with qubit string descriptions [A. Berthiaume, W. van Dam, and S. Laplante, *J. Comput. Systems Sci.*, 63:2(2001), 201–221].

8.7.5. [44] In analogy with the universal (greatest) lower semicomputable probability mass function **m**, show that there exists a universal (greatest) lower semicomputable density matrix, denoted by μ. We can define the complexity of a pure quantum state $|x\rangle$ in two manners:

$$KG(|x\rangle) = -\langle x \mid \log \mu \mid x\rangle,$$
$$Kg(|x\rangle) = -\log \langle x \mid \mu \mid x\rangle.$$

We use QC and QC^ϵ according to the definitions in Exercise 8.7.4.

(a) Show that $Kg(|x\rangle) \leq KG(|x\rangle) + O(1)$.

(b) Show that there are pure quantum states $|x\rangle$ in $\mathcal{H}_N$ with $N = 2^n$ such that $Kg(|x\rangle) = O(1)$ and $KG(|x\rangle) = n/2 + O(1)$.

(c) Show that $Kg(|x\rangle) \leq KQ(|x\rangle) + O(1) \leq 4Kg(|x\rangle) + 2\log Kg(|x\rangle)) + O(1)$.

(d) Let $|x_1\rangle, |x_2\rangle, \ldots$ be a computable sequence of orthogonal states. Show that $KG(|x_i\rangle) = Kg(|x_i\rangle) + O(1) = C(x_i) + O(1)$, for all i.

(e) Show that $KG(|x\rangle) \leq n + O(1)$ for an n-qubit state $|x\rangle$.

(f) Show that $K(x) = KG(|x\rangle) + O(1) \leq QC(|x\rangle) + K(QC(|x\rangle)) + O(1)$.

(g) Show that for $\epsilon < \frac{1}{2}$, if $QC^\epsilon(|x\rangle) \leq k$, then $KG(|x\rangle) \leq k + K(k) + 2\epsilon n$.

(h) Show that for every rational ϵ and every computable density matrix μ we have $QC^\epsilon(|x\rangle) \leq \langle x \mid -\log \mu \mid x\rangle \;/\; \epsilon + K(\mu)$.

(i) Show that for every rational ϵ and every computable density matrix μ we have $QC^\epsilon(|x\rangle|\chi) \leq KG(|x\rangle)/\epsilon + O(1)$, where χ is the infinite binary characteristic sequence of the halting set, Definition 1.7.3 on page 34. (The notion of an oracle with a read-only classical oracle tape is not problematic.)

(j) For $|x\rangle$ and $|y\rangle$ being n-qubit states, show that $KG(|x\rangle|y\rangle) \leq KG(|x\rangle) + KG(|y\rangle) + O(1)$, and similarly for Kg.

(k) For $|x\rangle$ and $|y\rangle$ being n-qubit states, show that $KG(|x\rangle) \leq KG(|x\rangle|y\rangle) + O(1)$, and similarly for Kg.

(l) Show that for every m and n, for some n-bit qubit state $|x\rangle$, with $N = 2^n$, the complexity of multiples satisfies

$$\log \binom{m+N-1}{m} \leq Kg(|x\rangle^{\otimes m}).$$

(m) Show that for every m and n, every n-bit qubit state $|x\rangle$ satisfies, with $N = 2^n$,

$$KG(|x\rangle^{\otimes m}) \leq \log \binom{m+N-1}{m} + O(\log m).$$

Comments. Hint for Item (b): use the n-qubit pure quantum state $|x\rangle = \frac{1}{\sqrt{2}}(|00\ldots0\rangle + |11\ldots1\rangle)$. Source: P. Gács, *Ibid.*

8.7.6. [29] Show that the complexity of multiples for KQ satisfies the following. First, for every m and n there is an n-bit qubit state $|x\rangle$ such that with $N = 2^n$, the complexity of multiples satisfies

$$\log \binom{m+N-1}{m} \leq KQ(|x\rangle^{\otimes m}).$$

Second, for every m and n, every n-bit qubit state $|x\rangle$ satisfies, with $N = 2^n$,

$$KQ(|x\rangle^{\otimes m}) \leq 4\left[K(m) + \log \binom{m+N-1}{m} + O(\log m)\right].$$

Comments. Hint: use the argument to prove Exercise 8.7.4, Items (d), (e), combined with Exercise 8.7.5, Items (a), (c), (l), (m). Source: P.M.B. Vitányi, *IEEE Trans. Inform, Theory*, 47:6(2001), 2464–2479; Correction, 48:4(2002), 1000.

8.7.7. [26] Prove the nonsubadditivity property of KQ in two ways. First, show that there are n-qubit states $|x\rangle$ and $|y\rangle$ such that

$$KQ(|x\rangle|y\rangle) > KQ(|x\rangle) > KQ(|x\rangle|\,|y\rangle) + K(|y\rangle).$$

Second, show that there are infinitely many m and n such that

$$KQ(|x\rangle^{\otimes m} > KQ(|x\rangle^{\otimes m/2}|\,|x\rangle^{\otimes m/2}) + KQ(|x\rangle^{\otimes m/2}),$$

where $>$ is meant in the sense of '$\not<$ up to an additive constant.'

Comments. Hint for Item (a): use $|y\rangle = \frac{1}{\sqrt{2}}(|00\ldots0\rangle + |x\rangle)$. Source: P.M.B. Vitányi, *Ibid.*

8.8 Compression in Nature

Learning, in general, appears to involve compression of observed data or the results of experiments. It may be assumed that this holds both for the organisms that learn, as well as for the phenomena that are being learned. If the learner can't compress the data, he or she doesn't learn. If the data are not regular enough to be compressed, the phenomenon that produces the data is random, and its description does not add value to the data of a generalizing or predictive nature.

8.8.1 Compression by Ants

In everyday life, we continuously compress information that is presented to us by the environment. Perhaps animals do this as well, as the following experiment reported by Zh.I. Reznikova and B.Ya. Ryabko [*Problems*

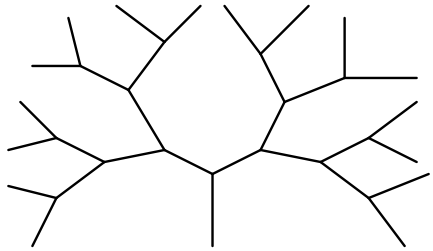


FIGURE 8.19. The maze: a binary tree constructed from matches

Inform. Transmission, 22:3(1986), 245–249] suggests. The authors claim that the transmission of information by ants using tactile code is a well-established fact. This led the researchers to probe both the information transmission rate and message-compressing capabilities of ants. We describe the latter issue.

Colonies of ants (*Formica sanguinea*) were put in artificial nests on laboratory tables. The ants were fed only once every three days, but only at some exit of a maze. This maze was a binary tree, as shown in Figure 8.19, with the root connecting to the nest and the food source at a leaf.

Each leaf contained a feeder. Only one feeder contained sugar syrup. All other feeders were empty. Since the tree was constructed by matches floating on water, the ants could not take short cuts. To reach the maze, an ant had to cross a bridge. This allowed hiding the maze in order that the remaining ants could not see the maze from their nest. To prevent marking of the trail by odorous substances, matches crossed by the ants were periodically replaced with fresh ones. During the experiments, the ants were marked with individual and group labels. Experiments with different numbers of branches were conducted in several sessions. In each case, the number of correct and wrong turns made by the ants was recorded, and also the total duration of tactile contacts between scouts and foragers was measured.

First, the scout ants would venture out to look for food. After a scout located the syrup in the maze, it returned to the nest to communicate the way to the syrup to the forager ants. Subsequently, the scouts were isolated and the foragers went in search of the syrup with, presumably, second-hand information. The simplest maze used was a two-leaf tree. In this setting, the scouts had to communicate only one bit. In the course of the experiment the depth of the tree was increased to 6. Since the scouts still managed to transmit the location of the syrup, the number of messages the ants could communicate is at least $2^7 - 1 = 127$.

No.	SEQUENCE OF TURNS TO SYRUP	MEAN TIME SEC.	SAMPLE STANDARD DEVIATION	NUMBER OF TESTS
1	LLL	72	8	18
2	RRR	75	5	15
3	LLLLLL	84	6	9
4	RRRRR	78	8	10
5	LLLLL	90	9	8
6	RRRRRR	88	9	5
7	LRLRLR	130	11	4
8	RLRLRL	135	9	8
9	LLR	69	4	12
10	LRLL	100	11	10
11	RLLLR	120	9	6
12	RRLRL	150	16	8
13	RLRRRL	180	20	6
14	RRLRRR	220	15	7
15	LRLLRL	200	18	5

FIGURE 8.20. Time required for *Formica sanguinea* scouts to transmit information about the direction to the syrup to the forager ants

Let L mean a left turn and R a right turn. It was found that the ants could communicate simple roads, like LLLLLL or LRLRLR, faster than more random roads. This seems to indicate that the ants compress the information before transmitting it.

The table in Figure 8.20 contains the results of the experiments. Apparently, it takes a longer time for the scouts to communicate random sequences to the foragers than to communicate regular sequences.

8.8.2 Compression by Science

Science may be regarded as the art of data compression. Compression of a great number of experimental data into the form of a short natural law yields an increase in predictive power, as shown in Chapter 5.

Whether science can exist would seem to depend on the question whether the mass of experimentally obtained data form a compressible sequence. A randomly generated string (or universe) is with overwhelming probability not algorithmically compressible.

In Section 2.6 it was shown that maximally random sequences must contain logarithmically long very regular sequences. It may be the case that our part of the universe is an oasis of regularity in a maximally random universe. In practice, discovery of scientific laws with great predictive powers and many applications progresses spectacularly. This evidences, even if it doesn't prove, inherent order in our universe. According to Francis Bacon (1561–1626), "The eye of the understanding is like the

eye of the sense: for as you may see great objects through small crannies or levels, so you may see great axioms of nature through small and contemptible instances."

8.9 History and References

Section 8.1.1 is partially based on T.M. Cover, P. Gács, and R.M. Gray, *Ann. Probab.*, 17:3(1989), 840–865. Relations between physical entropy and expected Kolmogorov complexity are studied by W.H. Zurek in [*Phys. Rev. A*, 40:8(1989), 4731–4751; pp. 73–89 in: *Complexity, Entropy and the Physics of Information*, W.H. Zurek, ed., Addison-Wesley, 1991; *Nature*, 341(September 1989), 119–124], by C.M. Caves in [pp. 91–115 in: *Complexity, Entropy and the Physics of Information*, W.H. Zurek, ed., Addison-Wesley, 1991; pp. 47–89 in: *Physical Origins of Time Asymmetry*, J.J. Halliwell, J. Pérez-Mercader, and W.H. Zurek, eds., Cambridge Univ. Press, 1994], and by R. Schack [*Int. J. Theoret. Phys.*, 36(1997), 209–226]. Theorem 8.1.3 is due to A.E. Romashchenko in [D. Hammer, A.E. Romashchenko, A.K. Shen, and N.K. Vereshchagin, *J. Comput. System Sci.*, 60(2000), 442–464]. This theorem states roughly that all linear information (in)equalities for Shannon entropy of random variables also hold for Kolmogorov complexity of strings up to logarithmic accuracy, and conversely. In [An.A. Muchnik and N.K. Vereshchagin, *Proc. Int. Comput. Sci. Symp. Russia* (*CSR*), Lect. Notes. Comput. Sci., Vol. 3967, Springer-Verlag, Berlin, 2006, 281–291] the information inequalities of Theorem 8.1.3 are interpreted as universaly quantified ($\forall$) statements in a language having $\leq$ as its only predicate symbol and terms of the left-hand side form. It is shown that in contrast to the predicate logic $\forall$ statements, if we compare the Shannon and Kolmogorov theories using $\forall\exists$ formulas in this language, then the situation is different. There is a $\forall\exists$-formula that is valid in Kolmogorov's theory but that is false in Shannon's theory. There also is a $\forall\exists$-formula that is true in Shannon's theory, provided the universal quantifiers range over sequences of i.i.d. random variables, but which is false in Kolmogorov's theory.

Sections 8.1.2 and 8.1.3 on algorithmic mutual information are based on [P. Gács, J.T. Tromp, and P.M.B. Vitányi, *IEEE Trans. Inform. Theory*, 47:6(2001), 2443–2463; Correction, 48:8(2002), 2427], using ideas of L.A. Levin [*Problems Inform. Transmission*, 10:3(1974), 206–210; *Inform. Contr.*, 61(1984), 15–37]. Application of mutual information to Gödel's incompleteness theorem is due to [L.A. Levin. *Proc. 43rd IEEE Symp. Found. Comput. Sci.*, 2002, 761–768; Arxiv cs.CC/0203029]. The quotation from K. Gödel is from his lecture [K. Gödel, "The Modern Development of the Foundations of Mathematics in the Light of Philosophy," 1961, Reprinted in *Collected Works, Volume III*, (1961), Oxford Univ. Press, 1981].

Section 8.1.4 is based on N.K. Vereshchagin and P.M.B. Vitányi, Arxiv cs.IT/ 0411014; *IEEE Trans. Inform. Theory*, Submitted. Application of the theory to practical denoising and lossy compression experiments using real compressors as in Figure 8.2 on page 621, are given in [S. de Rooij and P.M.B. Vitányi, Arxiv cs/0609121; *IEEE Trans. Comput.*, Submitted]. Algorithmic rate-distortion theory hails from [E.-H. Yang and S.-Y. Shen, *IEEE Trans. Inform. Theory*, 39:1(1993), 288–292; J. Muramatsu and F. Kanaya, *IEICE Trans. Fundamentals*, E77-A:8(1994), 1224–1229; D.M. Sow and A. Eleftheriadis, *IEEE Trans. Inform. Theory*, 49:3(2003), 604–608], which relate the classical and algorithmic approaches according to traditional information-theoretic concerns. J. Ziv [*IEEE Trans. Inform. Theory*, 26:2 (1980), 137–143] also considers a rate-distortion function for individual data. The rate-distortion function is assigned to every infinite sequence ω of letters of a finite alphabet $\mathcal{X}$, and not to a finite object, as in Section 8.1.4. The source words x are prefixes of ω and the encoding function is computed by a finite-state transducer. Kolmogorov complexity is not involved.

Section 8.2 is based on the following work. The relation between information and energy was derived from thermal noise considerations by J. H. Felker [*Proc. IRE* 40(1952), 728–729; *Ibid.* 42(1954), 1191]. There, a derivation based on statistical thermodynamic considerations is mentioned and attributed to J.R. Pierce. See also [J. R. Pierce and C. C. Cutler, pp. 55–109 in: F. I. Ordway, III, ed., *Advances in Space Science, Vol. 1*, Academic Press, Inc., New York, 1959]. A further attribution to J. von Neumann appears in the edited volume [*Theory of Self-Reproducing Automata*, A.W. Burks, ed., Univ. Illinois Press, Urbana, 1966]. (However, this relation is not mentioned in von Neumann's own writing.) The thermodynamic cost of erasing information was first studied by R. Landauer in [*IBM J. Res. Develop.*, 5(1961), 183–191]. Logical reversibility of computation was first studied by Y. Lecerf, *Comptes Rendus*, 257(1963), 2597–2600; C.H. Bennett, *IBM J. Res. Develop.*, 17(1973), 525–532. For Section 8.2 we used to a large extent [C.H. Bennett, *Scientific American*, 257(Nov. 1987), 108–116; C.H. Bennett and R. Landauer, *Scientific American*, 253(July 1985), 48–56; C.H. Bennett, *Int. J. Theoret. Phys.*, 21:12(1982), 905–940; see also *Phys. Rev. Lett.*, 53:12(1984), 1202–1206]. These papers discuss thermodynamics of computation in relation to reversible computing. The ballistic (molecular) computer in Section 8.2.3 and its reversibility were investigated in [T. Toffoli, *Proc. 7th Int. Colloq. Automata, Languages and Prog., Lect. Notes Comp. Sci.*, Vol. 85, Springer-Verlag, 1980, pp. 632–644; *Math. Systems Theory*, 14(1981), 13–23; and E. Fredkin and T. Toffoli, *Int. J. Theoret. Phys.*, 21(1982), 219–253]. The issue *Int. J. Theoret. Phys.*, 21(1982) is dedicated to the physics of computing, containing papers presented at a conference on this topic at MIT in 1981. The second and third conferences on these

and related issues took place in Dallas in 1992 and 1994 [*Proc. IEEE Workshop on Physics and Computation*, Dallas, 1992 and 1994 (also: *Preliminary Proceedings* of the 1992 Conference)]. C.H. Bennett [*Int. J. Theoret. Phys.*, 21:12(1982), 905–940] suggested a Brownian computer as a more stable alternative to the ballistic computer. Other stable alternatives in terms of quantum-mechanical models were devised by P.A. Benioff [*Int. J. Theoret. Physics*, 21(1982), 177–202; *Ann. New York Acad. Sci.*, 480(1986), 475–486], and R.P. Feynman [*Optics News*, 11(1985), 11]. Related work about time–space tradeoffs in reversible computing is [C.H. Bennett, *SIAM J. Comput.*, 18(1989), 766–776; R.Y. Levine and A.T. Sherman, *SIAM J. Comput.*, 19(1990), 673–677; M. Li and P.M.B. Vitányi, *Proc. Royal Soc. London, Ser. A*, 452(1996), 769–789]. A comprehensive study in time–energy tradeoffs in reversible computation and space–energy tradeoffs in reversible simulation of irreversible computation is [M. Li and P.M.B. Vitányi, *Ibid.*] and of time–space tradeoffs in [M. Li, J.T. Tromp, and P.M.B. Vitanyi, *Physica D*, 120(1998) 168–176; K.J. Lange, P. McKenzie, and A. Tapp, *J. Comput. Syst. Sci.*, 60:2(2000), 354–367; H.M. Buhrman, J.T. Tromp, and P.M.B. Vitanyi, *J. Physics A: Math. and General*, 34(2001), 6821–6830]. Related material is contained in [C.H. Bennett, P. Gács, M. Li, P.M.B. Vitányi, and W.H. Zurek, *IEEE Trans. Inform. Theory*, 44:4(1998), 1407–1423].

Other proposals to implement real physical reversible computing devices that dissipate no energy (or almost no energy) are bistable magnetic devices for reversible canceling/copying of records in [R. Landauer, *IBM J. Res. Develop.*, 5(1961), 183–191], and Brownian computers [R.W. Keyes and R. Landauer, *IBM J. Res. Develop.*, 14(1970), 152–157], for Turing machines and Brownian enzymatic computers [C.H. Bennett, *IBM J. Res. Develop.*, 17(1973), 525–532; C.H. Bennett, *BioSystems*, 11(1979), 85–90; C.H. Bennett and R. Landauer, *Scientific American*, 253(July 1985), 48–56], with respect to reversible Boolean circuits [E. Fredkin and T. Toffoli, *Int. J. Theoret. Phys.*, 21(1982), 219–253], and Brownian computing using Josephson devices [K. Likharev, *Int. J. Theoret. Phys.*, 21(1982), 311–326].

All these models seem mutually simulatable. For more background information, see [C.H. Bennett, *Int. J. Theoret. Phys.*, 21(1982), 905–940]. Related material on reversible computing is [C.H. Bennett, *IBM J. Res. Develop.*, 32(1988), 16–23; R.W. Keyes and R. Landauer, *IBM J. Res. Develop.*, 14(1970), 152–157; R. Landauer, *Int. J. Theoret. Phys.*, 21(1982), 283; *Nature*, 335(October 1988), 779–784; R.W. Keyes, *Science*, 230(October 1985), 138–144; *Rev. Mod. Phys.*, 61(1989), 279–287; *IBM J. Res. Develop.*, 32(1988), 24–28; *Phys. Today*, 45(August 1992), 42–48]. Proposals for implementation of reversible computation in existing electronic circuit technology (CMOS, nMOS) are given in [R.C. Merkle, *Nanotechnology*, 4(1993), 21–40]; see also [*Proc. IEEE Workshop*

on Physics and Computation, Dallas, 1992, 227–228, 237–247, 267–270; K.E. Drexler, *Nanosystems: Molecular Machinery, Manufacturing, and Computation*, Wiley, 1992].

Section 8.3 follows [C.H. Bennett, P. Gács, M. Li, P.M.B. Vitányi, and W.H. Zurek, *IEEE Trans. Inform. Theory*, 44:4(1998), 1407–1423]. Information distance was developed originally with thermodynamics of computation in mind [M. Li and P.M.B. Vitányi, *Proc. IEEE Workshop on Physics and Computation*, Dallas, 1992, 42–46 (complete version in the *Preliminary Proceedings* of the 1992 Conference); M. Li and P.M.B. Vitányi, *Proc. Royal Soc. London, Ser. A*, 452(1996), 769–789].

Further developments diverged in a theoretical direction and a practical one. On the theoretical side, N.K. Vereshchagin observed and suggested that the mutual information condition $C(x)-C(x|y) = 0$ can possibly be replaced by the stronger requirement $C(q|x) = 0$ and $C(p|y) = 0$, which led to the important work in Section 8.3.7. The main item, Muchnik's theorem, Theorem 8.3.7 appears in [An.A. Muchnik, *Theoret. Comput. Sci.*, 271(2002), 97–109]. Related topics have been investigated further in a series of wonderful papers by Russian scientists [N.K. Vereshchagin and M.V. Vyugin, *Theoret. Comput. Sci.*, 271(2002), 131–143; A.V. Chernov, An.A. Muchnik, A.E. Romashchenko, A.K. Shen, and N.K. Vereshchagin, *Theoret. Comput. Sci.*, 271(2002), 69–95; A.K. Shen and N.K. Vereshchagin, *Theoret. Comput. Sci.*, 271(2002), 125–129; An.A. Muchnik and N.K. Vereshchagin, *Proc. 16th Conf. Comput. Complexity*, 2001, pp. 256–265; M.V. Vyugin, *Theoret. Comput. Sci.*, 271(2002), 145–150].

The practical direction of normalized information distance was initiated by [M. Li, J.H. Badger, X. Chen, S. Kwong, P. Kearney, and H. Zhang, *Bioinformatics* 17:2(2001), 149–154]. Initially for the purpose of comparing genomes of different sizes, it introduced the ideas of alignment-free whole genome phylogeny and shared information distance, based on the normalized sum distance E_4, Theorem 8.3.6 and Exercise 8.4.5. They showed that it is possible to replace Kolmogorov complexity by common compression programs and still achieve good results, Example 8.4.4. C.H. Bennett, M. Li, and B. Ma [*Scientfic American*, 288:6(2003), 76–81] further applied this approach to infer the evolutionary history of some chain letters. These publications were based on research in 1998–1999.

Previously, there had been efforts to use the asymmetric conditional Kolmogorov complexity [S. Grumbach and F. Tahi, *J. Inform. Process. Management*, 30:6(1994) 857–866] or a problem-dependent transformation distance, specific only for genomes, also in the spirit of conditional Kolmogorov complexity [J.S. Varré, J.P. Delahaye, and É. Rivals, *Bioinformatics* 15:3(1999), 194–202].

Section 8.4 and the applications in Section 8.4.3 are based on the normalized version of the more optimal max distance version E_1, Theorem 8.3.2, as used in [M. Li, X. Chen, X. Li, B. Ma, and P.M.B. Vitányi, *IEEE Trans. Inform. Theory*, 50:12(2004), 3250–3264]. This reference is based on research in 2000–2001. (The survey [M. Li and P.M.B. Vitányi, pp. 376–382 in: *Int. Ency. Social & Behav. Sci.*, N.J. Smelser, P.B. Baltes, eds., Pergamon, Oxford, 2001/2002] already mentions normalized max distance and its metricity.) Other applications in phylogeny are reconstruction of the language tree and authorship attribution [D. Benedetto, E. Caglioti, V. Loreto, *Phys. Rev. Lett.*, 88:4(2002), 048702 in independent work using a related compression-based distance; M. Li, X. Chen, X. Li, B. Ma, and P.M.B. Vitányi, *Ibid*], and plagiarism detection [X. Chen, B. Francia, M. Li, B. McKinnon, and A. Seker, *IEEE Trans. Inform. Theory*, 50:7(2004), 1545–1551].

The method was first applied to hierarchical clustering of nontreelike data (nonphylogeny), see the example in Section 8.4.3, in research dating from 2002/2003 published in [R.L. Cilibrasi, P.M.B. Vitányi, and R. de Wolf, *Comput. Music J.*, 28:4(2004), 49–67] about music clustering, and [R.L. Cilibrasi and P.M.B. Vitányi, *IEEE Trans. Inform. Theory* 51:4(2005), 1523–1545] about general hierarchical clustering and classification of artificial data, heterogeneous data, genomics, viruses, fungi, languages, literature, music, optical character recognition, and astronomical data. In the latter paper an axiomatic treatment of the normalized compression distance of Equation 8.13 on page 664, based on real-world compressors, is given together with a proof of metricity; see Exercise 8.4.6 on page 672.

Applications of various forms of normalized information distance have become standard in many areas. A few of those applications are [A. Kraskov, H. Stögbauer, R.G. Andrzejak, and P. Grassberger, *Europhys. Lett.* 70:2(2005), 278–284], software design [S.R. Kirk and S. Jenkins, *J. Systems and Software*, 72(2004), 179–186], protein sequence classification [A. Kocsor, A. Kertesz-Farkas, L. Kajan, and S. Pongor, *Bioinformatics*, 22:4(2006), 407–412], measuring protein structure similarity [N. Krasnogor and D.A. Pelta, *Bioinformatics* 20:7(2004), 1015–1021], phylogenetic reconstruction [C. Ané and M.J. Sanderson, *Systematic Biology*, 54:1(2005), 146–157; S.L.K. Pond, S.D.W. Frost, and S.V. Muse, *Bioinformatics* 21:5(2004), 676–679; H.H. Otu and K. Sayood, *Bioinformatics* 19:6(2003), 2122–2130], hurricane risk assessment [K. Emanuel, S. Ravela, E. Vivant, and C. Risi, "A combined statistical-deterministic approach of hurricane risk assessment," Program in Atmospheres, Oceans, and Climate, MIT, 2005], SVM kernel for string classification [M. Cuturi and J.P. Vert, *Neural Networks*, 18:4(2005), 1111–1123], ortholog detection [H.K. Pao and J. Case, *Int. Conf. Comput. Intell.* Dec. 17–19, 2004, Istanbul Turkey], clustering fetal heart rate tracings in clinical medical

data [C. Costa Santos, J. Bernardes, P.M.B. Vitányi, and L. Antunes, *Proc. 19th IEEE Int. Symp. Computer-Based Medical Systems*, 2006, 685–690], analyzing worms and network traffic, S. Wehner [*J. Comput. Security*, 15:3(2007), 303–320], network structure and dynamic behavior [M. Nykter, N.D. Price, A. Larjo, T. Aho, S.A. Kauffman, O. Yli-Harja, and I. Shmulevich, *Phys. Rev. Lett.*, 100(2008), 058702(4)], gene expression dynamics in macrophage exhibiting criticality [M. Nykter, N.D. Price, M. Aldana, S.A. Ramsey, S.A. Kauffman, L.E. Hood, O. Yli-Harja, and I. Shmulevich, *Proc. Nat. Acad. Sci. USA*, 105:6(2008), 1897–1900], and the open-source Complearn toolkit of R.L. Cilibrasi at www.complearn.org.

The normalized information distance has been validated by performance studies confirming that it is superior or competitive: [E.J. Keogh, S. Lonardi, and C.A. Rtanamahatana *Proc. ACM SIGKDD Int. Conf. Knowledge Discov. Data Mining*, 2004, 206–215; E.J. Keogh, S. Lonardi, C.A. Rtanamahatana, L. Wei, S.H. Lee, and J. Handley, *Data Min. Knowl. Disc.*, 14:1(2007), 99–129] did a massive comparative performance study of the compression-based normalized information method with other methods used in data-mining of time sequences; see the comment on page 666. P. Ferragina, R. Giancarlo, V. Greco, G. Manzini, and G. Valiente, [*BMC Bioinformatics*, 8:1(2007) Jul 13, 252 17629909] experimentally tested the normalized information distance using 25 compressors to obtain the NCD, and six data sets of relevance to molecular biology; see the comment on page 665. M. Cebrián, M. Alfonseca, A. Ortega, [*Commun. Inform. Syst.*, 5:4(2005), 367–384] investigated how far the performance of real-world compressors like gzip, bzip2, and PPMZ satisfy the identity axiom of a normal compressor Z, see Exercise 8.4.6 on page 672, as well as a performance study on the well-known Calgary Corpus. The same authors have shown in [*IEEE Trans. Inform. Theory*, 53:5(2007), 1895–1900] that the NCD is resistant to noise.

Section 8.4.4 is based on [R.L. Cilibrasi and P.M.B. Vitányi, *IEEE Trans. Knowledge Data Engin.*, 19:3(2007), 370–383]. This paper introduced the idea of approximating a normalized semantic distance between names for objects and abstract concepts in general using Internet statistics in the normalized compression distance formula, with application to clustering, classification, and language translation. A massive experiment comparing the performance with the human-expert-entered information in the WordNet database yielded a mean accuracy of agreement of 87.25%. The method opens the door to a wider range of applications of the normalized information distance, for example [W. Wong, W. Liu, and M. Bennamoun, *Proc. ACM Int. Symp. Practical Cogn. Agents and Robots*, 2006, 177–191], and the QUANTA question–answer system incorporating an application of normalized information distance [X. Zhang, Y. Hao,

X. Zhu, and M. Li, *Proc. 13th ACM SIGKDD Int. Conf. Knowledge Discov. Data Mining*, 2007, 874–883].

Replacing Kolmogorov complexity by real-world compression has recently been used to do statistical testing, for example in [B.Ya. Ryabko, J. Astola, and A. Gammerman *Theoret. Comput. Sci.*, 1–3(2006), 440–448; B.Ya. Ryabko and J. Astola, *Statistical Methodology*, 3:4(2006), 375–397].

For classical thermodynamics and entropy see [E. Fermi, *Thermodynamics*, Dover, 1956]. We partly used the survey [J. Schumacher, *CWI Quarterly*, 6:2(1993), 97–120]. The study of statistical thermodynamics using Kolmogorov complexity was begun by C.H. Bennett, [*Int. J. Theoret. Phys.*, 21:12(1982), 905–940; *Scientific American*, 257(Nov, 1987), 108–116]. Initially, the algorithmic entropy of Section 8.6.1 was proposed by W.H. Zurek [*Phys. Rev. A*, 40:8(1989), 4731–4751; pp. 73–89 in *Complexity, Entropy and the Physics of Information*, W.H. Zurek, ed., Addison-Wesley, 1991; *Nature*, 341(September 1989), 119–124] under the name physical entropy.

The initial discussion of Maxwell's demon accounting for the thermodynamic cost of irreversible information erasure as in Example 8.6.1 is due to C.H. Bennett [*Int. J. Theoret. Phys.*, 21:12 (1982), 905–940; *Scientific American*, 257(Nov, 1987), 108–116]. These papers contain an excellent exposition on Maxwell's demon, including a construction of a device for measuring the position of a molecule. This type of solution was further explored by W.H. Zurek using physical entropy arguments in [*Nature*, 341(September 1989), 119–124; *Phys. Rev. A*, 40:8(1989), 4731–4751; pp. 73–89 in *Complexity, Entropy and the Physics of Information*, W.H. Zurek, ed., Addison-Wesley, 1991]. The discussion on Maxwell's demon in Example 8.6.1, and Claim 8.6.1, follow by and large Zurek's discussion. The Szilard engine was described by L. Szilard in a paper entitled "On the decrease of entropy in a thermodynamic system by the intervention of intelligent beings" [*Z. Phys.*, 53(1929), 840–856]. Szilard in this paper in fact also discovered the relationship between entropy and information. However, this was not generally accepted until this relation was rediscovered by C.E. Shannon in the 1940s. A collection of these and other key papers on this topic is H.S. Leff and A.F. Rex, eds. [*Maxwell's Demon: Entropy, Information, Computing*, Princeton University Press, 1990; *Maxwell's Demon 2: Entropy, Classical and Quantum Information, Computing*, Institute of Physics Publishing, London, 2002]. See also [R.G. Brewer and E.L. Hahn, *Scientific American*, December 1984, 42–49].

An initial approach to an algorithmic thermodynamics was partially published in [M. Li and P.M.B. Vitányi, *Proc. 19th Int. Colloq. Automata, Languages and Prog.*, Lect. Notes Comp. Sci., Vol. 623, Springer-

Verlag, Berlin, 1992, 1–16]. P. Gács removed scale-dependence of our approach by adding coarse-graining and randomness tests, obtaining the physical entropy of W.H. Zurek, *Ibid.* from mathematical first principles. The definition of algorithmic entropy in Section 8.6.2 and most results and examples are based on the treatment using continuous time by P. Gács, *Proc. 2nd IEEE Workshop on Physics and Computation*, 1994, 209–216. We have tried to simplify the discussion by discretizing time. We have considered a state space $\{0,1\}^\infty$. An analogous treatment can be given with the state space $\mathcal{R}$, the real numbers. Since this space has a different metric, we need to define appropriate notions of continuity, computablity, and semicomputability of functions over it. Let us look at some inherent distinctions between the $\mathcal{R}$ and $\{0,1\}^\infty$ spaces. Our space is $\{0,1\}^\infty$ with the μ-metric. In this space, the function $f(\omega) = 0$ for $\omega \in \Gamma_0$ and $f(\omega) = 1$ for $\omega \in \Gamma_1$ is a continuous computable function. This is different from the situation in case of the set of real numbers $\mathcal{R}$. If we take the space to be $\mathcal{R}$ with the μ-metric defined by the measure of all nonzero intervals (instead of just the cylinders), then the analogous function $f(r) = 0$ for $r < \frac{1}{2}$, and $f(r) = 1$ for $r \geq \frac{1}{2}$, is not continuous and not computable. Namely, however close we approximate $r = \frac{1}{2}$, we may never know whether $f(r) = 0$ or $f(r) = 1$.

Separating the information in an object into a part (or model) accounting for the *useful* information, the regularities, and a part describing the remaining *random* information was first proposed by Kolmogorov as the algorithmic sufficient statistic treated in Section 5.5.1. Related ideas are 'sophistication' in Exercise 5.5.20 on page 429 and in general the idea of two-part codes, Section 2.1.1, and the minimum description length (MDL) principle of Section 5.4. The latter approach is a statistical inference method to obtain the right hypothesis, or model, for a given data sample. Here 'right' means capturing the regular, or useful, aspects of the data. Similar ideas were proposed in [M. Gell-Mann, *The Quark and the Jaguar*, W.H. Freeman, New York, 1994; M. Gell-Mann, *Complexity*, 1:1(1995), 16–19; M. Gell-Mann and S. Lloyd, *Complexity*, 2:1(1996), 44–52] and applied to quantum-mechanics theory in [M. Gell-Mann and J.B. Hartle, *Proc. 4th Drexel Symp. Quantum Non-Integrability—The Quantum Classical Correspondence*, D.H. Feng and B.L. Hu, eds., 1997, pp. 3–35] and to the theory of adaptation and control in [S. Lloyd and J.J. Slotine, *Int. J. Adapt. Control Signal Process.*, 10(1996), 499]. The sum of the useful information and the random information is called 'total information.' It is another version of W.H. Zurek's [*Ibid.*] physical entropy, and is closely related to both the MDL principle (Section 5.4), and algorithmic entropy of Section 8.6.2.

[J. Ford, *Phys. Today*, (April 1983), 40–47; "Chaos: Solving the unsolvable, predicting the unpredictable," in *Chaotic Dynamics and Fractals*, M.F. Barnsley and S.G. Demko, eds., Academic Press, 1986] may

be the earliest paper applying Kolmogorov complexity to chaos. Use of Kolmogorov complexity and information theory to analyze quantum chaos related to issues in Section 8.6.2 are [R. Schack, G.M. D'Ariano, and C.M. Caves, *Phys. Rev. E*, 50(1994), 972]; and R. Schack and C. Caves [*Phys. Rev. E*, 53:4(1996), 3257–3270; *Phys. Rev. E*, 53:4(1996), 3387–3401]. The relation between chaos, Kolmogorov complexity, unpredictability, and instability is treated in [P.M.B. Vitányi, pp. 301–317 in *Kolmogorov's Heritage in Mathematics*, E. Charpentier, A. Lesne, and N.K. Nikolski, eds., Springer-Verlag, Berlin, 2007.]

Section 8.7 on quantum Kolmogorov complexity is based on the version using classical descriptions in [P.M.B. Vitányi,, *IEEE Trans. Inform. Theory*, 47:6(2001), 2464–2479; Correction, 48:4(2002), 1000]. Apart from this, there are two other main versions of quantum Kolmogorov complexity. In the exercises we discuss the version based on qubit descriptions [A. Berthiaume, W. van Dam, and S. Laplante, *J. Comput. Systems Sci.*, 63:2(2001), 201–221], and the version based on the universal lower semicomputable density matrix, forming a connection between the previous two approaches [P. Gács, *J. Phys. A: Math. Gen.*, 34(2001), 6859–6880]. Yet another version results from fixing the approximation precision (fidelity) in the classical description of quantum states [C.E. Mora and H.J. Briegel, *Phys. Rev. Lett.*, 95(2005), 200503; *Int. J. Quantum Information*, 4(2006), 715; (and with B. Kraus), Arxiv quant-ph/0610109]. For considerations of halting and universality of quantum Turing machines and incompressibility of quantum Kolmogorov complexity in the Berthiaume–van Dam–Laplante model see also [M. Müller, *Quantum Kolmogorov Complexity and the Quantum Turing Machine*, PhD thesis, Technische Univ. Berlin, Germany, 2007] and the references therein. An early paper mentioning quantum Kolmogorov complexity is [K. Zvozil, Quantum algorithmic information theory, Arxiv quant-ph/9510005, 1995].

The experiment with ants is reported by Zh.I. Reznikova and B.Ya. Ryabko [*Probl. Inform. Transmission*, 22(1986), 245–249; *Probl. Inform. Transmission*, 31:4(1995), 25–30] and [Zh.I. Reznikova, *Animal Intelligence: From Individual to Social Cognition*, Cambridge Univ. Press, 2007]. In a similar spirit, J. Feldman [*Nature*, 407:5(2000), 630–633] reported that human concept learning depends on the length of the shortest Boolean formulas expressing the concept. Further studies relating the concept of simplicity in psychology and cognition to Kolmogorov complexity can be found in [J. Feldman, *Curr. Directions Psychol. Sci.*, 12:6(2003), 227–232], N. Chater and P.M.B. Vitányi [*Trends Cognitive Sci.*, 7(1)(2003), 19–22; *J. Math. Psychology*, 47:3(2003), 346–369; *J. Math. Psychology*, 51:3(2007), 135–163], and [T.L. Griffiths, J.B. Tenenbaum, *Proc. 25th Ann. Conf. Cognitive Sci. Soc.*, 2003; *Cognition*, 103:2(2007), 180-226]. Francis Bacon is quoted from *Sylva Sylvarum*, 337, 1627.

References

[1] A.M. Abramov. Kolmogorov's pedagogic legacy. *Russian Math. Surveys*, 43(6):45–88, 1988.

[2] Y.S. Abu-Mostafa. The complexity of information extraction. *IEEE Trans. Inform. Theory*, 32(4):513–525, 1986.

[3] L. Adleman. Time, space, and randomness. Technical Report TM-131, MIT, Lab. Comput. Sci., March 1979.

[4] P. Adriaans and M. van Zaanen. Computational grammar induction for linguists. *Grammars*, 7:57–68, 2004.

[5] P. Adriaans and P.M.B. Vitányi. Approximation of the two-part MDL code. *IEEE Trans. Inform. Theory*. To appear.

[6] V.N. Agafonov. Normal sequences and finite automata. *Soviet Math. Dokl.*, 9:324–325, 1968.

[7] V.N. Agafonov. *On algorithms, frequency and randomness*. PhD thesis, University of Novosibirsk, Novosibirsk, 1970.

[8] G. Aggarwal, Q. Cheng, M.H. Goldwasser, M.Y. Kao, P. Moisset de Espanes, and R.T. Schweller. Complexities for generalized models of self-assembly. *SIAM J. Comput.*, 34:1493–1515, 2005.

[9] M. Agrawal, E. Allender, and S. Rudich. Reductions in circuit complexity: an isomorphism theorem and a gap theorem. *J. Comput. Syst. Sci.*, 57(2):127–143, 1998.

[10] P.S. Aleksandrov. A few words on A.N. Kolmogorov. *Russian Math. Surveys*, 38(4):5–7, 1983.

[11] V.M. Alekseev and M.V. Yakobson. Symbolic dynamics and hyperbolic dynamical systems. *Physics Reports*, 75:287–325, 1981.

[12] E. Allender. Some consequences of the existence of pseudorandom generators. *J. Comput. System Sci.*, 39:101–124, 1989.

[13] E. Allender. Applications of time-bounded Kolmogorov complexity in complexity theory. In O. Watanabe, editor, *Kolmogorov Complexity and Computational Complexity*, pages 4–22. Springer-Verlag, Berlin, 1992.

[14] E. Allender. When worlds collide: derandomization, lower bounds and Kolmogorov complexity. In *Proc. 21st Conf. Found. Software Technology Theor. Comput. Sci.*, volume 2245 of *Lect. Notes Comput. Sci.*, pages 1–15, Berlin, 2001. Springer-Verlag.

[15] E. Allender, H.M. Buhrman, and M. Koucký. What can be efficiently reduced to the Kolmogorov-random strings. *Annals Pure Applied Logic*, 138:2–19, 2006.

[16] E. Allender, H.M. Buhrman, M. Koucký, D. van Melkbeek, and D. Ronneburger. Power from random strings. In *Proc. 43rd IEEE Symp. Found. Comput. Sci.*, pages 669–678, New York, 2002. IEEE.

[17] E. Allender, M. Koucký, D. Ronneburger, and S. Roy. Derandomization and distinguishing complexity. In *Proc. 18th IEEE Conf. Comput. Complexity*, pages 209–220, 2003.

[18] E. Allender and R.S. Rubinstein. P-printable sets. *SIAM J. Comput.*, 17:1193–1202, 1988.

[19] E. Allender and O. Watanabe. Kolmogorov complexity and degrees of tally sets. *Inform. Comput.*, 86:160–178, 1990.

[20] A. Ambainis. Application of Kolmogorov complexity to inductive inference with limited memory. In *Proc. 6th Int. Workshop Algorithmic Learning Theory*, volume 997 of *Lect. Notes Artif. Intell.*, pages 313–318, Berlin, 1995. Springer-Verlag.

[21] K. Ambos-Spies and A. Kučera. Randomness in computability theory. In P. Cholak, S. Lempp, M. Lerman, and R.A. Shore, editors, *Computability Theory and Its Applications: Current Trends and Open Problems*, volume 257 of *Contemporary Mathematics*, pages 1–14. American Math. Society, 2000.

[22] M. Anand and L. Orlóci. Complexity in plant communities: the notion and quantification. *J. Theor. Biol.*, 179:179–186, 1996.

[23] C. Ané and M.J. Sanderson. Missing the forest for the trees: Phylogenetic compression and its implications for inferring complex evolutionary histories. *Systematic Biology*, 54(1):146–157, 2005.

[24] D. Angluin. Algorithmic theory of information and randomness. Lecture notes postgraduate course, Edinburgh University, 1977/1978.

[25] L. Antunes and L. Fortnow. Sophistication revisited. In *30th Int. Coll. Automata, Lang., Program.*, volume 2719 of *Lect. Notes Comput. Sci.*, pages 267–277, Berlin, 2003. Springer-Verlag.

[26] L. Antunes and L. Fortnow. Time-bounded universal distributions. *Electr. Coll. Comput. Complexity*, TR05-144, 2005.

[27] L. Antunes, L. Fortnow, A. Pinto, and A. Souto. Low-depth witnesses are easy to find. In *Proc. 22nd IEEE Conf. Comput. Complexity*, pages 46–51, 2007.

[28] L. Antunes, L. Fortnow, and D. van Melkebeek. Computational depth. In *Proc. IEEE Conf. Comput. Complexity*, pages 266–273, 2001.

[29] L. Antunes, L. Fortnow, D. van Melkebeek, and N.V. Vinodchandran. Computational depth: concepts and applications. *Theor. Comput. Sci.*, 354:391–404, 2006.

[30] L. Antunes, L. Fortnow, and N.V. Vinodchandran. Using depth to capture average-case complexity. In *Proc. 14th Int. Symp. Fundam. Com-*

put. Theory, volume 2751 of *Lect. Notes Comput. Sci.*, pages 303–310, Berlin, 2003. Springer-Verlag.

[31] B. Apolloni and C. Gentile. Sample size lower bounds in PAC learning by algorithmic complexity theory. *Theor. Comput. Sci.*, 209:141–162, 1998.

[32] F. Argenti, V. Benci, P. Cerrai, A. Cordelli, S. Galatolo, and G. Menconi. Information and dynamical systems: a concrete measurement on sporadic dynamics. *Chaos, Solutions and Fractals*, 13:461–469, 2002.

[33] A. Arning, R. Agrawal, and P. Raghavan. A linear method for deviation detection in large databases. In *Proc. 2nd Int. Conf. Knowledge Discov. Data Mining*, pages 164–169. ACM-SIGKDD, 1996.

[34] V.I. Arnol'd. A few words on Andrei Nikolaevich Kolmogorov. *Russian Math. Surveys*, 43(6):43–44, 1988.

[35] E.A. Asarin. Individual random continuous functions. In Yu.V. Prokhorov, editor, *Summaries of Reports of the First All-World Congress of the Bernouilli Society of Mathematical Statistics and Probability Theory*, volume 1, page 450. Nauka, 1986. In Russian.

[36] E.A. Asarin. Some properties of Kolmogorov δ-random finite sequences. *SIAM Theory Probab. Appl.*, 32:507–508, 1987.

[37] E.A. Asarin. On some properties of finite objects random in the algorithmic sense. *Soviet Math. Dokl.*, 36:109–112, 1988.

[38] E.A. Asarin and A.V. Pokrovskii. Application of Kolmogorov complexity to analyzing the dynamics of controlled systems. *Automat. and Telemekh.*, 1:25–33, 1986. In Russian.

[39] K.B. Athreya, J.M. Hitchcock, J.H. Lutz, and E. Mayordomo. Effective strong dimension in algorithmic information and computational complexity. *SIAM. J. Comput.*, 37:671–705, 2007.

[40] Y. Aumann, Y.Z. Ding, and M.O. Rabin. Everlasting security in the bounded storage model. *IEEE Trans. Inform. Theory*, 48(6):1668–1680, 2002.

[41] R.A. Baeza-Yates, R. Gavaldà, G. Navarro, and R. Scheihing. Bounding the length of least common subsequences and forests. *Theory Comput. Syst.*, 32(4):435–452, 1999.

[42] J.L. Balcázar and R.V. Book. Sets with small generalized Kolmogorov complexity. *Acta Informatica*, 23:679–688, 1986.

[43] J.L. Balcazar, H.M. Buhrman, and M. Hermo. Learnability of Kolmogorov-easy circuit expressions via queries. In P.M.B. Vitányi, editor, *Comput. Learning Theory; Proc. 2nd European Conf.*, volume 904 of *Lect. Notes Artificial Intelligence*, pages 112–124, Berlin, 1995. Springer-Verlag.

[44] J.L. Balcázar, J. Diaz, and J. Gabarró. *Structural Complexity*. Springer-Verlag, Berlin, 1988.

[45] J.L. Balcázar, R. Gavaldá, and H.T. Siegelmann. Computational power of neural networks: a characterization in terms of Kolmogorov complexity. *IEEE Trans. Inform. Theory*, 43(4):1175–1183, 1997.

[46] J.L. Balcázar and U. Schöning. Logarithmic advice classes. *Theor. Comput. Sci.*, 99:279–290, 1992.

[47] A.R. Barron and T.M. Cover. Minimum complexity density estimation. *IEEE Trans. Inform. Theory*, 37:1034–1054, 1991.

[48] A.R. Barron, J. Rissanen, and B. Yu. The minimum description length principle in coding and modeling. *IEEE Trans. Inform. Theory*, 44(6):2743–2760, 1998.

[49] J.M. Barzdins. Complexity of programs to determine whether natural numbers not greater than n belong to a recursively enumerable set. *Soviet Math. Dokl.*, 9:1251–1254, 1968.

[50] J.M. Barzdins. On computability by probabilistic machines. *Soviet Math. Dokl.*, 10:1464–1467, 1969.

[51] J.M. Barzdins. On the relative frequency of solution of algorithmically unsolvable mass problems. *Soviet Math. Dokl.*, 11:459–462, 1970.

[52] J.M. Barzdins. Algorithmic information theory. In *Encyclopaedia of Mathematics, volume 1*, pages 140–142. D. Reidel (Kluwer Academic Publishers), 1988. Updated and annotated translation of the *Soviet Mathematical Encyclopaedia*.

[53] J.M. Barzdins and R.V. Freivalds. On the prediction of general recursive functions. *Soviet Math. Dokl.*, 13:1251–1254 (1224–1228), 1972.

[54] M. Baumert, V. Baier, J. Hauersen, N. Wessel, U. Meyerfeldt, A. Schirdenwan, and A. Voss. Forecasting of life threatening arrhythmias using the compression entropy of heart rate. *Methods Information Medicine*, 43(2):202–206, 2004.

[55] Th. Bayes. An essay towards solving a problem in the doctrine of chances. *Philos. Trans. Royal Soc.*, 53:376–398, 1763. Ibid., 54:298-310, 1764, R. Price, ed.

[56] R. Beigel, H.M. Buhrman, P. Fejer, L. Fortnow, P. Grabowski, L. Longpré, A. Muchnik, F. Stephan, and L. Torenvliet. Enumerations of the Kolmogorov function. *Electr. Coll. Comput. Complexity*, 2004. TR04-015.

[57] R. Beigel, W. Gasarch, M. Li, and L. Zhang. Addition in $\log_2 n$ steps on average: a simple analysis. *Theor. Comput. Sci.*, 191(1-2):245–248, 1998.

[58] T. C. Bell, J. G. Cleary, and I. H. Witten. *Text Compression*. Prentice Hall, Englewood Cliffs, 1990.

[59] A.M. Ben-Amram and Z. Galil. On pointers versus addresses. *J. Assoc. Comput. Mach.*, 39(3):617–648, 1992.

[60] A.M. Ben-Amram and H. Petersen. Backing up in singly linked lists. In *Proc. 31st ACM Symp. Theory Comput.*, pages 780–786, 1999.

[61] F. Benatti, T. Krüger, Ra. Siegmund-Schultze, and A. Skola. Entropy and quantum Kolmogorov complexity. *Commun. Math. Phys.*, 265(2):437–461, 2006.

[62] D. Benedetto, E. Caglioti, and V. Loreto. Language trees and zipping. *Physical Review Letters*, 88(4):048702, 4 pages, 2002.

[63] C.H. Bennett. The thermodynamics of computation—a review. *Int. J. Theor. Physics*, 21:905–940, 1982.

[64] C.H. Bennett. Demons, engines and the second law. *Scientific American*, pages 108–116, Nov. 1987.

[65] C.H. Bennett. Dissipation, information, computational complexity and the definition of organization. In D. Pines, editor, *Emerging Syntheses in Science*, volume 1 of *Santa Fe Institute Studies in the Science of Complexity*, pages 297–313. Addison-Wesley, 1987.

[66] C.H. Bennett. Logical depth and physical complexity. In R. Herken, editor, *The Universal Turing Machine; A Half-Century Survey*, pages 227–258. Oxford University Press, 1988. In Germany: Kammerer & Unverzagt.

[67] C.H. Bennett. How to define complexity in physics, and why. In W.H. Zurek, editor, *Complexity, Entropy and the Physics of Information*, pages 137–148. Addison-Wesley, 1991.

[68] C.H. Bennett, P. Gács, M. Li, P.M.B. Vitányi, and W. Zurek. Information distance. *IEEE Trans. Inform. Theory*, 44(4):1407–1423, 1998.

[69] C.H. Bennett and M. Gardner. The random number omega bids fair to hold the mysteries of the universe. *Scientific American*, 241:20–34, November 1979.

[70] C.H. Bennett and R. Landauer. The fundamental physical limits of computation. *Scientific American*, 256(7):48–56, July 1985.

[71] C.H. Bennett, M. Li, and B. Ma. Chain letters and evolutionary histories. *Scientific American*, 288(6):76–81, 2003.

[72] C.H. Bennett and P.W. Shor. Quantum information theory. *IEEE Trans. Inform. Theory*, 44:2724–2742, 1998.

[73] T. Berger. *Rate Distortion Theory: A Mathematical Basis for Data Compression*. Prentice-Hall, Englewood Cliffs, NJ, 1971.

[74] A. Berthiaume, W. van Dam, and S. Laplante. Quantum Kolmogorov complexity. *J. Comput. Syst. Sci.*, 63(2):201–221, 2001.

[75] W. Bialek, I. Nemenman, and N. Tishby. Predictability, complexity, and learning. *Neural Computation*, 13:2409–2463, 2001.

[76] L. Bienvenue, W. Merkle, and A.K. Shen. A simple proof of the miller–yu theorem. *undamenta Informaticae*, 83:1–4, 2008.

[77] M. Blum. On the size of machines. *Inform. Contr.*, 11:257–265, 1967.

[78] M. Blum and O. Goldreich. Towards a computational theory of statistical tests. In *Proc. 33rd IEEE Symp. Found. Comput. Sci.*, pages 406–416, 1992.

[79] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *J. Assoc. Comput. Mach.*, 36(4):929–965, 1989.

[80] D.E. Boekee, R.A. Kraak, and E. Backer. On complexity and syntactic information. *IEEE Trans. Systems Man Cybernet.*, 12:71–79, 1982.

[81] N.N. Bogolyubov, B.V. Gnedenko, and S.L. Sobolev. Andrei Nikolaevich Kolmogorov (on his eightieth birthday). *Russian Math. Surveys*, 38(4):9–27, 1983.

[82] R.V. Book. On sets with small information content. In O. Watanabe, editor, *Kolmogorov Complexity and Computational Complexity*, pages 23–42. Springer-Verlag, Berlin, 1992.

[83] R.V. Book. On languages reducible to algorithmically random languages. *SIAM J. Comput.*, 23:1275–1282, 1994.

[84] R.V. Book and J.H. Lutz. On languages with very high space-bounded Kolmogorov complexity. *SIAM J. Comput.*, 22(2):395–402, 1993.

[85] R.V. Book, J.H. Lutz, and K. Wagner. An observation on probability versus randomness with applications to complexity classes. *Math. Systems Theory*, 27:201–209, 1994.

[86] R.V. Book and O. Watanabe. On random hard sets for NP. *Inform. Comput.*, 125:70–76, 1996.

[87] G. Boolos. A new proof of the Gödel incompleteness theorem. *Notices Amer. Math. Soc.*, 46:388–390, 1989.

[88] E. Borel. *Leçons sur la théorie des fonctions*. Gauthier-Villars, Paris, 3rd edition, 1927.

[89] S. Bose, L. Rallan, and V. Vedral. Communication capacity of quantum computation. *Phys. Rev. Lett.*, 85:5448–5451, 2000.

[90] B. Brejová. Analyzing variants of shellsort. *Inform. Process. Lett.*, 79(5):223–228, 2001.

[91] A.A. Brudno. On the complexity of paths of dynamic systems. *Russian Math. Surveys*, 33:197–198, 1978. Tranlation of Uspekhi Mat. Nauk.

[92] A.A. Brudno. Entropy and the complexity of trajectories of a dynamical system. *Trans. Mosc. Math. Soc.*, 44:127–151, 1983.

[93] H.M. Buhrman, L. Fortnow, and S. Laplante. Resource-bounded Kolmogorov complexity revisited. *SIAM J. Comput.*, 31(3):887–905, 2001.

[94] H.M. Buhrman, L. Fortnow, I. Newman, and N.K. Vereshchagin. Increasing Kolmogorov complexity. In *Proc. 22nd Symp. Theor. Aspects Comp. Sc.*, volume 3404 of *Lect. Notes. Comput. Sci.*, pages 412–421, Berlin, 2005. Springer-Verlag.

[95] H.M. Buhrman, E. Hemaspaandra, and L. Longpré. SPARSE reduces conjunctively to TALLY. *SIAM J. Comput.*, 24(4):673–681, 1995.

[96] H.M. Buhrman, J.H. Hoepman, and P.M.B. Vitányi. Space-efficient routing tables for almost all networks and the incompressibility method. *SIAM J. Comput.*, 28(4):1414–1432, 1999.

[97] H.M. Buhrman, T. Jiang, M. Li, and P.M.B. Vitányi. New applications of the incompressibility method: Part II. *Theor. Comput. Sci.*, 235(1):59–70, 2000.

[98] H.M. Buhrman, H. Klauck, N.K. Vereshchagin, and P.M.B. Vitányi. Individual communication complexity. *J. Comput. System Sci.*, 73:973–985, 2007.

[99] H.M. Buhrman, S. Laplante, and P. Miltersen. New bounds for the language compression problem. In *Proc. 15th IEEE Conf. Comput. Complexity*, pages 126–130, 2000.

[100] H.M. Buhrman, T. Lee, and D. van Melkebeek. Language compression and pseudorandom generators. *Computational Complexity*, 14:247–274, 2005.

[101] H.M. Buhrman, M. Li, J.T. Tromp, and P.M.B. Vitányi. Kolmogorov random graphs and the incompressibility method. *SIAM J. Comput.*, 29(2):590–599, 1999.

[102] H.M. Buhrman and L. Longpré. Compressibility and resource bounded measure. *SIAM J. Comput.*, 31(3):876–886, 2002.

[103] H.M. Buhrman and E. Mayordomo. An excursion to the Kolmogorov random strings. *J. Comput. Syst. Sci.*, 54(3):393–399, 1997.

[104] H.M. Buhrman and P. Orponen. Random strings make hard instances. *J. Comput. System Sci.*, 53(2):261–266, 1996.

[105] H.M. Buhrman and L. Torenvliet. Complicated complementations. In *Proc. 14th IEEE Conf. Comput. Complexity*, pages 227–236, 1999.

[106] H.M. Buhrman and L. Torenvliet. Randomness is hard. *SIAM J. Comput.*, 30(5):1485–1501, 2000.

[107] H.M. Buhrman, D. van Melkebeek, K.W. Regan, D. Sivakumar, and M. Strauss. A generalization of resource-bounded measure, with application to the BPP vs EXP problem. *SIAM J. Comput.*, 30(2):576–601, 2000.

[108] M.S. Burgin. Generalized Kolmogorov complexity and duality in computational theory. *Soviet Math. Dokl.*, 25(3):559–564, 1982.

[109] J.-Y. Cai and J. Hartmanis. On Hausdorff and topological dimensions of the Kolmogorov complexity of the real line. *J. Comput. System Sci.*, 49(3):605–619, 1994.

[110] J.-Y. Cai and L. Hemachandra. A note on enumerative counting. *Inform. Process. Lett.*, 38:215–219, 1991.

[111] C. Calude. *Information and Randomness: An Algorithmic Perspective.* Springer-Verlag, Berlin, 1994.

[112] C. Calude. A characterization of c.e. random reals. *Theor. Comput. Sci.*, 271(1-2):3–14, 2002.

[113] C. Calude, S. Marcus, and L. Staiger. A topological characterization of random sequences. *Inform. Process. Lett.*, 88:245–250, 2003.

[114] C.S. Calude. *Theories of Computational Complexity*, chapter 4. North-Holland, Amsterdam, 1988.

[115] C.S. Calude, I. Chitescu, and L. Staiger. P. Martin-Löf tests: representability and embedability. *Revue Roumaine Math. Pures Appl.*, 30:719–732, 1985.

[116] C.S. Calude, P.H. Hertling, B. Khoussainov, and Y. Wang. Recursively enumerable reals and Chaitin ω numbers. *Theor. Comput. Sci.*, 255:125–149, 2001.

[117] C.S. Calude and H. Jürgensen. Randomness as an invariant for number representations. In H. Maurer, J. Karhumaki, and G. Rozenberg, editors, *Results and Trends in Theoretical Computer Science*, pages 44–66. Springer-Verlag, Berlin, 1994.

[118] R. Carnap. *Logical Foundations of Probability.* Univ. Chicago Press, Chicago, 1950.

[119] J. Castro and J.L. Balcázar. Simple pac learning of simple decision lists. In *Proc. 6th Int. Workshop Algorithmic Learning Theory*, volume 997 of *Lect. Notes Artif. Intell.*, pages 239–248, Berlin, 1995. Springer-Verlag.

[120] J. Castro and D. Guijarro. PACs, simple-PAC and query learning. *Inform. Process. Lett.*, 73(1-2):11–16, 2000.

[121] C.M. Caves. Entropy and information: How much information is needed to assign a probability? In W.H. Zurek, editor, *Complexity, Entropy and the Physics of Information*, pages 91–115. Addison-Wesley, New York, 1991.

[122] C.M. Caves. Information, entropy and chaos. In J.J. Halliwell, J. Pérez-Mercader, and W.H. Zurek, editors, *Physical Origins of Time Asymmetry*, pages 47–89. Cambridge Univ. Press, Cambridge, England, 1994.

[123] M. Cebrián, M. Alfonseca, and A. Ortega. Common pitfalls using normalized compression distance: what to watch out for in a compressor. *Commun. Inform. Syst.*, 5(4):367–384, 2005.

[124] M. Cebrián, M. Alfonseca, and A. Ortega. The normalized compression distance is resistant to noise. *IEEE Trans. Inform. Theory*, 53(5):1895–1900, 2007.

[125] G.J. Chaitin. On the length of programs for computing finite binary sequences. *J. Assoc. Comput. Mach.*, 13:547–569, 1966.

[126] G.J. Chaitin. On the length of programs for computing finite binary sequences: statistical considerations. *J. Assoc. Comput. Mach.*, 16:145–159, 1969.

[127] G.J. Chaitin. On the simplicity and speed of programs for computing infinite sets of natural numbers. *J. Assoc. Comput. Mach.*, 16:407–422, 1969.

[128] G.J. Chaitin. On the difficulty of computations. *IEEE Trans. Inform. Theory*, 16:5–9, 1970.

[129] G.J. Chaitin. Computational complexity and Gödel's incompleteness theorem. *SIGACT News*, 9:11–12, 1971.

[130] G.J. Chaitin. Information-theoretic computational complexity. *IEEE Trans. Inform. Theory*, 20:10–15, 1974. Reprinted in T. Tymoczko, editor, *New Directions in the Philosophy of Mathematics*, Birkhäuser, 1986.

[131] G.J. Chaitin. Information-theoretic limitations of formal systems. *J. Assoc. Comput. Mach.*, 21:403–424, 1974.

[132] G.J. Chaitin. Randomness and mathematical proof. *Scientific American*, 232:47–52, May 1975.

[133] G.J. Chaitin. A theory of program size formally identical to information theory. *J. Assoc. Comput. Mach.*, 22:329–340, 1975.

[134] G.J. Chaitin. Algorithmic entropy of sets. *Comput. Math. Appl.*, 2:233–245, 1976.

[135] G.J. Chaitin. Information-theoretic characterizations of recursive infinite strings. *Theor. Comput. Sci.*, 2:45–48, 1976.

[136] G.J. Chaitin. Algorithmic information theory. *IBM J. Res. Develop.*, 21:350–359, 1977.

[137] G.J. Chaitin. Program size, oracles, and the jump operation. *Osaka J. Math.*, 14:139–149, 1977.

[138] G.J. Chaitin. Toward a mathematical definition of "life". In R.D. Levine and M. Tribus, editors, *The Maximal Entropy Formalism*, pages 477–498. MIT Press, 1979.

[139] G.J. Chaitin. Algorithmic information theory. In *Encyclopedia of Statistical Sciences, volume 1*, pages 38–41. Wiley, 1982.

[140] G.J. Chaitin. Gödel's theorem and information. *Int. J. Theor. Physics*, 22:941–954, 1982. Reprinted in T. Tymoczko, editor, *New Directions in the Philosophy of Mathematics*, Birkhäuser, Boston, 1986.

[141] G.J. Chaitin. Randomness and Gödel's theorem. *Mondes en Developement*, 14, No. 54-55:125–128, 356, 1986.

[142] G.J. Chaitin. *Algorithmic Information Theory*. Cambridge Univ. Press, Cambridge, England, 1987.

[143] G.J. Chaitin. Beyond Gödel's proof. *IBM Res. Magazine*, 25:12–15, Fall 1987.

[144] G.J. Chaitin. Computing the busy beaver function. In T.M. Cover and B. Gopinath, editors, *Open Problems in Communication and Computation*, pages 108–112. Springer-Verlag, 1987.

[145] G.J. Chaitin. Incompleteness theorems for random reals. *Advances in Applied Math.*, 8:119–146, 1987.

[146] G.J. Chaitin. *Information, Randomness and Incompleteness—Papers on Algorithmic Information Theory*. World Scientific, Singapore, 1987.

[147] G.J. Chaitin. An algebraic equation for the halting probability. In R. Herken, editor, *The Universal Turing Machine; A Half-Century Survey*, pages 279–284. Oxford University Press, 1988. In Germany: Kammerer & Unverzagt.

[148] G.J. Chaitin. *Information-Theoretic Incompleteness*. World Scientific, Singapore, 1992.

[149] G.J. Chaitin. Information-theoretic incompleteness. *Applied Mathematics and Computation*, 52:83–101, 1992.

[150] G.J. Chaitin. On the number of n-bit strings with maximum complexity. *Applied Mathematics and Computation*, 59:97–100, 1993.

[151] G.J. Chaitin. A new version of algorithmic information theory. *Complexity*, 1(4):55–59, 1995/1996.

[152] G.J. Chaitin and J.T. Schwartz. A note on Monte-Carlo primality tests and algorithmic information theory. *Comm. Pure Applied Math.*, 31:521–527, 1978.

[153] D.G. Champernowne. The construction of decimals normal in the scale of ten. *J. London Math. Soc.*, 8:254–260, 1933.

[154] M. Charikar, E. Lehman, D. Liu, R. Panigrahy, M. Prabhakaran, A. Rasala, A. Sahai, and A. Shelat. The smallest grammar. *IEEE Trans. Inform. Theory*, 51(7):2554–2576, 2005.

[155] N. Chater. Reconciling simplicity and likelihood principles in perceptual organization. *Psychological Review*, 103:566–581, 1996.

[156] N. Chater. The search for simplicity: a fundamental cognitive principle? *The Quarterly J. Exper. Psych.: Sect. A*, 52(2):273–302, 1999.

[157] N. Chater. Cognitive science: the logic of human learning. *Nature*, 407:572–573, 2000.

[158] N. Chater and P.M.B. Vitányi. The generalized universal law of generalization. *J. Math Psychology*, 47(3):346–369, 2003.

[159] N. Chater and P.M.B. Vitányi. Simplicity: A unifying principle in cognitive science? *Trends in Cognitive Sciences*, 7(1):19–22, 2003.

[160] N. Chater and P.M.B. Vitányi. "Ideal learning" of natural language: positive results about learning from positive evidence. *J. Math Psychology*, 51(3):135–206, 2007.

[161] N. Chater, P.M.B. Vitányi, and N. Steward. Universal generalization and universal inter-item confusability. *Behavior and Brain Sciences*, 24(4):559–660, 2001.

[162] F. Chen, J. Xu, F. Gu, X. Yu, X. Meng, and Z. Qiu. Dynamic process of information transmission complexity in human brains. *Biol. Cybernetics*, 83:355–366, 2000.

[163] X. Chen, B. Francia, M. Li, B. Mckinnon, and A. Seker. Shared information and program plagiarism detection. *IEEE Trans. Inform. Theory*, 50(7):1545–1550, 2004.

[164] X. Chen, S. Kwong, and M. Li. A compression algorithm for DNA sequences. *IEEE Eng. Med. Biol. Magaz.*, 20(4):61–66, 2001.

[165] Q. Cheng and F. Fang. Kolmogorov random graphs only have trivial stable colorings. *Inform. Process. Lett.*, 81(3):133–136, 2002.

[166] H. Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sums of observations. *Ann. Math. Stat.*, 23:493–509, 1952.

[167] A.V. Chernov, An.A. Muchnik, A.E. Romashchenko, A.K. Shen, and N.K. Vereshchagin. Upper semi-lattice of binary strings with the relation "x is simple conditional to y". *Theor. Comput. Sci.*, 271(1-2):69–95, 2002.

[168] I. Chlamtac and A. Farago. A new approach to the design and analysis of peer-to-peer mobile networks. *Wireless Networks*, 5:149–156, 1999.

[169] M. Chrobak and M. Li. $k + 1$ heads are better than k for PDAs. *J. Comput. System Sci.*, 37:144–155, 1988.

[170] F.R.K. Chung, R.E. Tarjan, W.J. Paul, and R. Reischuk. Coding strings by pairs of strings. *SIAM J. Algebra Discrete Math.*, 6(3):445–461, 1985.

[171] A. Church. On the concept of a random sequence. *Bull. Amer. Math. Soc.*, 46:130–135, 1940.

[172] R.L. Cilibrasi. The complearn toolkit. *www.complearn.org*, 2003.

[173] R.L. Cilibrasi and P.M.B. Vitányi. Clustering by compression. *IEEE Trans. Inform. Theory*, 51(4):1523–1545, 2005.

[174] R.L. Cilibrasi and P.M.B. Vitányi. Similarity of objects and the meaning of words. In *Proc. 3rd Conf. Theory Appl. Models Comput.*, volume 3959 of *Lect. Notes. Comput. Sci.*, pages 21–45, Berlin, 2006. Springer-Verlag.

[175] R.L. Cilibrasi and P.M.B. Vitányi. The Google similarity distance. *IEEE Trans. Knowledge and Data Engineering*, 19(3):370–383, 2007.

[176] R.L. Cilibrasi, P.M.B. Vitányi, and R. de Wolf. Algorithmic clustering of music based on string compression. *Computer Music J.*, 28(4):49–67, 2004.

[177] J.T. Coffey and R.M. Goodman. Any code of which we cannot think is good. *IEEE Trans. Inform. Theory*, 36:1453–1461, 1990.

[178] J.D. Collier. Two faces of Maxwell's demon reveal the nature of irreversibility. *Stud. Hist. Phil. Sci.*, 21(2):257–268, 1990.

[179] J.D. Collier. Information originates in symmetry breaking. *Symmetry: Culture and Science*, 7:247–256, 1996.

[180] T.M. Cover. Enumerative source encoding. *IEEE Trans. Inform. Theory*, 19:73–77, 1973.

[181] T.M. Cover. Generalization on patterns using Kolmogorov complexity. In *Proc. 1st Int. Conf. Pattern Recognition*, pages 551–553, 1973.

[182] T.M. Cover. On the determination of the irrationality of the mean of a random variable. *Ann. Statist.*, 1:862–871, 1973.

[183] T.M. Cover. Universal gambling schemes and the complexity measures of Kolmogorov and Chaitin. Technical Report 12, Statistics Department, Stanford University, October 1974.

[184] T.M. Cover. Kolmogorov complexity, data compression, and inference. In J.K. Skwirzynski, editor, *The Impact of Processing Techniques on Communications*, pages 23–33. Martinus Nijhoff Publishers, The Hague, Netherlands, 1985.

[185] T.M. Cover, P. Gács, and R.M. Gray. Kolmogorov's contributions to information theory and algorithmic complexity. *Ann. Probab.*, 17:840–865, 1989.

[186] T.M. Cover and R.C. King. A convergent gambling estimate of the entropy of English. *IEEE Trans. Inform. Theory*, 24:413–421, 1978.
[187] T.M. Cover and J.A. Thomas. *Elements of Information Theory*. Wiley & Sons, New York, 1991.
[188] J.P. Crutchfield and D.P. Feldman. Statistical complexity of simple one-dimensional spin systems. *Phys. Rev. E*, 55(2):1239–1242, 1997.
[189] J.P. Crutchfield and K. Young. Computation at the onset of chaos. In W.H. Zurek, editor, *Complexity, Entropy and the Physics of Information*, pages 223–269. Addison-Wesley, New York, 1991.
[190] R.R. Cuykendall. *Kolmogorov information and VLSI lower bounds*. PhD thesis, University of California, Los Angeles, Dec. 1984.
[191] R.P. Daley. Complexity and randomness. In R. Rustin, editor, *Computational Complexity; Courant Comput. Sci. Symp. 7*, pages 113–122, New York, 1971. Algorithmics Press.
[192] R.P. Daley. An example of information and computation resource tradeoff. *J. Assoc. Comput. Mach.*, 20(4):687–695, 1973.
[193] R.P. Daley. Minimal-program complexity of sequences with restricted resources. *Inform. Contr.*, 23:301–312, 1973.
[194] R.P. Daley. The extent and density of sequences within the minimum-program complexity hierarchies. *J. Comput. System Sci.*, 9:151–163, 1974.
[195] R.P. Daley. Minimal-program complexity of pseudo-recursive and pseudo-random sequences. *Math. Systems Theory*, 9:83–94, 1975.
[196] R.P. Daley. Noncomplex sequences: characterizations and examples. *J. Symbolic Logic*, 41:626–638, 1976.
[197] R.P. Daley. On the inference of optimal descriptions. *Theor. Comput. Sci.*, 4:301–319, 1977.
[198] R.P. Daley. Quantitative and qualitative information in computation. *Inform. Contr.*, 45:236–244, 1980.
[199] R.P. Daley. The process complexity and the understanding of sequences. In *Proc. Symp. Summer School MFCS*, High Tatras, September 1973.
[200] J. Daugman. The importance of being random: statistical principles of iris recognition. *Pattern Recognition*, 36:279–291, 2003.
[201] G. Davie. Characterising the Martin-Löf random sequences using computably enumberable sets of measure one. *Inform. Process. Lett.*, 92(3):157–160, 2004.
[202] P.C.W. Davies. Why is the physical world so comprehensible? In W.H. Zurek, editor, *Complexity, Entropy and the Physics of Information*, pages 61–70. Addison-Wesley, New York, 1991.
[203] L.D. Davisson. Universal noiseless encoding. *IEEE Trans. Inform. Theory*, 19:783–795, 1973.
[204] A.P. Dawid. Discussion of papers by Rissanen and by Wallace and Dowe. *Comput. J.*, 42(4):323–326, 1999. Discussions and rejoinders from both camps can be found in the same issue.
[205] B. de Finetti. *Probability, Induction, and Statistics*. Wiley, 1972.
[206] E.D. Demaine and A. Lopez-Ortiz. A linear lower bound on index size for text retrieval. *J. Algorithms*, 48(1):2–15, 2003.
[207] F. Denis. Learning regular languages from simple positive examples. *Machine Learning*, 44(1-2):37–66, 2001.

[208] F. Denis, C. D'Halluin, and R. Gilleron. PAC learning with simple examples. In *Proc. 13th Symp. Theor. Aspects Comput. Sci.*, volume 1046 of *Lect. Notes Comput. Sci.*, pages 231–242, Berlin, 1996. Springer-Verlag.

[209] F. Denis and R. Gilleron. PAC learning under helpful distributions. *Theor. Informatics Applications*, 35:129–148, 2001.

[210] A. DeSantis, G. Markowsky, and M.N. Wegman. Learning probabilistic prediction functions. In *Proc. 29th IEEE Symp. Found. Comput. Sci.*, pages 110–119, 1988.

[211] T.G. Dewey. Algorithmic complexity of a protein. *Phys. Review E*, 54(1):R39–R41, 1996.

[212] J.-E. Dies. Information et complexité. *Ann. Inst. Henri Poincaré, B*, 12:365–390, 1976. Ibid., 14:113-118, 1978.

[213] M. Dietzfelbinger. *Lower bounds on computation time for various models in computational complexity theory*. PhD thesis, Dept. Comput. Sci., Univ. Illinois at Chicago, 1987.

[214] M. Dietzfelbinger. The speed of copying on one-tape off-line Turing machines. *Inform. Process. Lett.*, 33:83–90, 1989/1990.

[215] M. Dietzfelbinger and W. Maass. The complexity of matrix transposition on one-tape off-line Turing machines with output tape. *Theor. Comput. Sci.*, 108(2):271–290, 1993.

[216] M. Dietzfelbinger, W. Maass, and G. Schnitger. The complexity of matrix transposition on one-tape off-line Turing machines. *Theor. Comput. Sci.*, 82(1):113–129, 1991.

[217] D. Ding and L. Yu. There is no SW-complete c.e. real. *J. Symb. Logic*, 69(4):1163–1170, 2004.

[218] D. Donoho. The Kolmogorov sampler. *Ann. Statistics*, Submitted.

[219] J.L. Doob. Kolmogorov's early work on convergence theory and foundations. *Ann. Probab.*, 17:815–821, 1989.

[220] R. Downey and D.R. Hirschfeldt. *Algorithmic Randomness and Complexity*. Springer-Verlag, New York, To appear.

[221] R.G. Downey, D.R. Hirschfeldt, and G. LaForte. Randomness and reducibility. *J. Comput. Syst. Sci.*, 68(1):96–114, 2004.

[222] R.G. Downey, D.R. Hirschfeldt, and A. Nies. Randomness, computability, and density. *SIAM J. Comput.*, 31(4):1169–1183, 2002.

[223] R.G. Downey, D.R. Hirschfeldt, A. Nies, and F. Stephan. Trivial reals. In *Proc. 7th and 8th Asian Logic Confs*, pages 103–131. Singapore Univ. Press, 2003.

[224] J.C. Dubacq, B. Durand, and E. Formenti. Kolmogorov complexity and cellular automata classification. *Theor. Comput. Sci.*, 259:271–285, 2001.

[225] B. Durand, L. Levin, and A.K. Shen. Complex tilings. In *Proc. 33rd ACM Symp. Theory Comput.*, pages 732–739, 2001.

[226] B. Durand and S. Porrot. Comparison between the complexity of a function and the complexity of its graph. *Theor. Comput. Sci.*, 271(1-2):37–46, 2002.

[227] B. Durand, A.K. Shen, and N.K. Vereshchagin. Descriptive complexity of computable sequences. *Theor. Comput. Sci.*, 271(1-2):47–58, 2002.

[228] B. Durand and N.K. Vereshchagin. Kolmogorov-Loveland stochasticity for finite strings. *Inform. Process. Lett.*, 91:263–269, 2004.

[229] P. Duris, Z. Galil, W.J. Paul, and R. Reischuk. Two nonlinear lower bounds for on-line computations. *Inform. Contr.*, 60:1–11, 1984.
[230] E.B. Dynkin. Kolmogorov and the theory of Markov processes. *Ann. Probab.*, 17:822–832, 1989.
[231] V.D. Dzhunushaliev. Kolmogorov algorithmic complexity and its probability interpretation in quantum gravity. *Class. Quantum Grav.*, 15:603–612, 1998.
[232] T. Ebert, W. Merkle, and H. Vollmer. On the autoreducibility of random sequences. *SIAM J. Comput.*, 32(6):1542–1569, 2003.
[233] P. Erdős and J. Spencer. *Probabilistic Methods in Combinatorics*. Academic Press, 1974.
[234] S.C. Evans and B. Barnett. Network security through conservation of complexity. In *Proc. IEEE Military Communications Conf.*, pages 1133–1138, 2002.
[235] S.C. Evans, S.F. Bush, and J. Hershey. Information assurance through Kolmogorov complexity. In *Proc. DARPA Information Survivability Conf. and Exposition*, volume 2, pages 1322–1331, 2001.
[236] R. Falk and C. Konold. Making sense of randomness: implicit encoding as a basis for judgement. *Psychological Rev.*, 104(2):301–318, 1997.
[237] M. Feder. Maximal entropy as special case of the minimum length description criterion. *IEEE Trans. Inform. Theory*, 32:847–849, 1986.
[238] J. Feigenbaum, L. Fortnow, S. Laplante, and A. Naik. On coherence, random-self-reducibility, and self-correction. *Comput. Complexity*, 7:174–191, 1998.
[239] J. Feldman. Minimization of Boolean complexity in human concept learning. *Nature*, 407:630–633, 2000.
[240] J. Feldman. The simplicity principle in human concept learning. *Current Directions in Psychological Science*, 12(6):227–232, 2003.
[241] J.H. Felker. A link between information and energy. *Proc. IRE*, 40:728–729, 1952. Discussion, *Ibid.*, 52(1954), 1191.
[242] W. Feller. *An Introduction to Probability Theory and Its Applications*, volume 1. Wiley, 3rd edition, 1968.
[243] S. Fenner, L. Fortnow, S.A. Kurtz, and L. Li. An oracle builder's toolkit. *Inform. Comput.*, 182(2):95–136, 2003.
[244] P. Ferragina, R. Giancarlo, V. Greco, and G. Manzini end G. Valiente. Compression-based classification of biological sequences and structures via the Universal Similarity Metric: experimental assessment. *BMC Bioinformatics*, 8(1):doi:10.1186/1471–2105–8–252, 2007.
[245] T.L. Fine. On the apparent convergence of relative frequencies and its implications. *IEEE Trans. Inform. Theory*, 16:251–257, 1970.
[246] T.L. Fine. *Theories of Probability*. Academic Press, 1973.
[247] T.L. Fine. Uniformly reasonable source encoding is often practically impossible. *IEEE Trans. Inform. Theory*, 21:368–373, 1975.
[248] R.A. Fisher. On the mathematical foundations of theoretical statistics. *Philos. Trans. Royal Soc. London, Ser. A*, 222:309–368, 1922.
[249] P. Flocchini, E. Kranakis, D. Krizanc, F.L. Luccio, and N. Santoro. Sorting multisets in anonymous rings. *J. Parall. Distrib. Comput.*, 64(2):254–265, 2004.

[250] J. Ford. How random is a random coin toss? *Physics Today*, 36:40–47, April 1983.

[251] J. Ford. Chaos: solving the unsolvable, predicting the unpredictable. In M.F. Barnsley and S.G. Demko, editors, *Chaotic Dynamics and Fractals*. Academic Press, 1986.

[252] L. Fortnow and M. Kummer. Resource-bounded instance complexity. *Theor. Comput. Sci. A*, 161:123–140, 1996.

[253] L. Fortnow and S. Laplante. Circuit complexity à la Kolmogorov. *Inform. Comput.*, 123:121–126, 1995.

[254] L. Fortnow and S. Laplante. Nearly optimal language compression using extractors. In *Proc. 15th Symp. Theor. Aspects Comput. Sci.*, volume 1373 of *Lect. Notes Comput. Sci.*, pages 84–93, Berlin, 1998. Springer-Verlag.

[255] L. Fortnow, T. Lee, and N.K. Vereshchagin. Kolmogorov complexity with error. In *Symp. Theor. Aspects Comput. Sci.*, volume 3884 of *Lect. Notes Comp. Sci.*, pages 137–148, Berlin, 2006. Springer-Verlag.

[256] L. Fortnow and J.H. Lutz. Prediction and dimension. *J. Comput. Syst. Sci.*, 70:570–589, 2005.

[257] L. Fortnow, A. Pavan, and A.L. Selman. Distributionally hard languages. *Theory Comput. Syst.*, 34:245–261, 2001.

[258] W.L. Fouché. Identifying randomness given by high descriptive complexity. *Acta Applicandae Mathematicae*, 34:313–328, 1994.

[259] W.L. Fouché. Descriptive complexity and reflective properties of combinatorial configurations. *J. London Math. Soc.*, 54(2):199–208, 1996.

[260] R.V. Freivalds. On the running time of deterministic and nondeterministic Turing machines. *Latv. Mat. Ezhegodnik*, 23:158–165, 1979. In Russian.

[261] R.V. Freivalds and M. Karpinski. Lower time bounds for randomized computation. In *Proc. 22nd Int. Colloq. Automata, Lang. Prog.*, volume 944 of *Lect. Notes Comput. Sci.*, pages 183–195, Berlin, 1995. Springer-Verlag.

[262] B. Fu. With quasi-linear queries, EXP is not polynomial-time Turing reducible to sparse sets. *SIAM J. Comput.*, 24(5):1082–1090, 1995.

[263] P. Gács. On the symmetry of algorithmic information. *Soviet Math. Dokl.*, 15:1477–1480, 1974. Correction, *Ibid.*, 15:1480, 1974.

[264] P. Gács. *Komplexität und Zufälligkeit*. PhD thesis, Fachbereich Mathematik, J.W. Goethe Universität, Frankfurt am Main, 1978.

[265] P. Gács. Exact expressions for some randomness tests. *Z. Math. Logik Grundl. Math.*, 26:385–394, 1980.

[266] P. Gács. On the relation between descriptional complexity and algorithmic probability. *Theor. Comput. Sci.*, 22:71–93, 1983.

[267] P. Gács. Every sequence is reducible to a random sequence. *Inform. Contr.*, 70:186–192, 1986. See also: A. Kučera, Measure, Π^0_1-classes and complete extensions of PA, pp. 245–259 in: H.D. Ebbinghaus, G.H. Müller, and G.E. Sacks, editors, *Recursion Theory Week*, Lect. Notes Math., Vol. 1141, Springer Verlag, Berlin, 1985.

[268] P. Gács. Randomness and probability—complexity of description. In Kotz-Johnson, editor, *Encyclopedia of Statistical Sciences*, volume 7, pages 551–555. Wiley, 1986.

[269] P. Gács. Lecture notes on descriptional complexity and randomness. Technical report, Comput. Sci. Dept., Boston Univ., 1988–2007.
[270] P. Gács. Gregory J. Chaitin, Algorithmic Information Theory. *J. Symbolic Logic*, 54:624–627, 1989. Book review.
[271] P. Gács. The Boltzmann entropy and randomness tests. In *Proc. 2nd IEEE Workshop Phys. Comput.*, pages 209–216, 1994.
[272] P. Gács. Quantum algorithmic entropy. *J. Phys. A: Math. Gen.*, 34:6859–6880, 2001.
[273] P. Gács and J. Korner. Common information is far less than mutual information. *Problems of Control and Inform. Theory*, 2:149–162, 1973.
[274] P. Gács, J. Tromp, and P.M.B. Vitányi. Algorithmic statistics. *IEEE Trans. Inform. Theory*, 47(6):2443–2463, 2001. Correction, *Ibid.*, 48(8): 2427, 2002.
[275] H. Gaifman and M. Snir. Probabilities over rich languages, randomness and testing. *J. Symbolic Logic*, 47:495–548, 1982.
[276] Z. Galil, R. Kannan, and E. Szemerédi. On 3-pushdown graphs with large separators. *Combinatorica*, 9:9–19, 1989.
[277] Z. Galil, R. Kannan, and E. Szemerédi. On nontrivial separators for k-page graphs and simulations by nondeterministic one-tape Turing machines. *J. Comput. System Sci.*, 38:134–149, 1989.
[278] Z. Galil and J. Seiferas. Time-space-optimal string matching. *J. Comput. System Sci.*, 26:3:280–294, 1983.
[279] R.G. Gallager. *Information Theory and Reliable Communication*. Wiley, 1968.
[280] A. Gammerman and V.G. Vovk. Kolmogorov complexity: sources, theory and applications. *Comput. J.*, 42:252–255, 1999.
[281] A. Gammerman and V.G. Vovk. Prediction algorithms and confidence measures based on algorithmic randomness theory. *Theor. Comput. Sci.*, 287(1):209–217, 2002.
[282] Q. Gao, M. Li, and P.M.B. Vitányi. Applying MDL to learning best model granularity. *Artificial Intelligence*, 121(1-2):1–29, 2000.
[283] M.R. Garey and D.S. Johnson. *Computers and Intractability, a Guide to the Theory of NP-Completeness*. W.H. Freeman and Co., 1979.
[284] P. Gaspard and X.-J. Wang. Sporadicity: Between periodic and chaotic dynamical behaviors. *Proc. Nat'l Acad. Sci. USA*, 85:4591–4595, 1988.
[285] R. Gavaldá. *Kolmogorov Randomness and Its Applications to Structural Complexity Theory*. PhD thesis, Universitat Politécnica de Catalunya, Barcelona, 1992.
[286] R. Gavaldá, L. Torenvliet, O. Watanabe, and J.L. Balcázar. Generalized Kolmogorov complexity in relativized separations. In *Proc. 15th Conf. Math. Found. Comput. Sci.*, volume 452 of *Lect. Notes Comput. Sci.*, pages 269–276, Berlin, 1991. Springer-Verlag.
[287] R. Gavaldá and O. Watanabe. On the computational complexity of small descriptions. *SIAM J. Comput.*, 22(6):1257–1275, 1993.
[288] M. Gell-Mann. *The Quark and the Jaguar*. W.H. Freeman, New York, 1994.
[289] M. Gell-Mann. Remarks on simplicity and complexity. *Complexity*, 1(1):16–19, 1995.

[290] M. Gell-Mann and J.B. Hartle. Strong decoherence. In D.H. Feng and B.L. Hu, editors, *Proc. 4th Drexel Symp. Quantum Non-Integrability – Quantum Classical Correspondence.*, pages 3–35. International Press, Cambridge, Mass., 1997.

[291] M. Gell-Mann and S. Lloyd. Information measures, effective complexity, and total information. *Complexity*, 2(1):44–52, 1996.

[292] M. Geréb-Graus and M. Li. Three one-way heads cannot do string matching. *J. Comput. System Sci.*, 48:1–8, 1994.

[293] B.V. Gnedenko. Andrei Nikolaevich Kolmogorov (on the occasion of his seventieth birthday). *Russian Math. Surveys*, 28(5):5–16, 1973.

[294] S. Goel and S. Bush. Kolmogorov complexity estimates for detection of viruses in biologically inspired security systems: a comparison with traditional approaches. *Complexity Journal*, 9(2), 2003.

[295] E.M. Gold. Language identification in the limit. *Inform. Contr.*, 10:447–474, 1967.

[296] A. Goldberg and M. Sipser. Compression and ranking. *SIAM J. Comput.*, 20:524–536, 1991.

[297] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *J. Assoc. Comput. Mach.*, 33:792–807, 1986.

[298] P. Grassberger. Towards a quantitative theory of self-generated complexity. *Int. J. Theor. Physics*, 25(9):907–938, 1986.

[299] T.L. Griffiths and J.B. Tenenbaum. From algorithmic to subjective randomness. In *Advances in Neural Information Processing Systems 16*. NIPS 2003, MIT Press, 2004.

[300] T.L. Griffiths and J.B. Tenenbaum. From mere coincidences to meaningful discoveries. *Cognition*, 103(2):180–226, 2007.

[301] R.I. Grigorchuk. A connection between algorithmic problems and entropy characteristics of groups. *Soviet Math. Dokl.*, 32:356–360, 1985.

[302] S. Grigorieff and J.Y. Marion. Kolmogorov complexity and nondeterminism. *Theor. Comput. Sci.*, 271(1-2):151–180, 2002.

[303] P. Grünwald. A minimum description length approach to grammar inference. In *Lect. Notes Artif. Intell.*, volume 1040, pages 203–216, 1996.

[304] P.D. Grünwald. *The Minimum Description Length Principle*. MIT Press, 2007.

[305] P.D. Grünwald, J. Myung, and M.A. Pitt, editors. *Advances in Minimum Description Length: Theory and Applications*. MIT Press, 2005.

[306] P.D. Grünwald and P.M.B. Vitányi. Kolmogorov complexity and information theory. With an interpretation in terms of questions and answers. *J. Logic, Lang., Inform.*, 12(4):497–529, 2003.

[307] Y. Gurevich. The logic of computer science column. *EATCS Bulletin*, 35:71–82, June 1988.

[308] V.G. Gurzadyan. Kolmogorov complexity as a descriptor of cosmic microwave background maps. *Europhysics Lett.*, 46(1):114–117, 1999.

[309] S. Gutmann. Using classical probability to guarantee properties of infinite quantum sequences. *Phys. Rev. A*, 52(5):3560–3562, 1995.

[310] U. Hahn and N. Chater. Concepts and similarity. In K. Lamberts and D. Shanks, editors, *Knowledge, Concepts and Categories*, pages 43–92. Psychology Press/MIT Press, 1997.

[311] U. Hahn, N. Chater, and L.B. Richardson. Similarity as transformation. *Cognition*, 87:1–32, 2003.
[312] D. Hammer and A.K. Shen. A strange application of Kolmogorov complexity. *Theory Comput. Systems*, 31(1):1–4, 1998.
[313] Y. Han and L.A. Hemaspaandra. Pseudorandom generators and the frequency of simplicity. *J. Cryptology*, 9(4):251–261, 1996.
[314] J.B. Hartle. Quantum pasts and utility of history. *Phys. Scr.*, T76:67–77, 1998.
[315] J. Hartmanis. Generalized Kolmogorov complexity and the structure of feasible computations. In *Proc. 24th IEEE Symp. Found. Comput. Sci.*, pages 439–445, 1983.
[316] J. Hartmanis and L. Hemachandra. On sparse oracles separating feasible complexity classes. *Inform. Process. Lett.*, 28:291–295, 1988.
[317] J. Hartmanis and R.E. Stearns. On the computational complexity of algorithms. *Trans. Amer. Math. Soc.*, 117:285–306, 1969.
[318] R. Heim. On the algorithmic foundation of information theory. *IEEE Trans. Inform. Theory*, 25:557–566, 1979.
[319] L. Hemachandra and S. Rudich. On the complexity of ranking. *J. Comput. System Sci.*, 41:2:251–271, 1990.
[320] L. Hemachandra and G. Wechsung. Kolmogorov characterizations of complexity classes. *Theor. Comput. Sci.*, 83(2):313–322, 1991.
[321] M. Hermo. Compressibility and uniform complexity. *Inform. Process. Lett.*, 62(5):256–264, 1997.
[322] M. Hermo and E. Mayordomo. A note on polynomial size circuits with low resource-bounded Kolmogorov complexity. *Math. Systems Theory*, 27:247–356, 1994.
[323] P. Hertling and K. Weihrauch. Random elements in effective topological spaces with measure. *Inform. Computation*, 181(1):32–56, 2003.
[324] J.M. Hitchcock, M. López-Valdés, and E. Mayordomo. Scaled dimension and the Kolmogorov complexity. In *Proc. 29th Int. Symp. Math. Found. Comput. Sci.*, volume 3153 of *Lect. Notes Comput. Sci.*, pages 476–487, Berlin, 2004. Springer-Verlag.
[325] J.M. Hitchcock and J.H. Lutz. Why computational complexity requires stricter martingales. *Theory Comput. Syst.*, 39(2):277–296, 2006.
[326] J.M. Hitchcock and N. V. Vinodchandran. Dimension, entropy rates, and compression. *J. Comput. Syst. Sci.*, 72(4):760–782, 2006.
[327] G. Hotz. Komplexität als Kriterium in der Theorienbildung. *Akademie der Wissenschaften und der Literatur (Mainz)/Abhandlungen Mathematisch-Naturwissenschaftliche Klasse*, 1, 1988. Steiner-Verlag, Wiesbaden.
[328] T. Housel and V.A. Kanevsky. Re-engineering business processes: a complexity theory approach. *Inform. Syst. Operations Res.*, 33(4), 1995.
[329] D.A. Huffman. A method for construction of minimum-redundancy codes. *Proceedings IRE*, 40:1098–1101, 1952.
[330] M. Hühne. Linear speed-up does not hold on Turing machines with tree storages. *Inform. Process. Lett.*, 47(6):313–318, 1993.
[331] M. Hühne. On the power of several queues. *Theor. Comput. Sci.*, 113(1):75–91, 1993.
[332] M. Hutter. New error bounds for Solomonoff prediction. *J. Comput. Syst. Sci.*, 62(4):653–667, 2001.

[333] M. Hutter. The fastest and shortest algorithm for all well-defined problems. *Int. J. Found. Comput. Sci.*, 13(3):431–443, 2002.

[334] M. Hutter. Convergence and loss bounds for Bayesian sequence prediction. *IEEE Trans. Inform. Theory*, 49(8):2061–2067, 2003.

[335] M. Hutter. On the existence and convergence of computable universal priors. In *Proc. 14th Conf. Algorithmic Learn. Theory*, volume 2842 of *Lect. Notes Comput. Sci.*, pages 298–312, Berlin, 2003. Springer-Verlag.

[336] M. Hutter. Optimality of universal Bayesian sequence prediction for general loss and alphabet. *J. Mach. Learn. Res.*, 4:971–1000, 2003.

[337] M. Hutter. *Universal Artificial Intelligence: Sequential Decisions Based on Algorithmic Probability*. Springer-Verlag, Berlin, 2005.

[338] M. Hutter. Sequential predictions based on algorithmic complexity. *J. Comput. Syst. Sci.*, 72:95–117, 2006.

[339] M. Hutter. On universal prediction and Bayesian confirmation. *Theor. Comput. Sci.*, 384(1):33–48, 2007.

[340] M. Hutter and An.A. Muchnik. On semimeasures predicting Martin-Löf random sequences. *Theor. Comput. Sci.*, 382:247–261, 2007.

[341] D.T. Huynh. Resource-bounded Kolmogorov complexity of hard languages. In *Proc. 1st IEEE Conf. Struct. Complexity Theory*, volume 223 of *Lect. Notes Comput. Sci.*, pages 184–195, Berlin, 1986. Springer-Verlag.

[342] D.T. Huynh. Effective entropies and data compression. *Inform. Comput.*, 90(1):67–85, 1991.

[343] D.T. Huynh. The effective entropies of some extensions of context-free languages. *Inform. Process. Lett.*, 37:165–169, 1991.

[344] D.T. Huynh. Non-uniform complexity and the randomness of certain complete languages. *Theor. Comput. Sci.*, 96:305–324, 1992.

[345] L. Ilie, S. Yu, and K. Zhang. Word complexity and repetition in words. *Int. J. Found. Comput. Sci.*, 15(1):41–55, 2004.

[346] R. Impagliazzo, R. Shaltiel, and A. Wigderson. Extractors and pseudorandom generators with optimal seed-length. In *Proc. 32th ACM Symp. Theory Comput.*, pages 1–10, 2000.

[347] K. Jacobs. Turingmaschinen und zufällige 0-1-Folgen. *Selecta Mathematica*, 2:141–167, 1970.

[348] A.K. Jagota and K.W. Regan. Performance of neural net heuristics for maximum clique on diverse highly compressible graphs. *J. Global Optimization*, 10(4):439–465, 1997.

[349] A. Jakoby, R. Reischuk, and C. Schindelhauer. Malign distributions for average case circuit complexity. *Inform. Comput.*, 150:187–208, 1999.

[350] E.T. Jaynes. Prior probabilities. *IEEE Trans. Systems Man Cybernet.*, 4:227–241, 1968.

[351] E.T. Jaynes. On the rationale of maximum entropy methods. *Proceedings of the IEEE*, 70:939–952, 1982.

[352] E.T. Jaynes. *Papers on Probability, Statistics, and Statistical Physics*. Kluwer Academic Publishers, 1989. Second edition.

[353] T. Jiang and M. Li. On the approximation of shortest common supersequences and longest common subsequences. *SIAM J. Comput.*, 24(5):1122–1139, 1995.

[354] T. Jiang and M. Li. k one-way heads cannot do string-matching. *J. Comput. Syst. Sci.*, 53(3):513–524, 1996.

[355] T. Jiang, M. Li, and P.M.B. Vitányi. New applications of the incompressibility method. *Comput. J.*, 42(4):287–293, 1999.

[356] T. Jiang, M. Li, and P.M.B. Vitányi. Average-case analysis of algorithms using Kolmogorov complexity. *J. Comput. Sci. Tech.*, 15(5):402–408, 2000.

[357] T. Jiang, M. Li, and P.M.B. Vitányi. A lower bound on average-case complexity of Shellsort. *J. Assoc. Comp. Mach.*, 47(5):905–911, 2000.

[358] T. Jiang, M. Li, and P.M.B. Vitányi. The average-case area of Heilbronn-type triangles. *Rand. Struct. Alg.*, 20(2):206–219, 2002.

[359] T. Jiang, J.I. Seiferas, and P.M.B. Vitányi. Two heads are better than two tapes. *J. Assoc. Comput. Mach.*, 44(2):237–256, 1997.

[360] D. Joseph and M. Sitharam. Kolmogorov complexity, restricted nondeterminism and generalized spectra. In C. Choffrut and T. Lengauer, editors, *Proc. 7th Symp. Theor. Aspects Comput. Sci.*, pages 152–164, Berlin, 1990. Springer-Verlag.

[361] D.J. Juedes, J.I. Lathrop, and J.H. Lutz. Computational depth and reducibility. *Theor. Comput. Sci.*, 132(1-2):37–70, 1994.

[362] D.W. Juedes and J.H. Lutz. Kolmogorov complexity, complexity cores, and the distribution of hardness. In O. Watanabe, editor, *Kolmogorov Complexity and Computational Complexity*, pages 43–65. Springer-Verlag, Berlin, 1992.

[363] D.W. Juedes and J.H. Lutz. Modeling time-bounded prefix Kolmogorov complexity. *Theory Comput. Systems*, 33:111–123, 2000.

[364] T. Jurdziński and K. Loryś. Lower bound technique for length-reducing automata. *Inf. Comput.*, 205(9):1387–1412, 2007.

[365] Y. Kalnishkan. General linear relations between different types of predictive complexity. *Theor. Comput. Sci.*, 271(1-2):181–200, 2002.

[366] A. Kaltchenko. Algorithms for estimation of information distance with applications in bioinformatics and linguistics. In *Canadian Conf. Electrical and Comput. Engin., volume 4*, pages 2255–2258, 2004.

[367] B. Kalyanasundaram and G. Schnitger. The probabilistic communication complexity of set intersection. *SIAM J. Discrete Math.*, 5(4):545–557, 1992.

[368] T. Kamae. On Kolmogorov's complexity and information. *Osaka J. Math.*, 10:305–307, 1973.

[369] T. Kamae. Subsequences of normal sequences. *Israel J. Math.*, 16:121–149, 1973.

[370] T. Kamae and B. Weiss. Normal numbers and selection rules. *Israel J. Math.*, 21:101–110, 1975.

[371] M.I. Kanovich. On the decision complexity of algorithms. *Soviet Math. Dokl.*, 10:700–701, 1969.

[372] M.I. Kanovich. Complexity of resolution of a recursively enumerable set as a criterion of its universality. *Soviet Math. Dokl.*, 11:1224–1228, 1970.

[373] M.I. Kanovich. On the complexity of enumeration and decision of predicates. *Soviet Math. Dokl.*, 11:17–20, 1970.

[374] M.I. Kanovich. On the decision complexity of recursively enumerable sets. *Soviet Math. Dokl.*, 11:704–706, 1970.

[375] M.I. Kanovich. On the complexity of Boolean function minimization. *Soviet Math. Dokl.*, 12(3):720–724, 1971.

[376] M.I. Kanovich. On the precision of a complexity criterion for nonrecursiveness and universality. *Soviet Math. Dokl.*, 18:232–236, 1977.

[377] M.I. Kanovich. An estimate of the complexity of arithmetic incompleteness. *Soviet Math. Dokl.*, 19:206–210, 1978.

[378] M.I. Kanovich and N.V. Petri. Some theorems on the complexity of normal algorithms and their computations. *Soviet Math. Dokl.*, 10(1):233–234, 1969.

[379] F. Kaspar and H.G. Schuster. Easily calculable measures for the complexity of spatiotemporal patterns. *Phys. Review A*, 36(2):842–848, July 1987.

[380] H.P. Katseff. Complexity dips in infinite binary sequences. *Inform. Contr.*, 38:258–263, 1978.

[381] H.P. Katseff and M. Sipser. Several results in program size complexity. *Theor. Comput. Sci.*, 15:291–309, 1981.

[382] J. Kececioglu, M. Li, and J.T. Tromp. Reconstructing a DNA sequence from erroneous copies. *Theor. Comput. Sci.*, 185(1):3–13, 1997.

[383] J.G. Kemeny. The use of simplicity in induction. *Philos. Rev.*, 62:391–408, 1953.

[384] E. Keogh, S. Lonardi, and C.A. Ratanamahatana. Towards parameter-free data mining. In *Proc. 10th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, pages 206–215, 2004.

[385] E. Keogh, S. Lonardi, C.A. Ratanamahatana, L. Wei, H.S. Lee, and J. Handley. Compression-based data mining of sequential data. *Data Mining and Knowledge Discovery*, 14:99–129, 2007.

[386] H.A. Keuzenkamp and M. McAleer. Simplicity, scientific inference and econometric modelling. *The Economic Journal*, 105:1–21, 1995.

[387] A.I. Khinchin. *Mathematical Foundations of Information Theory*. Dover, 1957.

[388] J.C. Kieffer and E.H. Yang. Sequential codes, lossless compression of individual sequences, and Kolmogorov complexity. *IEEE Trans. Inform. Theory*, 42(1):29–39, 1996.

[389] J.C. Kieffer and E.H. Yang. Grammar-based codes: a new class of universal lossless source codes. *IEEE Trans. Inform. Theory*, 46(3):737–754, 2000.

[390] M. Kikuchi. Kolmogorov complexity and the second incompleteness theorem. *Arch. Math. Logic*, 36(1):437–443, 1997.

[391] W.W. Kirchherr. Kolmogorov complexity and random graphs. *Inform. Process. Lett.*, 41:125–130, 1992.

[392] W.W. Kirchherr, M. Li, and P.M.B. Vitányi. The miraculous universal distribution. *Math. Intelligencer*, 19(4):7–15, 1997.

[393] S.R. Kirk and S. Jenkins. Information theory-based software metrics and obfuscation. *Journal of Systems and Software*, 72:179–186, 2004.

[394] D.E. Knuth. *The Art of Computer Programming. Volume 1: Fundamental Algorithms*. Addison-Wesley, 1973. Second edition.

[395] D.E. Knuth. *The Art of Computer Programming. Volume 2: Seminumerical Algorithms*, pages 163–166. Addison-Wesley, 1981. Second edition.

[396] Ker-I Ko. On the definition of infinite pseudo-random sequences. *Theor. Comput. Sci.*, 48:9–34, 1986.
[397] Ker-I Ko. On the complexity of learning minimum time-bounded Turing machines. *SIAM J. Comput.*, 20:962–986, 1991.
[398] K. Kobayashi. On the structure of one-tape nondeterministic Turing machine time hierarchy. *Theor. Comput. Sci.*, 40:175–193, 1985.
[399] K. Kobayashi. Σ_n^0-complete properties of programs and Martin-Löf randomness. *Inform. Process. Lett.*, 46:37–42, 1993.
[400] K. Kobayashi. On malign input distributions for algorithms. *IEICE Trans. Inform. and Syst.*, E76-D(6):634–640, 1993.
[401] K. Kobayashi. The Kolmogorov complexity, the universal distribution and the coding theorem with generalized length functions. *IEEE Trans. Inform. Theory*, 43(3):816–826, 1997.
[402] A. Kocsor, A. Kertész-Farkas, L. Kaján, and S. Pongor. Application of compression-based distance measures to protein sequence classification: a methodology study. *Bioinformatics*, 22(4):407–412, 2006.
[403] A.N. Kolmogorov. *Grundbegriffe der Wahrscheinlichkeitsrechnung*. Springer-Verlag, Berlin, 1933. English translation (by N. Morrison): *Foundations of the Theory of Probability*, Chelsea, 1956; 2nd Russian edition: *Osnovnye Poniatiia Teorii Veroiatnostei*, Nauka, 1974.
[404] A.N. Kolmogorov. On tables of random numbers. *Sankhyā, The Indian Journal of Statistics, Ser. A*, 25:369–376, 1963.
[405] A.N. Kolmogorov. Three approaches to the quantitative definition of information. *Problems Inform. Transmission*, 1(1):1–7, 1965.
[406] A.N. Kolmogorov. Logical basis for information theory and probability theory. *IEEE Trans. Inform. Theory*, 14(5):662–664, 1968.
[407] A.N. Kolmogorov. Some theorems on algorithmic entropy and the algorithmic quantity of information. *Uspekhi Mat. Nauk*, 23(2):201, 1968. Meeting of the Moscow Mathematical Society.
[408] A.N. Kolmogorov. On the logical foundations of information theory and probability theory. *Problems Inform. Transmission*, 5:1–4, 1969.
[409] A.N. Kolmogorov. Complexity of algorithms and objective definition of randomness. *Uspekhi Mat. Nauk*, 29(4):155, 1974. Abstract of a talk at the Moscow Math. Soc. meeting 4/16/1974. In Russian.
[410] A.N. Kolmogorov. Combinatorial foundations of information theory and the calculus of probabilities. *Russian Math. Surveys*, 38(4):29–40, 1983.
[411] A.N. Kolmogorov. On logical foundations of probability theory. In K. Itô and Yu.V. Prokhorov, editors, *Probability Theory and Mathematical Statistics*, volume 1021 of *Lect. Notes Math.*, pages 1–5. Springer-Verlag, Berlin, 1983.
[412] A.N. Kolmogorov. Memories of P.S. Aleksandrov. *Russian Math. Surveys*, 41(6):225–246, 1986.
[413] A.N. Kolmogorov. *Information Theory and Theory of Algorithms, Selected Works*, volume 3. Nauka, 1987. Edited by Yu.V. Prokhorov and A.N. Shiryaev. In Russian.
[414] A.N. Kolmogorov. Letters of A.N. Kolmogorov to A. Heyting. *Russian Math. Surveys*, 43(6):89–93, 1988.

[415] A.N. Kolmogorov and V.A. Uspensky. On the definition of an algorithm. *Uspekhi Mat. Nauk*, 13(4):3–28, 1958. In Russian. English translation: *Amer. Math. Soc. Translat.*, 29:2(1963), 217–245.

[416] A.N. Kolmogorov and V.A. Uspensky. Algorithms and randomness. *SIAM Theory Probab. Appl.*, 32:389–412, 1987. Without annoying translation errors pp. 3–53 in: Yu.V. Prokhorov and V.V. Sazonov, editors, *Proc. 1st World Congress of the Bernoulli Society* (*Tashkent 1986*), *Vol. 1: Probab. Theory and Appl.*, VNU Science Press, Utrecht, 1987.

[417] D.K. Kondepudi. Non-equilibrium polymers, entropy, and algorithmic information. In W.H. Zurek, editor, *Complexity, Entropy and the Physics of Information*, pages 199–206. Addison-Wesley, New York, 1991.

[418] M. Koppel. Complexity, depth, and sophistication. *Complex Systems*, 1:1087–1091, 1987.

[419] M. Koppel. Structure. In R. Herken, editor, *The Universal Turing Machine: A Half-Century Survey*, pages 435–452. Oxford University Press, 1988. In Germany: Kammerer & Unverzagt, Hamburg.

[420] M. Koppel and H. Atlan. Program-length complexity, sophistication and induction. Memo, 1988.

[421] M. Koucký. *On traversal sequences, exploration sequences and completeness of Kolmogorov random strings*. PhD thesis, Rutgers, the State Univ., New Jersey, 2003.

[422] L.G. Kraft. A device for quantizing, grouping and coding amplitude modulated pulses. Master's thesis, Dept. of Electrical Engineering, M.I.T., Cambridge, Mass., 1949.

[423] E. Kranakis and D. Krizanc. Lower bounds for compact routing. In *Proc. 13th Symp. Theor. Aspects Comput. Sci.*, volume 1046 of *Lect. Notes Comput. Sci.*, pages 529–540, Berlin, 1996. Springer-Verlag.

[424] E. Kranakis, D. Krizanc, and F. Luccio. On recognizing a string on an anonymous ring. *Theory Comput. Syst.*, 34(1):3–12, 2001.

[425] A. Kraskov, H. Stögbauer, R.G. Andrzejak, and P. Grassberger. Hierarchical clustering using mutual information. *Europhys. Lett.*, 70(2):278–284, 2005.

[426] N. Krasnogor and D.A. Pelta. Measuring the similarity of protein structures by means of the universal similarity metric. *Bioinformatics*, 20(7):1015–1021, 2004.

[427] V. Kreinovich and L. Longpré. Unreasonable effectiveness of symmetry in physics. *Int. J. Theor. Phys.*, 36:1549–1555, 1996.

[428] V. Kreinovich and L. Longpré. Why Kolmogorov complexity in physical equations? *Int. J. Theor. Phys.*, 37(11):2791–2801, 1998.

[429] V. Kreinovich and R. Watson. How difficult is it to invent a nontrivial game? *Cybernetics and Systems*, 25:629–640, 1994.

[430] R.E. Krichevskii. Universal encoding and Kolmogorov complexity. In *Proc. 5th Int. Symp. Inform. Theory, Part 1, Abstracts of Papers*, pages 22–25, Moscow-Tbilisi, 1979. In Russian.

[431] R.E. Krichevskii and V.K. Trofimov. The performance of universal encoding. *IEEE Trans. Inform. Theory*, 27:199–207, 1983.

[432] M. Kummer. Kolmogorov complexity and instance complexity of recursively enumerable sets. *SIAM J. Comput.*, 25(6):1123–1143, 1996.

[433] M. Kummer. On the complexity of random strings. In *Proc. 13th Symp. Theor. Aspects Comput. Sci.*, volume 1046 of *Lect. Notes Comput. Sci.*, pages 25–36, Berlin, 1996. Springer-Verlag.

[434] A. Kučera and T.A. Slaman. Randomness and recursive enumerability. *SIAM J. Comput.*, 31(1):199–211, 2002.

[435] K. Lanctot, M. Li, and E.H. Yang. Estimating DNA sequence entropy. In *Proc. 11th ACM-SIAM Symp. Discr. Algorithms*, pages 409–418, 2000.

[436] R. Landauer. Irreversibility and heat generation in the computing process. *IBM J. Res. Develop.*, 5:183–191, 1961.

[437] P.S. Laplace. *A Philosophical Essay on Probabilities*. Dover, 1952. Originally published in 1819. Translated from 6th French edition.

[438] S. Laplante and F. Magniez. Lower bounds for randomized and quantum query complexity using Kolmogorov arguments. In *Proc. 19th IEEE Conf. Comput. Complexity*, pages 294–304, 2004.

[439] J.I. Lathrop. Compression depth and the behavior of cellular automata. Technical Report TR96-05, Comput. Sci. Dept., Iowa State Univ., 1996.

[440] J.I. Lathrop and J.H. Lutz. Recursive computational depth. *Inform. Comput.*, 153(2):139–172, 1999.

[441] T. Lee and A. Romashchenko. Resource bounded Kolmogorov complexity revisited. *Theor. Comput. Sci.*, 245:386–405, 2005.

[442] A. Lempel and J. Ziv. On the complexity of finite sequences. *IEEE Trans. Inform. Theory*, 22:75–81, 1976.

[443] S.K. Leung-Yan-Cheong and T.M. Cover. Some equivalences between Shannon entropy and Kolmogorov complexity. *IEEE Trans. Inform. Theory*, 24:331–338, 1978.

[444] L.A. Levin. *Some theorems on the algorithmic approach to probability theory and information theory*. PhD thesis, Moscow University, 1971. In Russian.

[445] L.A. Levin. On storage capacity for algorithms. *Soviet Math. Dokl.*, 14:1464–1466, 1973.

[446] L.A. Levin. On the notion of a random sequence. *Soviet Math. Dokl.*, 14:1413–1416, 1973.

[447] L.A. Levin. Universal search problems. *Problems Inform. Transmission*, 9:265–266, 1973.

[448] L.A. Levin. Laws of information conservation (non-growth) and aspects of the foundation of probability theory. *Problems Inform. Transmission*, 10:206–210, 1974.

[449] L.A. Levin. On the principle of conservation of information in intuitionistic mathematics. *Soviet Math. Dokl.*, 17:601–605, 1976.

[450] L.A. Levin. Uniform tests of randomness. *Soviet Math. Dokl.*, 17:337, 1976.

[451] L.A. Levin. Various measures of complexity for finite objects (axiomatic description). *Soviet Math. Dokl.*, 17:522–526, 1976.

[452] L.A. Levin. Randomness conservation inequalities; information and independence in mathematical theories. *Inform. Contr.*, 61:15–37, 1984.

[453] L.A. Levin. One-way functions and pseudorandom generators. *Combinatorica*, 7:357–363, 1987.

[454] L.A. Levin. Robust measures of information. *Comput. J.*, 42(4):284–286, 1999.

[455] L.A. Levin. Forbidden information. In *Proc. 47th IEEE Symp. Found. Comput. Sci.*, pages 761–768, 2002.

[456] L.A. Levin and V.V. Vyugin. Invariant properties of information bulks. In *Proc. 6th Symp. Math. Found. Comput. Sci.*, volume 53 of *Lect. Notes Comput. Sci.*, pages 359–364, Berlin, 1977. Springer-Verlag.

[457] J.P. Lewis. Large limits to software estimation. *ACM Software Eng. Notes*, 26(4):54–59, 2001.

[458] M. Li. Information distance and its applications. *Int. J. Found. Comput. Sci.*, 18(4):669–681, 2007.

[459] M. Li, J. Badger, X. Chen, S. Kwong, P. Kearney, and H. Zhang. An information-based sequence distance and its application to whole mitochondrial genome phylogeny. *Bioinformatics*, 17(2):149–154, 2001.

[460] M. Li, X. Chen, X. Li, B. Ma, and P.M.B. Vitányi. The similarity metric. *IEEE Trans. Inform. Theory*, 50(12):3250–3264, 2004.

[461] M. Li, L. Longpré, and P.M.B. Vitányi. The power of the queue. *SIAM. J. Comput.*, 21(4):697–712, 1992.

[462] M. Li, B. Ma, and L. Wang. Finding similar regions in many strings. In *Proc. 31st ACM Symp. Theory Comput.*, pages 473–482, 1999.

[463] M. Li, J.T. Tromp, and P.M.B. Vitányi. Sharpening Occam's razor. *Inform. Process. Lett.*, 85(5):267–274, 2003.

[464] M. Li and P.M.B. Vitányi. Kolmogorovskaya slozhnost' dvadsat' let spustia. *Uspekhi Mat. Nauk*, 43(6):129–166, 1988. In Russian.

[465] M. Li and P.M.B. Vitányi. Tape versus queue and stacks: The lower bounds. *Inform. Comput.*, 78:56–85, 1988.

[466] M. Li and P.M.B. Vitányi. Two decades of applied Kolmogorov complexity: In memoriam A.N. Kolmogorov 1903–1987. In *Proc. 3rd IEEE Conf. Structure in Complexity Theory*, pages 80–101, 1988.

[467] M. Li and P.M.B. Vitányi. Applications of Kolmogorov complexity in the theory of computation. In A.L. Selman, editor, *Complexity Theory Retrospective*, pages 147–203. Springer-Verlag, New York, 1990.

[468] M. Li and P.M.B. Vitányi. Kolmogorov complexity and its applications. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, chapter 4, pages 187–254. Elsevier and MIT Press, 1990.

[469] M. Li and P.M.B. Vitányi. Learning simple concepts under simple distributions. *SIAM J. Comput.*, 20(5):911–935, 1991.

[470] M. Li and P.M.B. Vitányi. Inductive reasoning and Kolmogorov complexity. *J. Comput. System Sci.*, 44(2):343–384, 1992.

[471] M. Li and P.M.B. Vitányi. Worst case complexity is equal to average case complexity under the universal distribution. *Inform. Process. Lett.*, 42:145–149, 1992.

[472] M. Li and P.M.B. Vitányi. Mathematical theory of thermodynamics of computation. In *IEEE Proc. Workshop Phys. Comput.*, pages 42–46, 1993. Complete version in prelim. proceedings, Oct., 1992.

[473] M. Li and P.M.B. Vitányi. Kolmogorov complexity arguments in combinatorics. *J. Comb. Theory, Ser. A*, 66(2):226–236, 1994. Erratum, *Ibid.*, 69(1995), 183.

[474] M. Li and P.M.B. Vitányi. Statistical properties of finite sequences with high Kolmogorov complexity. *Math. Systems Theory*, 27:365–376, 1994.

[475] M. Li and P.M.B. Vitányi. Computational machine learning in theory and praxis. In J. van Leeuwen, editor, *Computer Science Today, Recent Trends and Developments*, volume 1000 of *Lect. Notes Comput. Sci.*, pages 518–535. Springer-Verlag, Berlin, 1995.

[476] M. Li and P.M.B. Vitányi. A new approach to formal language theory by Kolmogorov complexity. *SIAM J. Comput.*, 24(2):398–410, 1995.

[477] M. Li and P.M.B. Vitányi. Reversibility and adiabatic computation: trading time and space for energy. *Proc. Royal Soc. London, Ser. A*, 452:769–789, 1996.

[478] M. Li and Y. Yesha. String-matching cannot be done by 2-head 1-way deterministic finite automata. *Inform. Process. Lett.*, 22:231–235, 1986.

[479] M. Li and Y. Yesha. New lower bounds for parallel computation. *J. Assoc. Comput. Mach.*, 36:671–680, 1989.

[480] W. Li. On the relationship between complexity and entropy for Markov chains and regular languages. *Complex Systems*, 5(4):381–399, 1991.

[481] T.Y. Lin. Patterns in numerical data: practical approximations to Kolmogorov complexity. In *Proc. 7th Int. Workshop New Directions in Rough Sets, Data Mining, Granular-Soft Comput.*, volume 1711 of *Lect. Notes Artif. Intell.*, pages 509–513, Berlin, 1999. Springer-Verlag.

[482] S. Lloyd and J.J. Slotine. Algorithmic Lyapunov functions for stable adaptation and control. *Int. J. Adapt. Contr. Sign. Proc.*, 1996.

[483] L. Löfgren. Explicability of sets and transfinite automata. In E. Caianiello, editor, *Automata Theory*, pages 251–268. Academic Press, 1966.

[484] L. Löfgren. Recognition of order and evolutionary systems. In J. Tou, editor, *Computer and Information Sciences II*, pages 165–175. Academic Press, 1967.

[485] L. Löfgren. Complexity of descriptions of systems. *Int. J. General Systems*, 3:197–214, 1977.

[486] L. Longpré. *Resource bounded Kolmogorov complexity, a link between computational complexity and information theory.* PhD thesis, Comput. Sci. Dept., Cornell Univ., 1986.

[487] L. Longpré. Resource bounded Kolmogorov complexity and statistical tests. In O. Watanabe, editor, *Kolmogorov Complexity and Computational Complexity*, pages 66–84. Springer-Verlag, Berlin, 1992.

[488] L. Longpré and S. Mocas. Symmetry of information and one-way functions. *Inform. Process. Lett.*, 46(2):95–100, 1993.

[489] L. Longpré and O. Watanabe. On symmetry of information and polynomial time invertibility. *Inform. Comput.*, 121(1):14–22, 1995.

[490] A. López-Ortiz. New lower bounds for element distinctness on a one-tape Turing machine. *Inform. Proc. Lett.*, 51(6):311–314, 1994.

[491] M.C. Loui. Optimal dynamic embedding of trees into arrays. *SIAM J. Comput.*, 12:463–472, 1983.

[492] M.C. Loui. Minimizing access pointers into trees and arrays. *J. Comput. System Sci.*, 28:359–378, 1984.

[493] M.C. Loui and D.R. Luginbuhl. The complexity of simulations between multidimensional Turing machines and random access machines. *Math. Systems Theory*, 25(4):293–308, 1992.

[494] M.C. Loui and D.R. Luginbuhl. Optimal on-line simulations of tree machines by random access machines. *SIAM J. Comput.*, 21(5):959–971, 1992.

[495] D.W. Loveland. The Kleene hierarchy classification of recursively random sequences. *Trans. Amer. Math. Soc.*, 125:497–510, 1966.

[496] D.W. Loveland. A new interpretation of von Mises' concept of a random sequence. *Z. Math. Logik und Grundlagen Math.*, 12:279–294, 1966.

[497] D.W. Loveland. On minimal-program complexity measures. In *Proc. (1st) ACM Symp. Theory Comput.*, pages 61–66, 1969.

[498] D.W. Loveland. A variant of the Kolmogorov concept of complexity. *Inform. Contr.*, 15:510–526, 1969.

[499] B. Lucier, T. Jiang, and M. Li. Average-case analysis of Quicksort and Binary Insertion Tree height using incompressibility. *Inform. Process. Lett.*, 103:45–51, 2007.

[500] D.R. Luginbuhl. *Computational complexity of random access models.* PhD thesis, University of Illinois at Urbana-Champaign, 1990.

[501] J.H. Lutz. Category and measure in complexity classes. *SIAM. J. Comput.*, 19:6:1100–1131, 1990.

[502] J.H. Lutz. An upward measure separation theorem. *Theor. Comput. Sci.*, 81:127–135, 1991.

[503] J.H. Lutz. Almost everywhere high nonuniform complexity. *J. Comput. System Sci.*, 44:220–258, 1992.

[504] J.H. Lutz. The quantitative structure of exponential time. In *Proc. 8th IEEE Conf. Structure in Complexity Theory*, pages 158–175, 1993.

[505] J.H. Lutz. Gales and the constructive dimension of individual sequences. In *27th Int. Colloq. Automata, Languages, Programming*, volume 1853 of *Lect. Notes Comput. Sci.*, pages 185–194, Berlin, 2000. Springer-Verlag.

[506] J.H. Lutz. The dimensions of individual strings and sequences. *Inform. Comput.*, 187(1):49–79, 2003.

[507] J.H. Lutz. Effective fractal dimensions. *Math. Logic Quarterly*, 51(1):62–72, 2004.

[508] J.H. Lutz and D.L. Schweizer. Feasible reductions to Kolmogorov-Loveland stochastic sequences. *Theor. Comput. Sci.*, 225:185–194, 1999.

[509] W. Maass. Combinatorial lower bound arguments for deterministic and nondeterministic Turing machines. *Trans. Amer. Math. Soc.*, 292:675–693, 1985.

[510] W. Maass, E. Szemerédi, G. Schnitger, and G. Turan. Two tapes versus one for off-line Turing machines. *Comput. Complexity*, 3:392–401, 1993.

[511] J. Machta. Entropy, information, and computation. *Am. J. Phys.*, 67:1074–1077, 1999.

[512] H.G. Mairson. The program complexity of searching a table. In *Proc. 24th IEEE Symp. Found. Comput. Sci.*, pages 40–47, 1983.

[513] K. Makarychev, Y. Makarychev, A. Romashchenko, and N.K. Vereshchagin. A new class of non-Shannon-type inequalities for entropies. *Comm. in Inform. Syst.*, 2(2):147–166, 2002.

[514] R.N. Mantegna and H.E. Stanley. *An introduction to econophysics: correlations and complexity in finance.* Cambridge University Press, 2000.

[515] G.B. Marandzhyan. On certain properties of asymptotically optimal recursive function. *Izv. Akad. Nauk Armyan. SSSR*, 4:3–22, 1969.

[516] A.A. Markov. On normal algorithms which compute Boolean functions. *Soviet Math. Dokl.*, 5:922–924, 1964.

[517] A.A. Markov. On normal algorithms associated with the computation of Boolean functions and predicates. *Izv. Akad. Nauk USSR Ser. Mat.*, 31:161–208, 1967.

[518] P. Martin-Löf. Algorithmen und zufällige Folgen. Lecture notes, University of Erlangen, 1966.

[519] P. Martin-Löf. The definition of random sequences. *Inform. Contr.*, 9:602–619, 1966.

[520] P. Martin-Löf. On the concept of a random sequence. *Theory Probability Appl.*, 11:177–179, 1966.

[521] P. Martin-Löf. Algorithms and randomness. *Rev. Int. Statist. Inst.*, 37:265–272, 1969.

[522] P. Martin-Löf. The literature on von Mises' Kollektivs revisited. *Theoria*, 35(1):12, 1969.

[523] P. Martin-Löf. *Notes on Constructive Mathematics*. Almqvist and Wiksell, 1970.

[524] P. Martin-Löf. On the notion of randomness. In A. Kino et al., editor, *Intuitionism and Proof Theory*, pages 73–78. North-Holland, 1970.

[525] P. Martin-Löf. Complexity oscillations in infinite binary sequences. *Z. Wahrscheinlichkeitstheorie verw. Gebiete*, 19:225–230, 1971.

[526] P. Martin-Löf. The notion of redundancy and its use as a quantitative measure of the discrepancy between a statistical hypothesis and a set of observational data. *Scand. J. Stat.*, 1:3–18, 1974.

[527] P. Martin-Löf. Reply to Sverdrup's polemical article "Tests without power". *Scand. J. Stat.*, 2:161–165, 1975.

[528] E. Mayordomo. A Kolmogorov complexity characterization of constructive Hausdorff dimension. *Inform. Process. Lett.*, 84(1):247–356, 2002.

[529] J.W. McAllister. Effective complexity as a measure of information content. *Philosophy of Science*, 70:302–307, 2003.

[530] K. Mehlhorn. On the program-size of perfect and universal hash functions. In *Proc. 23rd IEEE Symp. Found. Comput. Sci.*, pages 170–175, 1982.

[531] N. Merhav and M. Feder. Universal prediction of individual sequences. *IEEE Trans. Inform. Theory*, 38(4):1258–1270, 1992.

[532] W. Merkle. The Kolmogorov-Loveland stochastic sequences are not closed under selecting subsequences. *J. Symb. Logic*, 68:1362–1376, 2003.

[533] W. Merkle. The complexity of stochastic sequences. *J. Comput. Syst. Sci.*, 74(3):350–357, 2008.

[534] W. Merkle and N. Mihailovic. On the construction of effectively random sets. *J. Symb. Logic*, 69(3):862–878, 2004.

[535] W. Merkle, N. Mihailovic, and T.A. Slaman. Some results on effective randomness. *Theory Comput. Syst.*, 39:707–722, 2006.

[536] W. Merkle, J.S. Miller, A. Nies, F. Stephan, and J. Reimann. Kolmorogov-Loveland randomness and stochasticity. *Ann. Pure Appl. Logic*, 138(1-3):303–326, 2006.

[537] W. Merkle and J. Reimann. Selection functions that do not preserve normality. *Theory Comput. Syst.*, 39:685–697, 2006.

[538] N.C. Metropolis, G. Reitweiser, and J. von Neumann. Statistical treatment of values of the first 2,000 decimal digits of e and π calculated on the ENIAC. In A.H. Traub, editor, *John von Neumann, Collected Works, Vol. V.* Macmillan, 1963.

[539] J.S. Miller. Every 2-random real is Kolmorogov random. *J. Symbolic Logic*, 69(3):907–913, 2004.

[540] J.S. Miller. Contrasting plain and prefix-free Kolmogorov complexity. *SIAM. J. Comput.*, 2006. Submitted.

[541] J.S. Miller. The k degrees, low for k degrees, and weakly low for k sets. Manuscript, 2007.

[542] J.S. Miller and L. Yu. Oscillations in the initial segment complexity of random reals. Manuscript, 2007.

[543] J.S. Miller and L. Yu. On initial segment complexity and degrees of randomness. *Trans. American Math. Soc.*, 360(6):3193–3210, 2008.

[544] A. Milosavljević and J. Jurka. Discovering simple DNA sequences by the algorithmic significance method. *CABIOS*, 9(4):407–411, 1993.

[545] A. Milosavljević and J. Jurka. Discovery by minimal length encoding: a case study in molecular evolution. *Machine Learning*, 12:69–87, 1993.

[546] P.B. Miltersen. The complexity of malign ensembles. *SIAM J. Comput.*, 22(1):147–156, 1993.

[547] M.L. Minsky. Problems of formulation for artificial intelligence. In R.E. Bellman, editor, *Mathematical Problems in the Biological Sciences*, Proc. Symposia in Applied Mathematics XIV, page 43. American Mathematical Society, 1962.

[548] M.L. Minsky. Steps towards artificial intelligence. *Proceedings I.R.E.*, pages 8–30, January, 1961.

[549] J.L. Montana and L.M. Pardo. On Kolmogorov complexity in the real Turing machine setting. *Inform. Process. Lett.*, 67(2):81–86, 1998.

[550] C.E. Mora and H.J. Briegel. Algorithm complexity and entanglement of quantum states. *Phys. Rev. Lett.*, 95:200503, 2005.

[551] C.E. Mora and H.J. Briegel. Algorithm complexity of quantum states. *Int. J. Quantum Information*, 4:715, 2006.

[552] C.E. Mora, H.J. Briegel, and B. Kraus. Quantum Kolmogorov complexity and its applications. arXiv.org:quant-ph/0610109, 2006.

[553] P. Moulin and J. Liu. Analysis of multiresolution image denoising schemes using generalized Gaussian and complexity priors. *IEEE Trans. Inform. Theory*, 45(3):909–919, 1999.

[554] An.A. Muchnik. Lower limits on frequencies in computable sequences and relativized a priori probability. *SIAM Theory Probab. Appl.*, 32:513–514, 1987.

[555] An.A. Muchnik. On common information. *Theor. Comput. Sci.*, 207:319–328, 1998.

[556] An.A. Muchnik. Conditional complexity and codes. *Theor. Comput. Sci.*, 271(1-2):97–109, 2002.

[557] An.A. Muchnik and S.Y. Positselsky. Kolmogorov entropy in the context of computability theory. *Theor. Comput. Sci.*, 271(1-2):15–35, 2002.

[558] An.A. Muchnik, A.L. Semenov, and V.A. Uspensky. Mathematical metaphysics of randomness. *Theor. Comput. Sci.*, 207:263–317, 1998.

[559] An.A. Muchnik, A.K. Shen, M. Ustinov, N.K. Vereshchagin, and M.V. Vyugin. Non-reducible descriptions for conditional Kolmogorov complexity. In *Proc. 3rd Conf. Theory Appl. Models Comput.*, volume 3959 of *Lect. Notes. Comput. Sci,*, pages 308–317, Berlin, 2006. Springer-Verlag.

[560] An.A. Muchnik and N.K. Vereshchagin. Logical operations and Kolmogorov complexity II. In *Proc. 16th IEEE Conf. Comput. Complexity*, pages 256–265, 2001.

[561] An.A. Muchnik and N.K. Vereshchagin. Shannon entropy versus Kolmogorov complexity. In *Proc. Int. Comput. Sci. Symp. Russia (CSR)*, volume 2925 of *Lect. Notes. Comput. Sci,*, pages 281–291, Berlin, 2006. Springer-Verlag.

[562] S. Muggleton, A. Srinivasan, and M. Bain. Compression, significance and accuracy. In *Proc. 9th Int. Workshop Machine Learning*, pages 338–347, Aberdeen, Scotland, UK, 1992.

[563] D.W. Müller. Randomness and extrapolation. In *Proc. 6th Berkeley Symposium*, pages 1–31, 1970.

[564] M. Mundhenk. On hard instances. *Theor. Comput. Sci.*, 242:301–311, 2000.

[565] J. Muramatsu and F. Kanaya. Distortion-complexity and rate-distortion function. *IEICE Trans. Fundamentals*, E77-A:8:1224–1229, 1994.

[566] I.J. Myung, V. Balasubramanian, and M.A. Pitt. Counting probability distributions: differential geometry and model selection. *Proc. National Acad. Sci.*, 97(21):11170–11175, 2000.

[567] A. Naik, K.W. Regan, and D. Sivakumar. On quasilinear time complexity theory. *Theor. Comput. Sci.*, 148(2):325–349, 1995.

[568] A. Nies. *Computability and Randomness*. Oxford University Press, To appear.

[569] S.M. Nikol'skii. Aleksandrov and Kolmogorov in Dnjepropetrovsk. *Russian Math. Surveys*, 38(4):41–55, 1983.

[570] S.P. Novikov. Memories of A.N. Kolmogorov. *Russian Math. Surveys*, 43(6):40–42, 1988.

[571] M. Nykter, N.D. Price, M. Aldana, S.A. Ramsey, S.A. Kauffman, L.E. Hood, O. Yli-Harja, and I. Shmulevich. Gene expression dynamics in the macrophage exhibit criticality. *Proc. Nat. Acad. Sci. USA*, 105(6):1897–1900, 2008.

[572] M. Nykter, N.D. Price, A. Larjo, T. Aho, S.A. Kauffman, O. Yli-Harja, and I. Shmulevich. Critical networks exhibit maximal information diversity in structure-dynamics relationships. *Phys. Rev. Lett.*, 100:058702(4), 2008.

[573] Obituary. Mr. Andrei Kolmogorov—Giant of mathematics. *Times*, October 26 1987.

[574] Obituary. Andrei Nikolaevich Kolmogorov. *Bull. London Math. Soc.*, 22(1):31–100, 1990.

[575] P. Odifreddi. *Classical Recursion Theory*. North-Holland, 1989.

[576] P. Orponen, Ker-I Ko, U. Schöning, and O. Watanabe. Instance complexity. *J. Assoc. Comput. Mach.*, 41:96–121, 1994.

[577] M. Paatrascu and E.D. Demaine. Tight bounds for the partial-sums problem. In *Proc. 15th ACM-SIAM Symp. Discrete Alg.*, pages 20–29, 2004.

[578] D. Pager. On the problem of finding minimal programs for tables. *Inform. Contr.*, 14:550–554, 1969.

[579] P. Pajunen. Blind source separation using algorithmc information theory. *Neurocomput.*, 22:35–48, 1998.

[580] R. Parekh and V. Honavar. Learning DFA from simple examples. *Machine Learning*, 44(1-2):9–35, 2001.

[581] E.M. Pathos and N. Chater. A simplicity principle in unsupervised human categorization. *Cognitive Science*, 26(3):303–343, 2002.

[582] R. Paturi. *Study of certain probabilistic models of information transfer and on-line computation.* PhD thesis, Penn State University, 1985.

[583] R. Paturi and J. Simon. Lower bounds on the time of probabilistic online simulations. In *Proc. 24th IEEE Symp. Found. Comput. Sci.*, pages 343–350, 1983.

[584] R. Paturi, J. Simon, R.E. Newman-Wolfe, and J. Seiferas. Milking the Aanderaa argument. *Inform. Comput.*, 88:88–104, 1990.

[585] W.J. Paul. Kolmogorov's complexity and lower bounds. In L. Budach, editor, *Proc. 2nd Int. Conf. Fund. Comput. Theory*, pages 325–334. Akademie Verlag, 1979.

[586] W.J. Paul. On-line simulation of $k+1$ tapes by k tapes requires nonlinear time. *Inform. Contr.*, 53:1–8, 1982.

[587] W.J. Paul. On heads versus tapes. *Theor. Comput. Sci.*, 28:1–12, 1984.

[588] W.J. Paul, J.I. Seiferas, and J. Simon. An information theoretic approach to time bounds for on-line computation. *J. Comput. Syst. Sci.*, 23(2):108–126, 1981.

[589] W.J. Paul and R.J. Solomonoff. Autonomous theory building systems. In P. Bock, M. Loew, and M. Richter, editors, *Neural Networks and Adaptive Learning*, Knowledge Processing and Its Applications Series. Elsevier Science Publishers, 1992.

[590] J. Pearl. On the connection between the complexity and credibility of inferred models. *Int. J. Gen. Syst.*, 4:255–264, 1978.

[591] H. Petersen. Bounds for the element distinctness problem on one-tape Turing machines. *Inform. Process. Lett.*, 81(2):75–79, 2002.

[592] H. Petersen and J.M. Robson. Efficient simulations by queue machines. In *Proc. 25th Int. Coll. Automata, Lang. Programming*, number 1443 in Lect. Notes Comput. Sci., pages 884–895, Berlin, 1998. Springer-Verlag.

[593] G. Peterson. Succinct representations, random strings and complexity classes. In *Proc. 21st IEEE Symp. Found. Comput. Sci.*, pages 86–95, 1980.

[594] N.V. Petri. Algorithms connected with predicates and Boolean functions. *Soviet Math. Dokl.*, 10:294–297, 1969.

[595] N.V. Petri. The complexity of algorithms and their operating time. *Soviet Math. Dokl.*, 10:547–549, 1969.

[596] J.D. Phillips, P.A. Gares, and M.C. Slattery. Agricaltural soil redistribution and landscape complexity. *Landscape Ecology*, 14(2):197–211, 1999.

[597] J.R. Pierce and C.C. Cutler. Interplanetary communications. In F.I. Ordway, III, editor, *Advances in Space Science, Vol. 1*, pages 55–109, New York, 1959. Academic Press, Inc.

[598] N. Pippenger. An information-theoretic method in combinatorial theory. *J. Comb. Theory, Ser. A*, 23:99–104, 1977.

[599] J. Poland and M. Hutter. Asymptotics of discrete MDL for on-line prediction. *IEEE Trans. Inform. Theory*, 51(11):3780–3795, 2005.
[600] K.R. Popper. *The Logic of Scientific Discovery*. University of Toronto Press, 1959.
[601] M.B. Pour-El and J.I. Richards. *Computability in Analysis and Physics*. Springer-Verlag, 1989.
[602] J. Quinlan and R. Rivest. Inferring decision trees using the minimum description length principle. *Inform. Comput.*, 80:227–248, 1989.
[603] J. Rabaey, L. Guerra, and R. Mehra. Design guidance in the power dimension. In *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, pages 2837–2840, 1995.
[604] R.P.N. Rao and D.H. Ballard. Dynamic model of visual recognition predicts neural response properties in the visual cortex. *Neural Comput.*, 9(4):721–763, 1997.
[605] P.E. Rapp, C.J. Cellucci, K.E. Korslund, T.A.A. Watanabe, and M.A. Jiménez-Montano. Effective normalization of complexity measurements for epoch length and sampling frequency. *Phys. Rev. E*, 64:016209, 2001.
[606] K.W. Regan. On superlinear lower bounds in complexity theory. In *Proc. 10th IEEE Conf. Structure in Complexity Theory*, pages 50–64, 1995.
[607] K.W. Regan and J. Wang. The quasilinear isomorphism challenge. *SIGACT News*, 25:106–113, September 1994.
[608] S. Reisch and G. Schnitger. Three applications of Kolmogorov complexity. In *Proc. 23rd IEEE Symp. Found. Comput. Sci.*, pages 45–52, 1982.
[609] Zh.I. Reznikova. *Animal Intelligence: From Individual to Social Cognition*. Cambridge Univ. Press, 2007.
[610] Zh.I. Reznikova and B.Ya. Ryabko. Analysis of the language of ants by information-theoretical methods. *Problems Inform. Transmission*, 22:245–249, 1986.
[611] J.J. Rissanen. Modeling by the shortest data description. *Automatica*, 14:465–471, 1978.
[612] J.J. Rissanen. A universal prior for integers and estimation by minimum description length. *Ann. Statist.*, 11:416–431, 1983.
[613] J.J. Rissanen. Stochastic complexity. *J. Royal Stat. Soc., Ser. B*, 49:223–239, 1987. Discussion: pages 252-265.
[614] J.J. Rissanen. *Stochastical Complexity and Statistical Inquiry*. World Scientific, Singapore, 1989.
[615] J.J. Rissanen. Complexity of models. In W.H. Zurek, editor, *Complexity, Entropy and the Physics of Information*, pages 117–125. Addison-Wesley, New York, 1991.
[616] J.J. Rissanen. Fisher information and stochastic complexity. *IEEE Trans. Inform. Theory*, 42(1):40–47, 1996.
[617] J.J. Rissanen. Stochastic complexity in learning. *J. Comput. Syst. Sci.*, 55:89–95, 1997.
[618] J.J. Rissanen. Hypothesis selction and testing by the MDL principle. *Comput. J.*, 42(4):260–269, 1999.
[619] J.J. Rissanen. *Information and Complexity in Statistical Modeling*. Springer-Verlag, New York, 2007.
[620] J.J. Rissanen and I. Tabus. Kolmogorov structure function in MDL theory and lossy data compression. In P. Grünwald, I.J. Myung, and M.A.

Pitt, editors, *Advances in Minimum Description Length: Theory and Applications*, pages 245–262. MIT Press, 2005.

[621] E. Rivals and J.-P. Delahaye. Optimal representation in average using Kolmogorov complexity. *Theor. Comput. Sci.*, 200:261–287, 1998.

[622] H. Rogers, Jr. *Theory of Recursive Functions and Effective Computability*. McGraw-Hill, 1967.

[623] A.E. Romashchenko, A.K. Shen, and N.K. Vereshchagin. Combinatorial interpretation of Kolmogorov complexity. *Theor. Comput. Sci.*, 271(1-2):111–123, 2002.

[624] D. Ronneburger. *Kolmogorov complexity and derandomization*. PhD thesis, Rutgers, the State Univ., New Jersey, 2004.

[625] P.W.K. Rothemund and E. Winfree. The program-size complexity of self-assembled squares. In *Proc. 32nd ACM Symp. Theory of Computing*, pages 459–468, 2000.

[626] R. Rubinstein. *Structural complexity classes of sparse sets: intractability, data compression and printability*. PhD thesis, Northeastern Univ., 1988.

[627] B.Ya. Ryabko. Encoding of combinatorial sources and Hausdorff dimension. *Dokl. Akad. Nauk SSSR*, 27:1066–1070, 1984.

[628] B.Ya. Ryabko. Noiseless coding of combinatorial sources, Hausdorff dimension, and Kolmogorov complexity. *Problems Inform. Transmission*, 22:170–179, 1986.

[629] B.Ya. Ryabko. The complexity and effectiveness of prediction algorithms. *J. Complexity*, 10:281–295, 1994.

[630] B.Ya. Ryabko and J. Astola. Universal code as a basis for time series testing. *Statistical Methodology*, 3:375–397, 2006.

[631] B.Ya. Ryabko, J. Astola, and A. Gammerman. Application of Kolmogorov complexity and universal codes to identity testing and nonparametric testing of serial independence of time series. *Theor. Comput. Sci.*, 359(1-3):440–448, 2006.

[632] B.Ya. Ryabko and V.A. Monarev. Experimental investigation of forecasting methods based on data compression algorithms. *Problems Inform. Transmission*, 41(2):65–69, 2005.

[633] B.Ya. Ryabko and Z. Reznikova. Using Shannon entropy and Kolmogorov complexity to study the communicative system and cognitive capacities in ants. *Complexity*, 2(2):37–42, 1998.

[634] D. Ryabko and M. Hutter. On sequence prediction for arbitrary measures. In *Proc. IEEE Int. Symp. Inform. Theory*, pages 2346–2350, 2007.

[635] C.C. Santos, J. Bernardes, P.M.B. Vitányi, and L. Antunes. Clustering fetal heart rate tracings by compression. In *Proc. 19th IEEE Int. Symp. Computer-Based Medical Systems*, pages 685–670, 2006.

[636] R. Schack. Algorithmic information and simplicity in statistical physics. *Int. J. Theor. Phys.*, 36:209–226, 1997.

[637] R. Schack and C.M. Caves. Chaos for Liouville probability densities. *Physical Review E*, 53(4):3387–3401, 1996.

[638] R. Schack and C.M. Caves. Information-theoretic characterization of quantum chaos. *Physical Review E*, 53(4):3257–3270, 1996.

[639] R. Schack, G.M. D'Ariano, and C.M. Caves. Hypersensitivity to perturbation in a quantum kicked top. *Physical Review E*, 50:972, 1994.

[640] E.D. Scheirer. Structured audio, Kolmogorov complexity, and generalized audio coding. *IEEE Trans. Speech Audio Process.*, 9(8):914–931, 2001.

[641] J. Schmidhuber. Discovering neural nets with low Kolmogorov complexity and high generalization capability. *Neural Networks*, 10(5):857–873, 1997.

[642] J. Schmidhuber. Low-complexity art. *Leonardo, J. Int. Soc. Arts, Sciences, and Technology*, 30(2):97–103, 1997.

[643] J. Schmidhuber. Hierarchies of generalized Kolmogorov complexities and nonenumerable universal measures computable in the limit. *Int. J. Found. Comput. Sci.*, 13(4):587–612, 2002.

[644] J. Schmidhuber. The speed prior: a new simplicity measure yielding near-optimal computable predictions. In *Proc. 15th Conf. Comput. Learning Theory*, pages 216–228, 2002.

[645] J. Schmidhuber. The new AI: General & sound & relevant for physics. In B. Goertzel and C. Pennachin, editors, *Artificial General Intelligence*, pages 175–198. Springer-Verlag, 2006.

[646] J. Schmidhuber, J. Zhao, and M. Wiering. Shifting inductive bias with success-story algorithm, adaptive Levin search, and incremental self-improvement. *Machine Learning*, 28(1):105–130, 1997.

[647] C.P. Schnorr. Eine Bemerkung zum Begriff der zufälligen Folge. *Z. Wahrscheinlichkeitstheorie verw. Gebiete*, 14:27–35, 1969.

[648] C.P. Schnorr. Klassifikation der Zufallsgesetze nach Komplexität und Ordnung. *Z. Wahrscheinlichkeitstheorie verw. Gebiete*, 16:1–21, 1970.

[649] C.P. Schnorr. Über die Definition von effektiven Zufallstests, i-ii. *Z. Wahrscheinlichkeitstheorie verw. Gebiete*, 15:297–312, 313–328, 1970.

[650] C.P. Schnorr. Optimal Gödel numberings. In *Proc. 1971 IFIP Congress, TA-2*, pages 12–14, Ljubljana, Yugoslavia, 1971.

[651] C.P. Schnorr. A unified approach to the definition of random sequences. *Math. Systems Theory*, 5:246–258, 1971.

[652] C.P. Schnorr. *Zufälligkeit und Wahrscheinlichkeit; Eine algorithmische Begründung der Wahrscheinlichkeitstheorie*, volume 218 of *Lect. Notes Math.* Springer-Verlag, Berlin, 1971.

[653] C.P. Schnorr. Process complexity and effective random tests. *J. Comput. System Sci.*, 7:376–388, 1973.

[654] C.P. Schnorr. *Rekursive Funktionen und ihre Komplexität.* Teubner, 1974.

[655] C.P. Schnorr. A survey of the theory of random sequences. In R.E. Butts and J. Hintikka, editors, *Basic Problems in Methodology and Linguistics*, pages 193–210. D. Reidel, 1977.

[656] C.P. Schnorr. A review of the theory of random sequences. In *Proc. 5th Int. Congr. Logic, Meth. Phil. of Sci.*, London, Ontario, August 1975.

[657] C.P. Schnorr and P. Fuchs. General random sequences and learnable sequences. *J. Symbolic Logic*, 42:329–340, 1977.

[658] C.P. Schnorr and H. Stimm. Endliche Automaten und Zufallsfolgen. *Acta Informatica*, 1:345–359, 1972.

[659] C.P. Schnorr and G. Stumpe. A characterization of complexity sequences. *Z. Math. Logik und Grudl. Math.*, 21:47–56, 1975.

[660] U. Schöning. Resolution proofs, exponential lower bounds, and Kolmogorov complexity. In *Proc. Symp. Math. Foundat. Comput. Sci.*, volume 1295 of *Lect. Notes Comput. Sci.*, pages 110–116, Berlin, 1997. Springer-Verlag.

[661] U. Schöning. Construction of expanders and superconcentrators using Kolmogorov complexity. *Rand. Struct. Alg.*, 17:64–77, 2000.

[662] U. Schöning and R. Pruim. *Gems of Theoretical Computer Science*. Springer-Verlag, 1998.

[663] R. Schuler. A note on universal distributions for polynomial-time computable distributions. In *12th IEEE Conf. Comput. Complexity*, pages 69–73, 1997.

[664] R. Schuler. Universal distributions and time bounded Kolmogorov complexity. In *16th Sypm. Theor. Aspects Comput. Sci.*, volume 1563 of *Lect. Notes Comput. Sci.*, pages 434–443, Berlin, 1999. Springer-Verlag.

[665] P. Schweitzer. Using the incompressibility method to obtain local lemma results for ramsey-type problems. *Inform. Process. Lett.* To appear.

[666] J. Seiferas. A simplified lower bound for context-free-language recognition. *Inform. Contr.*, 69:255–260, 1986.

[667] J. Shallit and Y. Breitbart. Automaticity: Properties of a measure of descriptional complexity. *J. Comput. System Sci.*, 53(1):10–25, 1996.

[668] J. Shallit and M. Wang. Automatic complexity of strings. *J. Automata, Lang. Combinat.*, 6(4):537–554, 2001.

[669] C.E. Shannon. The mathematical theory of communication. *Bell System Tech. J.*, 27:379–423, 623–656, 1948.

[670] C.E. Shannon. A universal Turing machine with two internal states. In C.E. Shannon and J. McCarthy, editors, *Automata Studies*, pages 129–153. Princeton University Press, 1956.

[671] C.E. Shannon. Coding theorems for a discrete source with a fidility criterion. In *IRE National Convention Record, Part 4*, pages 142–163, 1959.

[672] A.K. Shen. The frequency approach to defining a random sequence. *Semiotika i Informatika*, 19:14–42, 1982. In Russian.

[673] A.K. Shen. The concept of Kolmogorov (α, β)-stochasticity and its properties. *Soviet Math. Dokl.*, 28:295–299, 1983.

[674] A.K. Shen. Algorithmic variants of the notion of entropy. *Soviet Math. Dokl.*, 29(3):569–573, 1984.

[675] A.K. Shen. Connections between different algorithmic definitions of randomness. *Soviet Math. Dokl.*, 38(2):316–319, 1989.

[676] A.K. Shen. Discussion on Kolmogorov complexity and statistical analysis. *Comput. J.*, 42(4):340–342, 1999.

[677] A.K. Shen. Multisource information theory. In *Proc. 3rd Conf. Theory Appl. Models Comput.*, volume 3959 of *Lect. Notes. Comput. Sci,*, pages 327–338, Berlin, 2006. Springer-Verlag.

[678] A.K. Shen, V.A. Uspensky, and N.K. Vereshchagin. *Kolmogorov Complexity and Randomness*. To appear.

[679] A.K. Shen and N.K. Vereshchagin. Logical operations and Kolmogorov complexity. *Theor. Comput. Sci.*, 271(1-2):125–129, 2002.

[680] A.N. Shiryaev. A.N. Kolmogorov: Life and creative activities. *Ann. Probab.*, 17:866–944, 1989. Publications of Kolmogorov: pages 945–964.

[681] M. Sipser. A complexity theoretic approach to randomness. In *Proc. 15th ACM Symp. Theory Comput.*, pages 330–335, 1983.

[682] S.S. Skiena. Further evidence for randomness in π. *Complex Systems*, 1:361–366, 1987.

[683] D. Sleator, R. Tarjan, and W. Thurston. Short encodings of evolving structures. *SIAM J. Discrete Math.*, 5:428–450, 1992.

[684] R.J. Solomonoff. The mechanization of linguistic learning. In *2nd Int. Congress on Cybernetics*, pages 180–193, 1958.

[685] R.J. Solomonoff. A new method for discovering the grammars of phrase structure languages. In *Information Processing*, pages 285–290, Paris, 1959. Unesco.

[686] R.J. Solomonoff. A preliminary report on a general theory of inductive inference. Technical Report ZTB-138, Zator Company, Cambridge, Mass., November 1960.

[687] R.J. Solomonoff. An inductive inference code employing definitions. Technical Report ZTB-141, Rockford Research, Cambridge, Mass., April 1962.

[688] R.J. Solomonoff. A formal theory of inductive inference, part 1 and part 2. *Inform. Contr.*, 7:1–22, 224–254, 1964.

[689] R.J. Solomonoff. Inductive inference research status, spring 1967. Technical report, Rockford Research Inst., July 1967. Distributed by Clearinghouse, US Dept. of Commerce.

[690] R.J. Solomonoff. Inductive inference theory—a unified approach to problems in pattern recognition and artificial intelligence. In *4th Int. Conf. Artificial Intelligence*, pages 274–280, Tbilisi, Georgia, USSR, 1975.

[691] R.J. Solomonoff. Complexity based induction systems: comparisons and convergence theorems. Technical Report RR-329, Rockford Research, Cambridge, Mass., August 1976.

[692] R.J. Solomonoff. Complexity-based induction systems: comparisons and convergence theorems. *IEEE Trans. Inform. Theory*, 24:422–432, 1978.

[693] R.J. Solomonoff. Perfect training sequences and the costs of corruption—a progress report on inductive inference research. Memorandum, Oxbridge Research, P.O. box 559, Cambridge, Mass. 02238, August 1982.

[694] R.J. Solomonoff. Optimum sequential search. Memorandum, Oxbridge Research, P.O. box 559, Cambridge, Mass. 02238, June 1984.

[695] R.J. Solomonoff. An application of algorithmic probability to problems in artificial intelligence. In L.N. Kanal and J.F. Lemmer, editors, *Uncertainty in Artificial Intelligence*, pages 473–491. North-Holland, 1986.

[696] R.J. Solomonoff. The application of algorithmic probability to machine learning. Grant proposal manuscript, Oxbridge Research, P.O. box 559, Cambridge, Mass. 02238, September 1988.

[697] R.J. Solomonoff. A system for machine learning based on algorithmic probability. In *Proc. 6th Israeli Conf. on AI and Computer Vision*, 1989.

[698] R.J. Solomonoff. The discovery of algorithmic probability. *J. Comput. System Sci.*, 55:73–88, 1997.

[699] R.J. Solomonoff. Two kinds of probabilistic induction. *Comput. J.*, 42:256–259, 1999.

[700] R.J. Solomonoff. The probability of "undefined" (non-converging) output in generating the universal probability distribution. *Inform. Process. Lett.*, 106(6):238–240, 2008.

[701] R.M. Solovay. Lecture notes on algorithmic complexity. Unpublished, UCLA, 1975.

[702] R.M. Solovay. On random r.e. sets. In A.I. Arruda et al., editor, *Non-Classical Logic, Model Theory and Computability*, pages 283–307. North-Holland, 1977.

[703] D. Sow and A. Eleftheriadis. Complexity distortion theory. *IEEE Trans. Inform. Theory*, 49(3):604–608, 2003.

[704] L. Staiger. Complexity and entropy. In *Proc. Math. Found. Comput. Sci.*, volume 118 of *Lect. Notes Comput. Sci.*, pages 508–514, Berlin, 1981. Springer-Verlag.

[705] L. Staiger. Representable P. Martin-Löf tests. *Kybernetika*, 21:235–243, 1985.

[706] L. Staiger. Kolmogorov complexity and Hausdorff dimension. *Inform. Comput.*, 120(2):159–194, 1993.

[707] L. Staiger. A tight upper bound on Kolmogorov complexity by Hausdorff dimension and uniformly optimal prediction. *Theory Comput. Syst.*, 31(3):215–229, 1998.

[708] L. Staiger. The Kolmogorov complexity of real numbers. *Theor. Comput. Sci.*, 284(2):455–466, 2002.

[709] L. Staiger. Constructive dimension equals Kolmogorov complexity. *Inform. Process. Lett.*, 93:149–153, 2005.

[710] J. Storer. *Data Compression: Method and Theory*. Computer Science Press, 1988.

[711] E. Sverdrup. Tests without power. *Scand. J. Stat.*, 2:158–160, 1975.

[712] L. Szilard. On the decrease of entropy in a thermodynamic system by the intervention of intelligent beings. *Z. Phys.*, 53:840–856, 1929.

[713] V.M. Tikhomirov. The life and work of Andrei Nikolaevich Kolmogorov. *Russian Math. Surveys*, 43(6):1–39, 1988.

[714] M.R. Titchener, R. Nicolescu, L. Staiger, A. Gulliver, and U. Speidel. Deterministic complexity and entropy. *Int. J. Found. Comput. Sci.*, 64(1-4):443–461, 2005.

[715] B.A. Trakhtenbrot. A survey of Russian approaches to perebor (brute-force-search) algorithms. *Ann. Hist. Comput.*, 6:384–400, 1984.

[716] J.T. Tromp. Binary lambda calculus and combinatory logic. Manuscript, CWI, Amsterdam, 2004.

[717] A.M. Turing. On computable numbers with an application to the Entscheidungsproblem. *Proc. London Math. Soc., Ser. 2*, 42:230–265, 1936. Correction, *Ibid.*, 43:544–546, 1937.

[718] J. Tyszkiewicz. The Kolmogorov expression complexity of logics. *Inform. Comput.*, 135(2):113–135, 1997.

[719] J. Tyszkiewicz. A note on the Kolmogorov data complexity and nonuniform logical definitions. *Inform. Process. Lett.*, 64(4):187–195, 1997.

[720] J. Tyszkiewicz. On the Kolmogorov expressive power of Boolean query languages. *Theor. Comput. Sci.*, 190(2):317–361, 1998.

[721] V.A. Uspensky. Complexity and entropy: an introduction to the theory of Kolmogorov complexity. In O. Watanabe, editor, *Kolmogorov Com-*

plexity and Computational Complexity, pages 85–102. Springer-Verlag, Berlin, 1992.

[722] V.A. Uspensky. Kolmogorov and mathematical logic. *J. Symb. Logic*, 57(2):385–412, 1992.

[723] V.A. Uspensky and A.L. Semenov. *Algorithms: Main Ideas and Applications*. Kluwer Academic Publishers, Dordrecht, Netherlands, 1993. Also: in Lect. Notes Comput. Sci., vol. 122, A.P. Ershov and D.E. Knuth, editors, Springer-Verlag, 1981, pp. 100–234.

[724] V.A. Uspensky, A.L. Semenov, and A.K. Shen. Can an individual sequence of zeros and ones be random? *Russian Math. Surveys*, 45(1):121–189, 1990.

[725] V.A. Uspensky and A.K. Shen. Relations between varieties of Kolmogorov complexities. *Math. Systems Theory*, 29:271–292, 1996.

[726] M.A. Ustinov. Non-approximability of the randomness deficiency function. In *Proc. Int. Comput. Sci. Symp. Russia (CSR)*, volume 3967 of *Lect. Notes. Comput. Sci,*, pages 364–368, Berlin, 2006. Springer-Verlag.

[727] L.G. Valiant. A theory of the learnable. *Comm. Assoc. Comput. Mach.*, 27:1134–1142, 1984.

[728] P.A. van der Helm. Simplicity versus likelihood in visual perception: from surprisals to precisals. *Psychological Bull.*, 126(5):770–800, 2000.

[729] P.J. van Heerden. A general theory of prediction. Technical report, Polaroid Corporation, Cambridge, Mass., 1963.

[730] M. van Lambalgen. *Random Sequences*. PhD thesis, Universiteit van Amsterdam, Amsterdam, 1987.

[731] M. van Lambalgen. Von Mises' definition of random sequences reconsidered. *J. Symbolic Logic*, 52:725–755, 1987.

[732] M. van Lambalgen. Algorithmic Information Theory. *J. Symbolic Logic*, 54:1389–1400, 1989.

[733] V.N. Vapnik and A.Ya. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and Its Applications*, 16(2):264–280, 1971.

[734] J.S. Varré, J-P. Delahaye, and E. Rivals. Transformation distances: a family of dissimilarity measures based on movements of segments. *Bioinformatics*, 15(3):194–202, 1999.

[735] N.K. Vereshchagin. Kolmogorov complexity conditional to large integers. *Theor. Comput. Sci.*, 271(1-2):58–67, 2002.

[736] N.K. Vereshchagin. Kolmogorov complexity of enumerating finite sets. *Inform. Process. Lett.*, 103(1):34–39, 2007.

[737] N.K. Vereshchagin and P.M.B. Vitányi. Kolmogorov complexity, rate distortion and denoising of individual data. *IEEE Trans. Inform. Theory*. Submitted; Arxiv cs.IT/0411014.

[738] N.K. Vereshchagin and P.M.B. Vitányi. Kolmogorov's structure functions and model selection. *IEEE Trans. Inform. Theory*, 50(12):3265–3290, 2004.

[739] N.K. Vereshchagin and M.V. Vyugin. Independent minimum length programs to translate between given strings. *Theor. Comput. Sci.*, 271(1-2):131–143, 2002.

[740] J. Ville. *Étude Critique de la Notion de Collectif*. Gauthier-Villars, Paris, 1939.

[741] P.M.B. Vitányi. On the simulation of many storage heads by one. *Theor. Comput. Sci.*, 34:157–168, 1984.

[742] P.M.B. Vitányi. On two-tape real-time computation and queues. *J. Comput. Sys. Sci.*, 29:303–311, 1984.

[743] P.M.B. Vitányi. An $n^{1.618}$ lower bound on the time to simulate one queue or two pushdown stores by one tape. *Inform. Process. Lett.*, 21:147–152, 1985.

[744] P.M.B. Vitányi. Square time is optimal for the simulation of a pushdown store by an oblivious one-head tape unit. *Inform. Process. Lett.*, 21:87–91, 1985.

[745] P.M.B. Vitányi. Andrei Nikolaevich Kolmogorov. *CWI Quarterly*, 1(2):3–18, June 1988. Also: Scholarpedia, p. 8546.

[746] P.M.B. Vitányi. Multiprocessor architectures and physical law. In *Proc. 2nd IEEE Workshop Phys. Comput.*, pages 24–29. IEEE Comput. Soc. Press, 1994.

[747] P.M.B. Vitányi. Physics and the new computation. In *Proc. 20th Int. Symp. Math. Found. Comput. Sci.*, volume 969 of *Lect. Notes Comput. Sci.*, pages 106–128. Springer-Verlag, Berlin, 1995.

[748] P.M.B. Vitányi. Randomness. Arxiv preprint math/0110086, 1996.

[749] P.M.B. Vitányi. A discipline of evolutionary programming. *Theor. Comput. Sci.*, 241(1-2):3–23, 2000.

[750] P.M.B. Vitányi. Quantum Kolmogorov complexity based on classical descriptions. *IEEE Trans. Inform. Theory*, 47(6):2464–2479, 2001. Correction, *Ibid*, 48(4):1000, 2002.

[751] P.M.B. Vitányi. Asshuku ni motozuita hanyou na ruijido sokuteihou. *Surikagaku*, 519:54–59, September 2006. Translated into Japanese by O. Watanabe, English title: Universal similarity based on compression.

[752] P.M.B. Vitányi. Meaningful information. *IEEE Trans. Inform. Theory*, 52(10):4617–4626, 2006.

[753] P.M.B. Vitányi. Algorithmic chaos and the incompressibility method. In E. Charpentier, A. Lesne, and N.K. Nikolski, editors, *Kolmogorov's Heritage in Mathematics*, pages 301–317. Springer-Verlag, Berlin, 2007.

[754] P.M.B. Vitányi. Analysis of sorting algorithms by Kolmogorov complexity (a survey). In I. Csiszár, G.O.H. Katona, and G. Tardós, editors, *Entropy, Search, Complexity*, number 16 in Bolyai Society Mathematical Studies, pages 209–232. Springer-Verlag, Berlin, 2007.

[755] P.M.B. Vitányi, F.J. Balbach, R.L. Cilibrasi, and M. Li. Normalized information distance. In M. Dehmer and F. Emmert-Streib, editors, *Information Theory and Statistical Learning*. Springer-Verlag, New-York. To appear.

[756] P.M.B. Vitányi and M. Li. Algorithmic arguments in physics of computation. In *Proc. 4th Workshop Alg. Data Struct.*, volume 955 of *Lect. Notes Comput. Sci.*, pages 315–333. Springer-Verlag, Berlin, 1995.

[757] P.M.B. Vitányi and M. Li. Minimum description length induction, Bayesianism, and Kolmogorov complexity. *IEEE Trans. Inform. Theory*, 46(2):446–464, 2000.

[758] R. von Mises. Grundlagen der Wahrscheinlichkeitsrechnung. *Mathemat. Zeitsch.*, 5:52–99, 1919.

[759] R. von Mises. *Probability, Statistics and Truth.* Macmillan, 1939. Reprint: Dover, 1981.

[760] J. von Neumann. Various techniques used in connection with random digits. In A.H. Traub, editor, *John von Neumann, Collected Works, volume V.* Macmillan, 1963.

[761] V.G. Vovk. Algorithmic information theory and prediction problems. In M.I. Kanovich et al., editor, *Complexity Problems of Mathematical Logic*, pages 21–24. Kalininsk. Gos. Univ., Kalinin, 1985. In Russian.

[762] V.G. Vovk. The law of the iterated logarithm for random Kolmogorov, or chaotic, sequences. *SIAM Theory Probab. Appl.*, 32(3):413–425, 1987.

[763] V.G. Vovk. On a randomness criterion. *Soviet Math. Dokl.*, 35:656–660, 1987.

[764] V.G. Vovk. Prediction of stochastic sequences. *Problems Inform. Transmission*, 25:285–296, 1989.

[765] V.G. Vovk. Universal forecasting algorithms. *Inform. Comput.*, 96:245–277, 1992.

[766] V.G. Vovk and A. Gammerman. Complexity approximation principle. *Comput. J.*, 42(4):318–322, 1999.

[767] V.G. Vovk, A. Gammerman, and C. Saunders. Machine-learning applications of algorithmic randomness. In *Proc. 16th Int. Conf. Machine Learning*, pages 444–453, 1999.

[768] V.G. Vovk and C. Watkins. Universal portfolio selection. In *Proc. 11th Conf. Comput. Learning Theory*, pages 12–23. ACM Press, 1998.

[769] M.V. Vyugin. Information distance and conditional complexities. *Theor. Comput. Sci.*, 271(1-2):145–150, 2002.

[770] M.V. Vyugin. Systems of strings with high mutual complexity. *Probl. Inform. Transmission*, 39(4):88–92, 2003.

[771] M.V. Vyugin and V.V. Vyugin. Predictive complexity and information. *J. Comput. Syst. Sci.*, 70(4):539–554, 2005.

[772] V.V. Vyugin. Algorithmic entropy (complexity) of finite objects, and its application to defining randomness and quantity of information. *Semiotika and Informatika*, 16:14–43, 1981. In Russian. Translated into English in: *Selecta Mathematica* formerly *Sovietica*, 13:4(1994), 357–389.

[773] V.V. Vyugin. The algebra of invariant properties of binary sequences. *Problems Inform. Transmission*, 18:147–161, 1982.

[774] V.V. Vyugin. On nonstochastic objects. *Problems Inform. Transmission*, 21:3–9, 1985.

[775] V.V. Vyugin. On the defect of randomness of a finite object with respect to measures with given complexity bounds. *SIAM Theory Probab. Appl.*, 32:508–512, 1987.

[776] V.V. Vyugin. Bayesianism: an algorithmic analysis. *Inform. Comput.*, 127(1), 1996.

[777] V.V. Vyugin. Effective convergence in probability and an ergodic theorem for individual random sequences. *SIAM Theory Probab. Appl.*, 42(1):39–50, 1997.

[778] V.V. Vyugin. Ergodic theorems for individual random sequences. *Theor. Comput. Sci.*, 207(4):343–361, 1998.

[779] V.V. Vyugin. Non-stochastic infinite and finite sequences. *Theor. Comput. Sci.*, 207(4):363–382, 1998.

[780] V.V. Vyugin. Algorithmic complexity and stochastic properties of finite binary sequences. *Comput. J.*, 42(4):294–317, 1999.

[781] V.V. Vyugin. Most sequences are stochastic. *Inform. Comput.*, 169(2):252–263, 2001.

[782] V.V. Vyugin. Does snooping help? *Theor. Comput. Sci.*, 276(1-2):407–415, 2002.

[783] V.V. Vyugin. On complexity of easy predictable sequences. *Inform. Comput.*, 178(1):241–252, 2002.

[784] V.V. Vyugin. Suboptimal measures of predictive complexity for absolute loss function. *Inform. Comput.*, 175(2):146–157, 2002.

[785] V.V. Vyugin and V.P. Maslov. Extremal relations between additive loss functions and the Kolmogorov complexity. *Probl. Inform. Transmission*, 39(4):71–87, 2003.

[786] A. Wald. Sur la notion de collectif dans la calcul des probabilités. *Comptes Rendus des Scéances de l'Académie des Sciences*, 202:1080–1083, 1936.

[787] A. Wald. Die Wiederspruchsfreiheit des Kollektivbegriffes der Wahrscheinlichkeitsrechnung. *Ergebnisse eines mathematischen Kolloquiums*, 8:38–72, 1937.

[788] C.S. Wallace. *Statistical and Inductive Inference by Minimum Message Length*. Springer-Verlag, New York, 2005.

[789] C.S. Wallace and D.M. Boulton. An information measure for classification. *Comput. J.*, 11:185–195, 1968.

[790] C.S. Wallace and D.L. Dowe. Mimimum message length and Kolmogorov complexity. *Comput. J.*, 42(4):270–283, 1999.

[791] C.S. Wallace and D.L. Dowe. Refinements of MDL and MML coding. *Comput. J.*, 42(4):330–337, 1999.

[792] C.S. Wallace and P.R. Freeman. Estimation and inference by compact coding. *J. Royal Stat. Soc.*, 49:240–251, 1987. Discussion: pages 252–265.

[793] X.J. Wang. Intermittent fluctuations and complexity. In W.H. Zurek, editor, *Complexity, Entropy and the Physics of Information*, pages 319–330. Addison-Wesley, New York, 1991.

[794] Y. Wang. The law of the iterated logarithm for p-random sequences. In *Proc. 11th IEEE Conf. Structure in Complexity Theory*, pages 180–189, 1996.

[795] Y. Wang. A separation of two randomness concepts. *Inform. Process. Lett.*, 69(3):115–118, 1999.

[796] O. Watanabe. Comparison of polynomial time completeness notions. *Theor. Comput. Sci.*, 53:249–265, 1987.

[797] O. Watanabe, editor. *Kolmogorov Complexity and Computational Complexity*. Springer-Verlag, Berlin, 1992.

[798] M. Wax and I. Ziskind. Detection of the number of coherent signals by the MDL principle. *IEEE Trans. Acoust. Speech Signal Process.*, ASSP-37(8):1190–1196, 1989.

[799] S. Wehner. Analyzing worms and network traffic using compression. *J. Computer Security*, 15(3):303–320, 2007.

[800] H.S. White. Algorithmic complexity of points in dynamical systems. *Ergodic Theory and Dynamical Systems*, 13:807–830, 1993.

[801] D.G. Willis. Computational complexity and probability constructions. *J. Assoc. Comput. Mach.*, 17:241–259, 1970.

[802] C.H. Woo. Laws and boundary conditions. In W.H. Zurek, editor, *Complexity, Entropy and the Physics of Information*, pages 127–135. Addison-Wesley, New York, 1991.

[803] E.H. Yang. Universal almost sure data compression for abstract alphabets and arbitrary fidelity criterions. *Probl. Contr. Inform. Theory*, 20(6):397–408, 1991.

[804] E.H. Yang and S.Y. Shen. Distortion program-size complexity with respect to a fidelity criterion and rate-distortion function. *IEEE Trans. Inform. Theory*, 39(1):288–292, 1993.

[805] L. Yu, D. Ding, and R. Downey. The Kolmogorov complexity of random reals. *Ann. Pure Appl. Logic*, 129:163–180, 2004.

[806] X. Zhang, Y. Hao, X. Zhu, and M. Li. Information distance from a question to an answer. In *Proc. 13th ACM SIGKDD Int. Conf. Knowledge Discov. Data Mining*, pages 874–883. ACM Press, 2007.

[807] I.G. Zhurbenko. *The Spectral Analysis of Time Series*, pages 231–236. Series in Statistics and Probability. North-Holland, 1986. Appendix II: Kolmogorov's algorithm of the Random Number Generator.

[808] M. Zimand. On the topological size of sets of random strings. *Zeitschr. f. math. Logik und Grundlagen d. Math.*, 32:81–88, 1986.

[809] M. Zimand. A high-low Kolmogorov complexity law equivalent to the 0-1 law. *Inform. Process. Lett.*, 57:59–64, 1996.

[810] M. Zimand. Large sets in AC0 have many strings with low Kolmogorov complexity. *Inform. Process. Lett.*, 62:165–170, 1997.

[811] J. Ziv. On the complexity of an individual sequence. *IEEE Trans. Inform. Theory*, 22:74–88, 1976.

[812] J. Ziv. Distortion-rate theory for individual sequences. *IEEE Trans. Inform. Theory*, 26(2):137–143, 1980.

[813] J. Ziv and A. Lempel. Compression of individual sequences via variable-rate encoding. *IEEE Trans. Inform. Theory*, 24:530–536, 1978.

[814] W.H. Zurek. Algorithmic randomness and physical entropy. *Physical Review, Ser. A*, 40(8):4731–4751, 1989.

[815] W.H. Zurek. Thermodynamic cost of computation, algorithmic complexity and the information metric. *Nature*, 341:119–124, 1989.

[816] W.H. Zurek. Algorithmic information content, Church-Turing thesis, physical entropy, and Maxwell's demon. In W.H. Zurek, editor, *Complexity, Entropy and the Physics of Information*, pages 73–89. Addison-Wesley, New York, 1991.

[817] W.H. Zurek, editor. *Complexity, Entropy and the Physics of Information*. Addison-Wesley, New York, 1991.

[818] W.H. Zurek. Decoherence, chaos, quantum-classical correspondence, and the algorithmic arrow of time. *Physica Scripta*, T76:186–198, 1998.

[819] W.H. Zurek. Quantum discord and Maxwell's demons. *Physical Review, Ser. A*, 67:012320, 2003.

[820] A.K. Zvonkin and L.A. Levin. The complexity of finite objects and the development of the concepts of information and randomness by means of the theory of algorithms. *Russian Math. Surveys*, 25(6):83–124, 1970.

Index

$(n)_k$: number of variations, 9
$0'$: Turing degree halting problem, 231
A^*: set of all finite sequences of elements of set A, 12
$A^{=n}$: set of all words of length n in A, 550
A^∞: set of one-way infinite sequences over set A, 14
$A^{\leq n}$: set of words of length $\leq n$ in A, 111
C: complexity, 106
$C(x|l(x))$: length-conditional C, 119
$C(x;n)$: uniform complexity, 130
$C[f(n), t(n), s(n)]$, 542
C^+: monotone upper bound on C-complexity, 217
C^s: space-bounded version of $C^{t,s}$, 534
C^t: time-bounded version of $C^{t,s}$, 534
$C^{t,s}$: time-space-bounded C complexity, 532
C_r:complexity of r-ary strings, 115
C_ϕ: complexity with respect to ϕ, 105
Cn: normalized complexity for reals, 132
E_0: information distance, 642
E_1: max distance, 643
E_2: reversible computation distance, 649
E_3: reversible sum distance, 651
E_4: sum distance, 652
E_5: min distance, 674
H: entropy stochastic source, 67
$I(X;Y)$: probabilistic mutual information, 71
$I(x : y)$: algorithmic mutual information $K(y) - K(y \mid x)$, 248
$I(x;y)$: algorithmic mutual information $K(x) + K(y) - K(x,y)$, 252
$I_C(x{:}y)$: algorithmic information in x about y, 186
K: diagonal halting set, 34
$K(K(x)|x)$: complexity of the complexity function, 242
$K(x|l(x))$: length-conditional K, 204
$K(x)$: prefix complexity, 202
K^+: monotone upper bound on K-complexity, 212
$K^{t,s}$: time-space-bounded K complexity, 535

K_0: halting set, 34
$L(x)$: uniform discrete distribution on $\mathcal{N}$, 23
$O(f(x))$: at most of order of magnitude $f(x)$, 16
Δ_i^p: class in polynomial hierarchy, 40
Δ_n^0: class in arithmetic hierarchy, 47
Γ_x: cylinder generated by x, 14
$l^*(x)$: optimal universal code-word length, 82
$\ell^*(x)$: lower bound on $l^*(x)$, 82
$\ell^k(x,\epsilon)$: upper bound on $l^*(x)$, 309
$\Omega(f(x))$: at least of order of magnitude $f(x)$, 16
Ω: halting probability, 226, 496
Π_i^p: class in polynomial hierarchy, 40
Π_n^0: class in arithmetic hierarchy, 47
Σ_i^p: class in polynomial hierarchy, 40
Σ_n^0: class in arithmetic hierarchy, 47
$\Theta(f(x))$: of order of magnitude $f(x)$, 16
$\bar{x}$: prefix-code $1^{l(x)}0x$ for x, 13
$\varnothing$: empty set, 7
$\exists$: there exists, 8
$\exists^\infty$: there exist infinitely many, 8
$\forall$: for all, 8
$\forall^\infty$: for all but finitely many, 8
$\lambda(\omega)$: uniform continuous distribution on $[0,1)$, 23
$\langle\cdot\rangle$: pairing function, 7
$\lceil\cdot\rceil$: ceiling of a number, 8
$\lfloor\cdot\rfloor$: floor of a number, 8
ln: natural logarithm, 8
log-DNF, **376**, 381
log-decision list, 381
log: binary logarithm, 8
$\log^* x$: number of terms in $l^*(x)$, 82
ω: infinite sequence of elements of $\mathcal{B}$, 14
$\phi(x) < \infty$: $\phi(x)$ converges, 8
$\phi(x) = \infty$: $\phi(x)$ diverges, 8
ρ_0: universal integral test, 223
σ-algebra, 20
$d(A)$: cardinality of set A, 7
e: normalized information distance, 661
e_G: normalized web distance, 668
e_Z: normalized compression distance, 664
$e_{\min}$: normalized min distance, 674
$l(x)$: length of string x, 13
n-cell, 691
n-string, **119**, 122, 129, 130, 132, 159, 193
$n(T)$: index of T, 30
$o(f(x))$: asymptotically less than $f(x)$, 16
x: finite sequence of elements of $\mathcal{B}$, 13
x^*: first shortest program for x in enumeration order, 110
x^R: reverse of string x, 13
x_i: ith letter of x, 13
$x_{1:n}$: first n letters of x, 13
$\mathbf{M}$: universal lower semicomputable continuous semimeasure, 294
$\mathbf{M}_{\text{norm}}$: Solomonoff measure, **303**, 324, 325
$\mathbf{M}c$: universal lower semicomputable extension semimeasure, 326
$\mathbf{m}$: universal lower semicomputable discrete semimeasure, **267**, 268
$\mathcal{B}$: basic elements, **12**, 262
$\mathcal{N}$: the nonnegative integers, 7
$\mathcal{Q}$: the rational numbers, 7
$\mathcal{R}$: the real numbers, 7
$\mathcal{Z}$: the integers, 7
$CD[f(n),t(n),s(n)]$, 544
CD^s: space-bounded version of $CD^{t,s}$, 534
CD^t: time-bounded version of $CD^{t,s}$, 534, 536
$CD^{t,s}$: resource-bounded accepting complexity, 533

$CU[f(n), t(n), s(n)]$, 547
$KD^{t,s}$: K version of $CD^{t,s}$, 535
KM: negative logarithm of $\mathbf{M}(x)$, 305
Kc: Chaitin's conditional prefix complexity, 252
Km: monotone complexity, 305
Kt: Levin-complexity, 577–582
ic^t: instance complexity, 571
$\binom{n}{k}$: number of combinations, 9
$|\cdot|$: absolute value of a number, 8

Aanderaa, S.O., 525, 528
Abel, N.H., 90
acceptable numbering, **41**, 112
accepting a language, 37
Ackermann, W., 45
Adleman, L.M., 596, 599, 709
Adriaans, P., 431, 439
Agafonov, V.N., 59, 185
Agrawal, M., 529
Aho, T., 719
Aldana, M., 719
Aleksandrov, P.S., 97
Alfonseca, M, 673, 719
algorithmic complexity theory, ix, 101–257
algorithmic entropy, *see* entropy, algorithmic
algorithmic information theory, *see* information theory, algorithmic
algorithmic probability theory, 259–338
algorithmic statistic, 402
Allender, E., 529, 570, 582, 597, 600
Allison, L., 436
Alon, N., 457, 463, 526
Ané, C., 718
Andreev, A.E., 597
Andrews, J., 184
Andrzejak, R.G., 718
Angluin, D., 94, 434
Anthony, M., 434
ants, 711
Antunes, L., 586, 588, 589, 719
Archimedes, 49
Aristotle, 341, 431
arithmetic hierarchy, 46
Asarin, E.A., 173, 174
Asmis, E., 431
Astola, J., 720
asymptotic notation, 15–17
Aumann, Y., 597
average-case
  adder design, 444–445
  complexity, 290–294, 330–331, 334, 444–445, 448–451, 461–490
  Dobosiewicz sort, 485–486
  Heapsort, 476–480
  longest common subsequence, 486–489
  Quicksort, 485
  routing in networks, 473
  Shakersort, 485–486
  Shellsort, 480–486
  shortest common supersequence, 489
Avogadro's number, 679

Bachman, P., 15
Bacon, F., 713, 722
Badger, J.H., 672, 717
Baeza-Yates, R.A., 489
Balcázar, J.L., 92, 381, 569, 597, 600
Baranyai, Zs., 468
Barendregt, H.P., 207
Barron, A.R., 436
Barzdins's lemma, 131, **178**, 180, 181, 195, 239, 245, 496, 537, 539, 594
Barzdins, J.M., 116, 178, 180, 185, 195, 434, 496, 539, 546, 596
basic element, **12**, 262, 294
Bassalygo, L.A., 459
Bayes's rule, **20**, 19–20, 60, 62, 65, 94, 96, 323, 332, 333, 343–347, 349, 360, 363, 384, 398, 432, 435
Bayes, T., 60, 343, 347, 432
Bayesian reasoning, 343–347
Beame, P., 515
Becher, V., 237
Beigel, R., 525

Ben-Amram, A.M., 501, 530
Ben-David, S., 588
Benedek, G., 380, 434
Benedetto, D., 718
Benioff, P.A., 165, 633
Bennamoun, M., 719
Bennett, C.H., 183, 195, 228, 255, 570, 589, 595, 600, 626, 633, 640, 641, 658, 659, 688, 715–717, 720
Berger, T., 95
Berman, P., 5, 569
Berman–Hartmanis conjecture, 563
Bernardes, J., 719
Bernoulli process, **59**, 63–65, 192, 284, 323, 324
Bernoulli, J., 59, 63
Bernstein, E., 709
Berry, G.G., 177
Berthiaume, A., 709, 722
betting, 284–287
Bienvenu, L., 231
Biggs, N., 434
binary interval, 274, 306
binomial coefficient, 9
bit: binary digit, vii
Blum, M., 550
Blumer, A., 380, 434
Bogolyubov, N.N., 97
Bollobás, B., 526
Boltzmann constant, 629, 682
Boltzmann, L., 678
Bolyai, J., 95
Bolyai, W., 95
Book, R.V., 530, 570, 597, 598, 600
Boolean formula, 376, 568
Boolean matrix rank, 445
Boppana, R., 526
Borel, E., 20, 165, 238
Borel–Cantelli lemmas, **64**, 159
Boswell, J., 259, 331
Boulton, D.M., 435
bra-ket notation, 696
Brady, A.H., 46
Brebner, G., 459
Breitbart, Y., 530
Brejová, B., 486
Brewer, R.G., 720
Briegel, H.J., 722
Briley, B.E., 525
Brouwer, L.E.J., 164
Brownian computer, 633
Bruijn sequence, 528
Bruijn, N.G. de, 528
Buck, M., 672
Buhrman, H.M., 123, 468, 469, 473, 519–521, 525, 526, 529, 536, 559, 570, 577, 588, 596–598, 641, 716
Burks, A.W., 444, 525, 715

Caglioti, E., 718
Cai, J.-Y., 132, 561
Calude, C., 158, 228, 235, 236
Cantelli, F.P., 64
Cantor, D.G., 455
Cantor, G., 40
Cardano, G., 23, 331
cardinality, **7**, 13
Carnap, R., 96, 332, 347
Carnot cycle, 675
Carnot, N.L.S., 675
Cartesian product, 7
Case, J., 718
Castro, J., 381
Cauchy–Schwarz inequality, 530
Caves, C.M., 625, 714, 722
Cebrián, M, 673, 719
Chaitin, G.J., 5, 91, 96, 99, 104, 123, 132, 160, 186, 192, 194, 195, 206, 207, 213, 214, 221, 224, 228, 232, 233, 237–240, 254–256, 327, 335, 435, 493, 600
Champernowne's number, *see* sequence, Champernowne
Champernowne, D.G., 54, 93
characteristic function, *see* function, characteristic
characteristic sequence, 125, **178**, 538
  of K_0, **180**, 245
  of a language, **492**, 496, 561, 600

of high Turing degree set, 183
of hyperimmune set, 185
of immune set, 184
of not recursively enumerable set, **179**, 181
of recursively enumerable set, **178**, 180, 181, 184, 239, 245, 537, 539, 546, 594
of semirecursive set, 183
random, 569
Chater, N., 437, 722
Chen, X., 672, 717
Chernoff bounds, **61**, 167, 346, 462, 471, 567
Chernoff, H., 94
Chernov, A.V., 717
Chervonenkis, A.Ya., 434
Chitescu, I., 158
Chomsky hierarchy, 490
Chomsky, N., 332
Chor, B., 588
Chrobak, M., 500
Church random sequence, *see* sequence, Mises–Wald–Church random
Church's thesis, 24, **29**, 53
Church, A., 24, 35, 42, 51, 53, 93, 156
Chvátal, V., 434, 489
Cilibrasi, R.L., 670, 673, 718, 719
classification, 669
clause, 376
Clementi, A.E.F., 597
clique, 454, 457
clustering
hierarchical, 666
CNF formula, 376
coarse-graining, 681
code
additively optimal universal, 253
ASCII, 74
asymptotically optimal universal, **81**, 82
average word length, **77**, 95, 199
data-to-model, 402
fixed-length, 74
Hamming, 123
instantaneous, *see* code, prefix
MDL, 407
model, 402
Morse, **66**, 73
optimal prefix, 77
prefix, 5, 13, 15, 68, **75**, 73–90, 94, 199
self-delimiting, 79
Shannon–Fano, **68**, 79, 87, 94, 274, 280, 327, 593, 604, 625, 668
two-part, 107–108, 721
uniquely decodable, **74**, 77, 88
universal, **81**, 79–82, 88, 95, 278
variable-length, 75
code sequence, 74
code word, **13**, 73
Cohen, P., 98
coin-weighing problem, 455
collective, 51, **51**, 53, 55, 57, 93, 143, 155, 163
combination, 9
combinatorics, 8–12, 92, 451–460
combinatory logic, 207, 255
communication complexity, 516–521
noncommunicable string, 520
protocol independent, 519–521
randomized, 521
Complearn toolkit, 719
complexity
C, **106**, 103–195, 206
C^s, **534**, 542
C^t, 534, 536–542, 546, 550–562
$C^{t,s}$, **532**, 532–536, 542
K, *see* complexity, prefix
$K^{t,s}$, 535, 546
K_μ, 327
r-ary strings C, 114
CD^s, 534
CD^t, **534**, 550–562, 571
$CD^{t,s}$, **533**, 533–536, 546

Ct, 581
KD^s, 546
$KD^{t,s}$, 535, 546
KM, **305**, 328
Kc, **252**, 253, 254
Km, *see* complexity, monotone
Kt, **578**, 577–582
additivity of, **109**, 118, 191, 197, 202, 245–248
additivity of C, 195
additivity of K, **248**, 251
additivity of Kc, 252
algorithmic, 1, 66, 96, 187
alternative approach to define it, 205
approximation of, 125, 127
average-case, **291**, 290–294, 330–331, 334
conditional C, **106**, 119
conditional K, 202
continuity of, 121, 128
expected C, 122, **188**, 603–608
expected K, 189, **246**, 603–608
extension, 216
fluctuation of, 128
incomputability of, 127, 193
instance, **571**, 571–577
length-conditional C, **119**, 122, 127, 129, 130, 132, 160, 162, 193
length-conditional K, **204**, 213, 217
lower bound on C, 126
lower bound on $C(x|l(x))$, **127**, 129
lower bound on K, 216
majorant of, **261**, 537
monotone, 206, 221, 225, **305**, 305–308, 328, 329, 335, 336
monotonic upper bound on K, **212**, 213, 217, 240, 241
monotonicity on prefixes, 119, 197, 199, 220
normalized for real numbers, 132
number of states, 184
of complexity function, 182, **242**, 241–245, 254, 256
of function, 116, **242**
prefix, 104, 132, **202**, 197–257, 334, 432
quantum Kolmogorov, 696–711
relation between C and K, 215
resource bound hierarchies, 542–545
resource-bounded, x, 97, 531–600
space, 37
state–symbol product, **91**, 90–92, 95, 99
stochastic, 435
time, 37
time-bounded uniform, 546
time-space-bounded, 532–536
uniform, **130**, 132, 160, 162, 180, 184, 193, 197, 206, 237, 308, 546, 547, 596
worst-case, **291**, 330
complexity class
Δ_2^{E}, 568
Δ_i^p, 40
Π_i^p, **40**, 597
Σ_i^p, **40**, 597
#P, 560, 561
BPP, **554**, 555, 597, 598
DSPACE, **38**, 545, 550, 598
DTIME, **38**, 548, 561, 563, 564, 569, 598
E, **564**, 565, 568, 570, 598
ESPACE, 598
EXPTIME, 569
IC[log, poly], 573
NE, 568
NP, **38**, 534, 555, 560, 563, 564, 568, 573, 581, 599
NSPACE, **38**, 569
NTIME, 38
P, **38**, 560, 563, 568, 573, 581, 598
P/log, 575

P/poly, **560**, 575
PSPACE, **38**, 560, 569
R, 555
complexity oscillation, 98, **143**, 143–147, 155, 159, 194, 198, 218, 220, 221, 224, 231
of K, 225, 229, 232, 233
of Km, 225, 335
compression, 343, 347, 348, 350, 362, 383, 423, 430, 433, 439, 620, 621, 664–674
in nature, 711–714
lossy, 405, 613
computability theory, 24–47, 92
computable majorants, 536–542
computational complexity, x, 37–40, 562–577, 598
computational depth, 586
computational learning theory, ix, 6, 370–382
concatenation, 12
context-free language, 581
convergence
apparent, of relative frequency, 142
recursive, of series, **152**, 155, 159, 162, 229
regulator of, 160
Cook, S.A., 599
Copolov, D.L, 436
counting method, 453
Cover, T.M., 90, 94, 95, 97, 113, 141, 214, 216, 218, 255, 324, 327, 337, 338, 366, 432, 433, 436, 439, 625, 714
crossing sequence, 442
Csiszár, I., 123, 366
Culik II, K., 483
Cutler, C.C., 715
Cuturi, M., 718
cylinder, **14**, 21, 55, 144, 148, 380

D'Ariano, G.M, 722
Dŭriš, P., 512, 528
Daley, R.P., 156, 160–162, 183, 546, 547, 596
Dančik, V., 489
data
(α, β)-stochastic, 424, **424**, 439
nonstochastic, **413**, 419, 424, 425, 438, 439
overfitting, 407
data-processing inequality, *see* inequality, data-processing
de la Vallée Poussin, C.J.G.N, 4
decision list, 381
decision tree, 393–397
degree of unsolvability, 44
Dekker, J.C.E., 43, 45
Delahaye, J.P., 717
Demaine, E.D., 501
DeMarrais, J., 709
denoising, 620–622
derandomization, 567–568
DeSantis, A., 433
Devroye, L., 485
Dewdney, A.K., 46
DFA, *see* finite automaton
Diacz, S., 237
diagonalization method, 34
Diaz, J., 92
Dietzfelbinger, M., 510, 511
dimension
Hausdorff, 132
topological, 132
Vapnik–Chervonenkis, 380
Ding, D., 237
Ding, Y.Z., 598
Diophantine equation, **179**, 180, 239, 256
Dirac, P., 696
distance, 646
χ^2, 328
Euclidean, 366
Hamming, **616**, 647
Hellinger, 328, 329, **353**
information, *see* information distance
Kullback–Leibler, *see* Kullback–Leibler divergence
max, **643**, 641–648
min, 673

mixed use NID approximations, 671
normalized compression, **664**, 663–667
normalized information, 661
normalized max, 661
normalized min, 671, **674**
normalized sum, 661, **672**, 717
normalized web, **668**, 667–671
reversible, **649**, 648–650
sum, **651**, 650–653
distortion
Euclidean, 617, 627–628
Hamming, 615–617, 620, 626–627
list, 424, 615–616
distortion ball, 614
distribution
binomial, 61, 346
Bose–Einstein, 11, 282
computable universal, 582–589
conditional universal, 275
Fermi–Dirac, 11, 281
malign, 586
Maxwell–Boltzmann, 11
normal, 390, 392
of description length, **211**, 255, 277, 287, 288
simple, 374
uniform, **21**, 69, 78, 136, 138, 368
uniform continuous, 23
uniform discrete, **23**, 283
universal, 6, **273**, 266–302, 330–331, 344, 605
universal time-limited, 582–589
distribution-free learning, 370–382
DNF formula, 376
Dobosiewicz, W., 486
Doerr, B., 483
Doob, J.L., 322, 337
Downey, R.G., 194, 232, 236, 237, 256
Drexler, K.E., 717
Duns Scotus, John, 63, 431
Durand, B., 125

Edmonds, J.E., 39
effective enumeration, 29
Ehrenfeucht, A., 380, 434, 490
Eleftheriadis, A., 629, 715
element-distinctness problem, 508
Elias, P., 57, 88, 95
Emanuel, K., 718
ensemble, 66, 685
entropy, **67**, 65–73, 78, 80, 87, 156, 187, 188, 192, 195, 198, 199, 246, 603–608
n-cell algorithmic, 692
algorithmic, **692**, 686–696, 720, 721
Boltzmann, 678–685
classical, 675–678
coarse-grained algorithmic, **693**, 694
complexity, 687
conditional, **70**, 68–70, 246
Gibbs, **686**, 685–687, 693
joint, 246
of English, 113
of Russian, 94
physical, 720
relation with complexity, 603–608
Epicurus, 339, 341, 343, 347, 431
Erdős, P., 94, 453–455, 457, 458, 463, 526
estimator
best-fit, 404
maximum likelihood, 405
minimum description length, 406
event, 18
certain, **18**, 21
impossible, **18**, 21
mutually independent, 20
probability of, 18

Fano, R.M., 94
Feder, M., 437
Feigenbaum, J., 597
Feldman, J., 722
Felker, J.H., 715

Feller, W., 10–12, 22, 23, 64, 65, 92, 93, 331
Fenner, S., 597
Ferguson, T.S., 57
Fermat, P. de, 23, 179
Fermi, E., 96, 720
Ferragina, P., 665, 719
Feynman, R.P., 633
Fich, F., 516
field, 19
  Borel, 20
  Borel extension, 21
  probability, 19
Fine, T.L., 93, 96, 142, 432
Finetti, B. de, 432
finite automaton
  k-head DFA, 499
  k-pass DFA, 500
  deterministic (DFA), **341**, 447, 500
  nondeterministic (NFA), 447
  sweeping two-way DFA, 501
Fisher, R.A., 83, 90, 95, 398, 437
Floyd, R.W., 476, 527
Ford, J., 721
Fortnow, L., 123, 192, 529, 559, 560, 567, 575, 576, 586, 588, 589, 597, 598
Foulser, D., 490
Fouz, M., 457, 460, 483
fractal, 132
Francia, B., 718
Fredkin gate, 631, 633
Fredkin, E., 715, 716
Fredman, M.L., 501
Freeman, P.R., 435
Freivalds, R.V., 434, 447, 524
frequency, 67
  a priori, 289
  lower, 289
  relative, 51
frequency interpretation of probability, 50, 93
Friedberg, R.A., 42, 44
Frost, S.D.W., 718
Fu, B., 570
function
  Ackermann, **45**, 89, 308, 309
  additively optimal, **103**, 107
  additively optimal partial recursive prefix, 202
  busy beaver, **46**, 130, 185, 325
  canonical rate-distortion, 614
  characteristic, **8**, 32, 371, 571
  coenumerable, *see* function, upper semicomputable
  complexity, 205
  composition of, 8
  computable, **36**, 35–37, 310–316
  computable in the limit, 672
  consistent, 571
  convergence of, 8
  decoding, 13, 73
  distance, 646
  distortion, 613
  distortion-rate, 613
  distribution, 22
  divergence of, 8
  encoding, 73
  enumerable, *see* function, lower semicomputable
  factorial, **9**, 17
  following shape, 407
  generalized exponential, 45
  hash, 654
  honest, **562**, 564
  incomputable, *see* function, nonrecursive
  inverse of, 8
  lower semicomputable, **35**, 135, 148, 149, 151, 222, 234, 260–262, 310–316
  many-to-one, 8
  minimum description length (MDL), 407
  monotone, **299**, 300, 301
  nonrecursive, 46, 174, 186, 241
  one-to-one, 8
  pairing, 7
  parity, 524
  partial, 7, 8
  partial recursive, **29**, 116
  partial recursive prefix, 200
  payoff, **285**, 284–287, 319
  predicate, 29

predictor, 58
primitive recursive, 89
probability density, 22
probability mass, **22**, 328
ranking, 566
rate-distortion, 613
recursive, **29**, 41, 46, 53, 116, 133, 331
recursive real, 36
regular, 300
semicomputable, **36**, 35–37, 92, 310–316, 331
shape match, 133
structure, *see* structure function
successor, 29
total, 8
total recursive, *see* function, recursive
unit integrable, 310–316
universal, **103**, 104, 105, 107
universal lower semicomputable, 260
universal partial recursive, 31
universal upper semicomputable, 261
upper semicomputable, **35**, 174, 205, 216, 222, 328

Gödel number, 30
Gödel numbering, *see* numbering, acceptable
Gödel, K., 3, 33, 34, 95, 175, 177, 195, 612, 714
Gács, P., 97, 115, 141, 173, 182, 183, 185, 191, 193–195, 206, 214, 218, 219, 231, 241, 245, 251, 254–257, 288, 289, 304, 307, 324, 328, 335, 337, 370, 421, 425, 426, 433, 434, 439, 597, 599, 600, 625, 626, 658, 659, 696, 708, 711, 714, 716, 717, 721, 722
Gabarró, J., 92
Gaifman, H., 164
Galil, Z., 500, 509, 512, 528–530
Gallager, R.G., 88, 94
Gallaire, H., 527
Gammerman, A., 720
Gao, Q., 436
garbage bits, 631
Gardner, M., 228, 255
Garey, M.R., 92
Gasarch, W., 182, 183, 525, 526, 560, 672
Gavaldà, R., 489, 569, 570
Gavoile, C., 475
Gell-Mann, M., 430, 721
generalized Kolmogorov complexity, *see* complexity, resource-bounded
genericity, 99
Geréb-Graus, M., 500
Giancarlo, R., 665, 719
Gibbs, J.W., 685
Gill, J, 570
Gnedenko, B.V., 97
Gold, E.M., 363, 434
Goldbach conjecture, 228
Goldberg, A., 556, 560, 597
Goldreich, O., 588
Goldsmith, J., 567
Goldstine, H.H., 444, 525
Good, I.J., 432
Grünwald, P.D., 436
Graham, R., 17
graph
expander, 459
labeled, 461
OR-concentrator, 459
random, *see* Kolmogorov random graphs
random directed, 457
statistics subgraphs, 464
tournament, *see* tournament
undirected, 457
unlabeled, *see* number of unlabeled graphs
Grassberger, P., 718
Gray, R.M., 97, 141, 338, 439, 625, 714
Greco, V., 665, 719
Griffiths, T.L., 722
Grimmett, G., 458
Grossman, J.W., 45
Grumbach, S., 717

Gurevich, Y., 599

Hühne, M., 510
Hadamard, J.S., 4
Hahn, E.L., 720
halting probability, **226**, 225–228, 232, 233, 235, 237, 238, 255, 262, 272, 594
halting problem, **33**, 33–35, 42, 182, 186, 226, 245, 249, 368, 414, 710
Hamiltonian equations, 681
Hammer, D., 192, 530, 626, 714
Hancock, T., 397
Handley, J, 667, 719
Hanson, N.R., 342
Hao, Y., 719
Harary, F., 465
Hardy, G.H., 16
Harrison, M.A., 495, 500, 527
Hartle, J.B., 721
Hartmanis, J., 92, 132, 508, 528, 550, 568, 596, 597
Haussler, D., 380, 434
Heilbronn's triangle problem, 458
Heilbronn, H.A., 458
Heim, R., 142, 337
Hemachandra, L., 561, 597, 599
Hemaspaandra, E., 570
Hennie, F.C., 525, 528, 532
Hermo, M., 600
Hertling, P., 235
Heyting, A., 164
Hilbert's tenth problem, 180
Hilbert, D., 45, 180, 612
Hirschfeldt, D.R., 194, 232, 256
Hochberg, J.E., 437
Hoeffding, W., 57
Hoepman, J.H., 469, 473, 526
Homer, 101
Honavar, V., 382
Hood, L.E., 719
Hopcroft, J.E., 527
Huang, M.-D.A, 709
Hume, D., 347
Hurewicz, W., 132
Hutter, M., 355, 358, 362, 367–369, 432, 433, 580, 582, 599

Huynh, D.T., 561, 562
Hwang, K., 525
hypotheses identification, 363–364
Håstad, J., 529

Ibarra, O.H., 500
ideal MDL, 423
immune
  bi, 562
  P/poly, 562
Impagliazzo, R., 570, 597
incompressibility method, 441–530
induction, 339
  in recursion theory, 363–366, 369–370
inductive inference, 339
  Gold paradigm, 363, **364**, 434
inductive reasoning, ix, 6, 59, 96, 332, 339–432
  using **M**, 350–360
inequality
  data-processing, **72**, 610
  information, 72
  Kraft, *see* Kraft inequality
inference, 82
information, 66
  algorithmic, 186
  algorithmic conditional mutual, 248
  algorithmic conditional mutual using $K(c)$, 252
  algorithmic mutual, 189–191, **248**, 247–254, 289, 608–613, 661, 714, 717
  algorithmic mutual using Kc, 252
  conservation inequalities, *see* inequality, information conservation
  dispersal of, 124
  in x about y, 246, 248
  maximal mutual, 653
  nongrowth, 610
  probabilistic mutual, **71**, 69–72
  symmetry of probabilistic, *see* symmetry

of probabilistic information
information distance, 602, **642**, 641–674
admissible, 646
normalized, 660–674
information inequality, *see* inequality, information
information theory, 48, 65–90, 94, 186, 187
algorithmic, 186–193, 199, 244–254, 602–629
instance complexity, *see* complexity, instance
instance complexity conjecture, 572
invariance theorems, *see* theorem, invariance
irreversible computation, 629, 630
Israeli, A., 515
Itai, A., 380, 434

Jürgenson, H., 228
Jaffe, J., 490
Jagota, A.K., 334
Jakoby, A., 334, 587
Janson, S., 458, 480, 527
Jaynes, E.T., 398, 437
Jenkins, S., 718
Jiang, T., 123, 397, 434, 458, 485, 489, 490, 500, 501, 513, 527
Jockusch, C.G., 183
Johnson, D.S., 92, 434
Johnson, Dr. Samuel, 259, 331
Jones, J.P., 239, 240, 256
Joseph, D., 575
Juedes, D.W., 536, 546, 598, 600
Jurdziński, T., 528
Jurka, J., 437

König's infinity lemma, 133
Kahn, J., 526
Kajan, L., 718
Kalyanasundaram, B., 459
Kamae, T., 59, 181, 213
Kanaya, J, 629, 715
Kannan, R., 509, 529
Kanovich, M.I., 184
Karp, R.M., 560
Kasami, T., 528
Katseff, H.P., 133, 159, 161, 194
Kauffman, S.A., 719
Kearney, P., 672, 717
Kearns, M., 434
Kemeny, J.G., 431
Keogh, E.J., 667, 719
Kertesz-Farkas, A., 718
Keuzenkamp, H.A., 437
Keyes, R.W., 716
Keynes, J.M., 56
Khintchin, A.I., 65, 88
Khoussainov, B., 235
Kim, C.E., 500
Kim, J., 526
Kirchherr, W.W., 469
Kirk, S.R., 718
Klauck, H., 519–521, 529
Kleene, S.C., 35, 41, 42
Knopp, K., 90
Knuth, D.E., xiii, 16, 17, 92, 93, 194, 480, 485, 486, 527
Ko, K.-I., 548, 575–577, 596, 598
Kobayashi, K., 334, 514, 586
Kocsor, A., 718
Kolmogorov Axioms, 18
Kolmogorov random graphs, 461–469
Kolmogorov structure function, *see* structure function
Kolmogorov, A.N., 18, 49, 50, 52, 53, 55, 56, 65, 66, 73, 92–99, 103, 104, 110, 125, 142, 156, 158, 173, 192–195, 221, 256, 290, 327, 331, 332, 336, 424, 438, 439, 531, 596, 599, 721
Komlós, J., 458, 501
Koppel, M., 430, 439
Koucký, M., 521, 570, 597
Kraft inequality, **76**, 76–78, 82, 88–90, 94, 199, 204, 211, 222, 223, 229, 247, 274, 647, 672
Kraft, L.G., 76, 94
Kranakis, E., 475, 476, 526, 527
Kraskov, A., 718

Krasnogor, N., 718
Kraus, B., 722
Kreinovich, V., 165
Krizanc, D., 475, 476, 527
Kučera, A., 236, 256
Kullback–Leibler divergence, **72**, 283, 326, 352
Kumar, V., 671
Kummer, M., 123, 181, 560, 576, 577, 598
Kurtz, S.A., 597
Kushilevitz, E., 529
Kwong, S., 672, 717

Löfgren, L., 193
López-Ortiz, A., 508
Lambalgen, M. van, 93, 195, 225, 229, 231, 233, 335, 526
Landauer, R., 629, 633, 715, 716
Lange, K.-J., 641, 716
language compression, **550**, 550–562
  optimal, **551**, 558–559
  P-rankable, 560
  probabilistic, 555–558
  ranking, **551**, 558–559
  with respect to C^p, 555–558
  with respect to CD^p, 551–555
Laplace, P.S., 20, 49, 60, 65, 259, 323, 331, 432
Laplante, S., 529, 559, 597, 709, 722
Larjo, A., 719
Lathrop, J.I., 600
law
  0-1, 530
  complete probabilities, 19
  excluded gambling strategy, 52, 53
  high-low Kolmogorov complexity, 530
  infinite recurrence, **58**, 156, 330
  inverse weak law of large numbers, 64
  iterated logarithm, 54, 55, 58, **65**, 93, 147, 156, 234, 238, 287, 329, 330
  of large numbers, 55, **63**, 147, 163, 284, 287
  of probability, 284, 287, 318
  of randomness, 55, 58, **147**
  of succession, **65**, 323, 356
  slow growth, **594**, 596
  strong law of large numbers, **64**, 330
  weak law of large numbers, **59**, 63, 64
learning
  log-DNF, 376–378
  log-DNF formula, 376
  log-decision list, 381
  by enumeration, 364
  decision list, 381
  decision tree, 393–397
  distribution-free, 373
  in the limit, 364
  monotone k-term DNF, 382
  simple DNF, 381
  under **M**, 379–380
  under **m**, 376–378
  under computable distributions, 373–382
Lecerf, Y., 715
Lee, S.H., 667, 719
Leeuw, K. de, 185
Leeuwenberg, E.L.J, 437
lemma
  Barzdins's, *see* Barzdins's lemma
  coding, 553
  honesty, 562, 564
  jamming, 503
  KC-regularity, 491
  Lovász's local, 460, 526
  pumping, 490
  switching, 521
Lengler, J., 483
Leung-Yan-Cheong, S.K., 95, 214, 216, 255
Levin, L.A., 92, 132, 163, 185, 191–195, 206, 221, 245, 251, 255, 257, 304, 306, 324, 325, 328, 331, 332, 334–338, 438, 546, 579, 589, 596, 599, 612, 613, 714

Levine, R.Y, 641, 716
Levy, M.A., 567
Levy, P., 335, 337
Lewis II, P., 550
Li, L., 597
Li, M., 123, 195, 294, 334, 375, 380–382, 397, 400, 433–436, 457, 458, 468, 483–485, 489, 490, 495, 500, 501, 508, 509, 515, 525–527, 529, 587, 600, 626, 641, 658, 659, 672, 674, 716, 717, 720
Li, X., 672, 718
lie-game, 483
Likharev, K., 716
Lindström, B., 455
Lipton, R.J., 560
List, B, 625
literal, 376
Littlestone, N., 434
Littlewood, J.E., 16, 93
Liu, W., 719
Lloyd, S., 721
logical depth, **591**, 589–596
  (d, b)-deep, **591**, 593
  machine-independent, 595
  of Ω, 594
  shallow, 593
  stability, 595
Lonardi, S., 667, 719
longest common subsequence, **486**, 486–489
Longpré, L., 165, 509, 510, 549, 570, 596, 598, 600
Lopez-Ortiz, A., 501
Loreto, V., 718
Loryś, K., 528
loss
  logarithmic, 358, 422
  measure, 358
Loui, M.C., 513, 514
Lovász, L., 434, 460
Loveland, D.W., 122, 130, 132, 156, 160, 162, 193, 237, 493, 496, 596
Low, L.H., 436
lower bounds, 469–530
  k-PDA, 500
  k-head automaton, 499
  k-pass DFA, 500
  Boolean matrix rank, 445
  circuit depth, 521–524
  converting NFA to DFA, 447
  for Turing machines, 502–514
  in formal language theory, 490–497
  index size for text, 501
  multihead automata, 499–501
  one-tape Turing machine, 442
  online CFL recognition, 497–499
  parallel computation, 514–516
  Ramsey theory, 454–455, 460, 490, 515, 526
  routing in networks, 472–473
  singly vs doubly linked list, 501
  string-matching, 500
  sweeping two-way DFA, 501
Luby, M., 588
Luccio, F.L., 476, 527
Lucier, B., 485
Lucretius, 340
Luginbuhl, D.R., 514
Luo, Z.Q., 458
Lutz, J.H., 133, 536, 546, 570, 598, 600

Müller, M, 722
Méré, Chevalier de, 23
Ma, B., 672, 717, 718
Maass, W., 508–511, 528, 529
machine
  k-pushdown store, 508
  k-queue, 509
  k-stack, 508, 512
  k-tape, 512
  deterministic Turing, **28**, 350
  Kolmogorov–Uspensky, 599
  monotone, **298**, 302, 306, 333, 334, 336
  nondeterministic 1-tape Turing, 509
  nondeterministic Turing, 37

offline Turing, 510
one-way Turing, 502–514
online Turing, **502**, 502–514
oracle Turing, 38
PRAM, 514
prefix, **201**, 333, 334
probabilistic Turing, 184, **184**, 447, 554
quantum Turing, 697, **700**, 700–701
reference monotone, 302
reference prefix, 202
reference quantum Turing, 704
reference Turing, 106, 107
Turing, **27**, 24–31, 37, 40, 90–92, 442
two-dimensional tape, 512
universal Turing, **30**, 90, 130, 192
macro state, 678, **681**
Mahaney, S., 567, 569
Mairson, H.G., 530
majorant of complexity, *see* complexity, majorant of
malign, *see* distribution, malign
Mamitsuka, H., 436
Mandelbrot, B., 132
Manzini, G., 665, 719
Margolus, N., 633
Markov process, 20, 350
Markov's inequality, 141, **282**, 286, 292, 356
Markov, A.A., 42, 192
Markowsky, G., 433
Martin-Löf, P., 54, 55, 98, 121, 133, 143, 156, 159, 160, 162–165, 194, 219, 221, 337
martingale, **320**, 335, 337, 598
Marxen, H., 46
matching, 644
Matijasevich, Yu.V., 180, 239, 240, 256
Matthew effect, 96
maximum likelihood estimator, *see* estimator, maximum likelihood
Maxwell's demon, 688–691
Maxwell, J.C., 688
Mayordomo, E., 133, 600
McAleer, M., 437
McAllister, E., 437
McCarthy, J., 332
McGorry, P.D., 436
McKenzie, D.P., 436
McKenzie, P., 641, 716
McKinnon, B., 718
McMillan, B., 88
measure, 19, 21, 348
computable, **36**, 264, 300, 324
computable continuous, 294–302, 305, 328, 359
computable discrete, 265–290
conditionally bounded away from zero, **359**, 360
constructive, 194
continuous, 21
countable, 21
defective, 331
discrete, 21
discrete computable, 245
discrete lower semicomputable, 245
Laplace, 356
Lebesgue, *see* measure, uniform
of random sequences, **153**, 229
probability, 263
recursive, *see* measure, computable
resource-bounded, 598
simple, 379
Solomonoff, **303**, 324, 325
uniform, 21, **264**, 300, 379, 541, 598
universal lower semicomputable, 379
Meertens, L.G.L.T., 17, 92
Mehlhorn, K., 530
Melkebeek, D. van, 586, 588, 589
Merkle, R.C., 634, 716
Merkle, W., 59, 157, 162, 183, 231, 236
metric, 646
Minkowski, 672

similarity, **661**, 661–663
Meyer auf der Heide, F., 515
Meyer, A.R., 132, 493
micro state, 679, **679**
Mihailovic, N., 183, 236
Miller, G.A., 670
Miller, J.S., 157, 215, 230, 231, 235, 237, 238, 256
Mills, W.H., 455
Milosavljević, A., 437
Miltersen, P.B., 334, 559, 586
minimal randomness deficiency function, 404
minimum description length premature termination, 416
minimum description length estimator, *see* estimator, minimum description length
Minsky, M., 31, 92, 97, 332, 333
Mises, R. von, 20, 50, 51, 53, 55–57, 60, 93–95, 97, 141, 143, 155, 158, 163, 165, 319, 337, 432
Mises–Wald–Church random sequence, *see* sequence, Mises–Wald–Church random
mistake bounds, *see* prediction, mistake bounds
Miyano, S., 500
Mocas, S., 549
model
  determinacy, 423
  finite set, 402
  not in model class, 418
  probability, **428**, 428–429
  total recursive function, **429**, 429–430, 439
model fitness, *see* estimator, best-fit
monomial, 376, 381
monotone k-term DNF, 382
Mooers, C., 333
Mora, C.E., 722
Moran, S., 475, 515
Moser, L., 455
Motwani, R., 460
Muchnik, A.A., 44, 158
Muchnik, An.A., 125, 158, 194, 215, 256, 290, 367–369, 426, 626, 659, 660, 714, 717
multinomial coefficient, **10**, 400
multiplication rule, 19
multiplicative domination, 266
Munro, I., 476, 527
Muramatsu, J., 629, 715
Muse, S.V., 718
mutual information, *see* information, [types of] mutual
Myung, I.J., 436

Naik, A., 597, 598
Natarajan, B.K., 434, 550
Navarro, G., 489
NCD, *see* distance, normalized compression
Nelson, C.G., 500
Neumann, J. von, xiii, 50, 57, 93, 444, 525, 629, 715
Newman, I., 526
Newman-Wolfe, R., 512, 525
Newton, I., vii, 9, 23, 341, 431
NFA, *see* finite automaton
NGD, *see* distance, normalized web
Nicholson, P., 457
Nies, A., 157, 194, 215, 230, 232, 238, 256
Nisan, N., 529
normalization of semimeasure, *see* semimeasure, normalization
NP-complete, **39**, 563, 564
NP-hard, 39
null set, 147
  Π^0_n-, 164
  μ-, 149
  constructive, **149**, 151, 152, 163
  total recursive, 163
number
  Ω-like real, **234**, 262
  arithmetically random real, 234, 235
  computable, 262

lower semicomputable, *see* sequence, lower semicomputable
lower semicomputable real, 324
nonrecursive real, 228
normal, *see* sequence, normal
of Wisdom, 228
prime, **4**, 17, 32
random real, 227, 235, 236, 324
recursive, *see* number, computable
recursive real, *see* sequence, recursive
transcendental, 227, 228
number of unlabeled graphs, 465–468
numbering, acceptable, 112
NWD, *see* distance, normalized web
Nykter, M., 719

O'Connor, M.G., 59
O'Neil, E.J., 447
O'Neil, P.E., 447
Oberschelp, A., 46
Occam algorithm, 372
Occam's razor, vii, 63, 260, 272, 323, **341**, 342, 358, 371, 431
occupancy number, 11
Ockham, William of, 63, 260, 341, 343, 347
Odifreddi, P., 41, 42, 45, 92, 182
oracle, **38**, 40, 534, 535, 555–557, 562–565, 568, 569
Baker–Gill–Solovay, 563
order
partial, 8
total, 8
order of magnitude symbols, 17
Oresme, N., 431
Orponen, P., 575–577, 597, 598
Ortega, A., 673, 719
Otu, H.H., 718
outcome of experiment, 18, 277, 278, 284, 358
overlap
maximal, 643–645
minimal, 653–657
Ozhegov, S.I., 94

P-isomorphic, 563
P-printability, 565
Pérennès, S., 475
Péter, R., 45
Pólya, G., 64
pac-learning, **371**, 370–382, 434
simple, 370, 373–382, 434
Palmer, E.M., 465
Pao, H.K., 718
paradox
Bertrand, 340
Richard–Berry, **1**, 177
Russell, 177
Parberry, I., 515
Parekh, R., 382
Parikh, R., 490
partition, 10
Pascal, B., 23
Patashnik, O., 17
Paterson, M., 489
Patrick, J.D., 436
Paturi, R., 512, 525
Paul, W.J., 512, 513, 524, 525, 528
Peano arithmetic, 35
Pearl, J., 434
Pednault, E.P.D., 436
Pelc, A., 483
Pelta, D.A., 718
Penrose, R., 207, 211
Pepys, S., 23
permutation, 8
perpetuum mobile
of the first kind, 675
of the second kind, 675
Petersen, H., 501
Peterson, G., 599
Petri, N.V., 185, 546
phase space, 679
phylogeny, 664
Pierce, J.R., 715
Pinto, A., 586
Pintz, J., 458
Pippenger, N., 455, 457, 514, 526
Pitt, L., 382

Pitt, M.A., 436
place-selection rule, 93, 161
  according to Kolmogorov, 56
  according to Kolmogorov–Loveland, 156
  according to Mises–Wald–Church, **53**, 156, 162
  according to von Mises, **52**, 156
  finite-state, 58
  total recursive function, 161
  Turing machine, 173
Plaxton, C.G., 484
Poland, J., 432
polynomial complexity core, 572
polynomial hierarchy, **40**, 570
polynomial many-to-one reduction, *see* reducibility, polynomial many-to-one
polynomial Turing reduction, *see* reducibility, polynomial Turing
Pond, S.L.K., 718
Pongor, S., 718
Poonen, P., 484
Popper, K.R., 343, 347
Positselsky, S.Y., 125, 215
Post's problem, 44
Post, E.L., 42–44, 92, 176
Pour-El, M.B., 92
Pratt, V.R., 184, 480
predicate, 29, 533
prediction, 348–370
  mistake bounds, 364–365, 369, 434
  snooping curve, 422–423, 439
  strategy, 422
prediction error, 6, 56, 59, 351–353, 355, 357, 432
  expected using **M**, 329
prefix-code, *see* code, prefix
Price, N.D., 719
Price, R., 432
principle
  indifference, 63, 340
  insufficient reason, 340
  maximum entropy, **398**, 398–400, 437
  maximum likelihood, **398**, 437
  minimum description length, 96, **382**, 382–399, 436, 437, 721
  minimum message length, 435
  multiple explanations, 339, 341
  pigeonhole, 441
  simplicity, 63
probabilistic communication complexity, 459
probabilistic method, 441, 450, 453, 454
probability
  a priori, *see* probability, prior
  algorithmic, 273
  conditional, **19**, 70
  function, *see* function, probability [types]
  inferred, *see* probability, posterior
  inverse, 60
  joint, 69
  marginal, 69
  posterior, **20**, 60, 64, 345, 347
  prior, **20**, 60, 65, 343, 345, 347, 349, 398, 399, 432
  prior continuous, 298–302
  uniform, 290
  universal prior, 63, 94, 96, 193, 198, 255, **272**, 273, 276, 283, 297, 302, 332–334, 344, 591
probability theory, 18–23, 92
Ptolomy, C., 431
pushdown automaton (PDA), 500, 528

quantum computation, 697–700
quantum interference, 699
quantum Kolmogorov complexity, *see* complexity, quantum Kolmogorov
quantum state, pure, 696

quantum Turing machine, *see* machine, quantum Turing
qubit, 697
queue, 509
Quinlan, J.R., 397, 436

Rényi, A., 455, 526
Rabin, M.O., 124, 525, 528, 598
Rado, T., 46
Ragde, P., 516, 526
Raghavan, P., 460
Rajski, C., 672
Ramsey number, 454
Ramsey, S.A., 719
random
  n-random sequence, 237
  2-random sequence, 237
  3-random sequence, 237
  Kolmogorov, 237
  strongly Chaitin, 238
random variable, 22
  (in)dependent, 22
  continuous, 22
  discrete, 22
randomness
  negative, 438
  positive, 438
randomness deficiency, 110, **120**, 120–121, 124, 125, 137, 139, 193, 219, 280, 281, 318, 328, 358, 402, 408, 424, 431, 448–451, 461
  for probability models, 428
  for total recursive function models, 429
rate distortion
  algorithmic, 613–624
  probabilistic, 85–86
Ratner, M., 94
Ravela, S., 718
Razborov, A., 529
recursion theory, *see* computability theory
recursively uniform limit, 128
reducibility
  many-to-one, **43**, 176, 184, 570
  one-to-one, 43
  polynomial many-to-one, **39**, 568, 598
  polynomial truth-table, 570
  polynomial Turing, **39**, 568
  self, 575
  truth-table, 570
  Turing, **43**, 176, 182, 183, 232, 570
  weak truth-table, 184
Regan, K.W., 334, 530, 598
regular language, 491
Reimann, J., 59, 157
Reisch, S., 459, 525
Reischuk, R., 334, 512, 513, 528, 587
relation
  n-ary, 7
  binary, 7
  encoding, 73
relative frequency stabilization, 142
resource-bounded Kolmogorov complexity, *see* complexity, resource-bounded
reversible
  ballistic computer, 632–634, 715
  Boolean gate, 631, 640, 716
  circuit, 631–632, 715
  computation, 629–641, 715
  Turing machine, 637–641, 715
Reznikova, Zh.I., 722
Richards, J.I., 92
Riemann hypothesis, 228
Risi, C., 718
Rissanen, J.J., 89, 95, 382, 390, 391, 435–437, 439
Rivals, É., 717
Rivest, R., 381, 397, 436, 500
Robbins, D., 672
Robinson, R.M., 45
Rockford Research, 333
Rogers, H., Jr., 41–47, 92, 112
Rolim, J.D.P., 597
Romashchenko, A.E., 607, 626, 714, 717
Ronneburger, D., 570, 581, 597

Rooij, S. de, 715
Rosenberg, A., 500
Roth, K.F., 458
routing in networks, 469–476
routing table, 469
Roy, S., 597
Rozenberg, G., 490
Rtanamahatana, C.A., 667, 719
Rubinstein, R., 597
Rudich, S., 529, 561
rule of succession, *see* law, of succession
run of zeros, 118
Russell, B., 177, 341
Russo, D., 597
Ryabko, B.Ya., 132, 720, 722
Ryabko, D, 433

Sakoda, W.J., 501
sample space, 6, **18**
  continuous, **18**, 20
  discrete, 18
Sanderson, M.J., 718
Sankoff, D., 489
Santos, C.C., 719
SAT, **39**, 560, 563, 564, 568, 573, 575, 578
satisfiable, 39
Savitch, W.J., 569
Sayood, K., 718
Schöning, U., 459, 575–577, 598, 600
Schack, R., 625, 714, 722
Schaffer, R., 527
Schay, G, 525
Scheihing, R., 489
Schindelhauer, C., 334, 587
Schmidhuber, J., 599
Schmidt, W.M., 458
Schmidt-Goettsch, K., 46
Schnitger, G., 459, 510, 511, 525
Schnorr's thesis, 164
Schnorr, C.P., 59, 115, 162–164, 194, 206, 221, 233, 235, 255, 256, 335–337
Schuler, R., 588
Schumacher, J., 720
Schweitzer, P, 460
Sedgewick, R., 527
Seiferas, J.I., 500, 512, 513, 525, 527
Seker, A., 718
self-delimiting code, *see* code, self-delimiting
Semenov, A.L., 158, 194, 256
semimeasure, **264**, 262–265, 331, 334
  computable, 264
  computable continuous, 328
  conditional, 350
  discrete, **265**, 265–290
  extension, 326
  lower semicomputable, 264
  lower semicomputable continuous, 294–302
  lower semicomputable discrete, 265–290
  maximal, *see* semimeasure, universal
  maximal lower semicomputable, *see* semimeasure, universal lower semicomputable
  normalization, **303**, 324, 325, 333, 335
  reference universal lower semicomputable continuous, 297
  relative enumerable, 289
  Solomonoff normalization, **303**, 302–304, 324–326
  universal, **266**, 294
  universal lower semicomputable, 255, 324, 334
  universal lower semicomputable conditional, 275
  universal lower semicomputable continuous, **294**, 294–298, 302, 325, 328
  universal lower semicomputable discrete, **267**, 268, 273, 276, 287, 386
  universal relative enumerable, 289
sequence

Δ^0_2-definable, 225, 232, **232**, 233
Π^0_n-random, 164
∞-distributed, *see* sequence, normal
μ-random, *see* sequence, random
k-distributed, 58
Bernoulli, **59**, 142, 163, 357
Champernowne, **54**, 59, 93, 165
characteristic, *see* characteristic sequence
computable, 262, 324
DNA, 589
effectively unpredictable, 238
finite, *see* string
hyperarithmetically random, 164
incompressible, 226
infinite, *see* sequence
Kolmogorov–Loveland random, 157
Kolmogorov–Loveland stochastic, **157**, 158
lower semicomputable, 161
Martin-Löf random, *see* sequence, random
Mises–Wald–Church random, **53**, 55, 58, 156, 238, 337, 547
Mises–Wald–Church stochastic, **156**, 157, 161, 162
nonrecursive, 231
normal, 58, 165, 238
pararecursive, 161
pseudorandom, 93
random, 55, 98, **149**, 143–165, 183, 194, 198, 219, 221, 222, 224, 225, 227–229, 233, 318, 328, 335
recursive, **47**, 113, 123, 131, 161, 231, 262, 289, 327, 540
Schnorr random, 163
Solovay random, **160**, 221, 233
strongly Chaitin random, 255
typical, 54
universal recursive, 289
von Mises random, *see* collective
weakly Chaitin random, **233**, 255

set
$C[f(n), t(n), s(n)]$, **542**, 542–550, 562–570
Q-immune, 545
m-complete, 176, 177, 184
r-complete ($r = 1, m, T$), 44
r-hard ($r = 1, m, T$), 44
$CD[f(n), t(n), s(n)]$, 544
$CU[f(n), t(n), s(n)]$, 546
arithmetic, 234
Borel, 164, 234
complete, 245
continuous, 8
countable, 8
cylinder, *see* cylinder
diagonal halting, **34**, 36, 44, 88, 182
effectively immune, 175, **182**
effectively simple, 175, **182**
empty, 7
enumerable without repetitions, 43
exponentially low, 564
fractal, 132
halting, **34**, 44, 176, 177, 180, 184
hyperarithmetic, 164
hyperimmune, 185
immune, **43**, 182, 184
intuitionistic measure zero, 164
Kolmogorov, 182
meager, **120**, 179, 193
P-printable, **565**, 581
RAND, 175, 182
recursive, 32
recursively enumerable, **32**, 178–180
recursively enumerable-complete, 563
relative enumerable, 289
semirecursive, 183

simple, **43**, 44, 45, 174, 176, 177, 181, 184
sparse, 194, 545, **556**, 559, 565, 566, 568
tally, 566
Turing complete, 182, 184
weak truth-table complete, 184
Shakespeare, W., 113
Shallit, J., 530
Shaltiel, R., 597
Shamir, A., 124, 561
Shannon, C.E., 48, 65, 66, 73, 77, 87, 90, 91, 94, 95, 98, 99, 101, 192, 199, 332, 720
shape following, *see* function, following shape
Shell, D.L, 480, 527
Shen, A.K., 125, 157, 158, 160, 192, 194, 206, 231, 256, 288, 337, 424, 429, 439, 530, 626, 660, 714, 717
Shen, S.-Y., 629, 715
Sherman, A.T., 641, 716
Shiryaev, A.N., 97, 98
Shmulevich, I, 719
shortest common supersequence, **486**, 489
shortest program, **110**, 118, 202, 213, 252, 253, 277, 287, 305
prefix complexity of, 213
similarity metric, *see* metric, similarity
Simon, J., 512, 525
Simons, G., 57
simple DNF, 381
Singh, B.S., 436
singly linked list, 501
Sipser, M., 133, 159, 161, 193, 194, 501, 556, 560, 596, 597
Sivakumar, D., 598
Skiena, S.S., 93
Slaman, T.A., 236, 256
Sleator, D., 483
Slotine, J.J., 721
Smith, C., 434
Smoluchowski, M. von, 57
Snell, J.L., 337
Snir, M., 164
snooping curve, *see* prediction, snooping curve
Sobolev, S.L., 97
Solomonoff measure, *see* measure, Solomonoff
Solomonoff normalization, *see* semimeasure, Solomonoff normalization
Solomonoff's induction theory, 348–370
Solomonoff's inductive formula, **358**, 432
Solomonoff, R.J., 59, 63, 94, 96–99, 104, 192–194, 199, 255, 303, 304, 325–327, 332, 333, 335, 347, 348, 352, 357, 367, 432, 433, 599
Solovay, R.M., 99, 132, 160, 181, 184, 203, 215, 217, 220, 221, 229, 232–235, 237, 238, 241, 255–257, 289, 290, 325, 335
sophistication, *see* model, total recursive function, 721
sorting
Bubblesort, 480, 484
Dobosiewicz sort, 485–486
Heapsort, 476–480, 527
Insertion sort, 480, 484
Queuesort, 484
Quicksort, 290–294, 485
Selection sort, 484
Shakersort, 485–486
Shellsort, 480–486
Stacksort, 484
source sequence, 74
source word, **13**, 73
Souto, A., 586
Sow, D.M., 629, 715
space–energy tradeoff, 641
Spencer, J.H., 94, 453, 457, 463, 483, 526
Srivastava, 671
Stögbauer, H., 718

stack, 460, 484, 508, 512
Staiger, L., 132
Stanat, D., 490
state space, 679
statistic
  algorithmic minimal sufficient, **403**, 403–430
  algorithmic sufficient, 108, **403**, 402–404, 406
  probabilistic, 83
  probabilistic minimal sufficient, 85
  probabilistic sufficient, 83–85
statistical inference, 82
statistical properties
  of graphs, 464–465
  of sequences, 165–174, 195
statistics
  algorithmic, 401–431
  nonprobabilistic, *see* statistics, algorithmic
  probabilistic, 82–85
Stearns, P., 453
Stearns, R., 508, 528, 532, 550
Stephan, F., 157, 232, 238
Steurer, D., 483
Stimm, H., 59
Stirling's formula, **17**, 67, 140, 188, 468
Stirling, J., 17
stochastic complexity, 435
stochastic source, 67
Stockmeyer, L, 588
string, 12
  C-random, 220
  K-random, 220
  δ-random, 124, 173
  c-incompressible, **116**, 140
  c-random, 140
  n-string, *see* n-string
  absolutely nonrandom, 125
  binary, 12–15
  compressibility of, 116
  cyclic shift, 162, 213
  empty, 12
  incompressibility of substring, 117
  incompressible, 123, 133, 142, 212, 218
  incompressible w.r.t. K, **212**, 219
  infinite, *see* sequence
  length of, 13
  random, 97, 121, **140**, 133–143, 194, 218–220
  reverse of, **13**, 109
  self-delimiting, 13
  strongly typical, 420
  typical, 125, **402**, 403–405, 419
structure function, **405**, 401–431, 519, 616
  bumps, 426
Sudborough, H., 500
Suel, T., 484
sufficiency line, **406**, 407, 419
sufficient statistic, *see* statistic, [types of] sufficient
supermartingale, 320
  universal lower semi-computable, **320**, 319–321
support vector machine (SVM), 669
Svetlova, N.D., 94
symmetry of information, 447
  C, 189–192, 195, 567
  K, **249**, 244–254
  Kc, 254
  algorithmic, 189, 190, **190**, 195, 244–254, 257
  algorithmic resource-bounded, 548
  conditional K, 252
  for K_μ, 327
  probabilistic, **71**, 327
Szemerédi, E., 458, 501, 509, 511, 529
Szilard engine, 688
Szilard, L., 688, 720

Tahi, F., 717
Tan, P.N., 671
Tang, S., 570
Tapp, A., 641, 716
Tarjan, R.E., 483, 485
Tenenbaum, J.B., 722
Terwijn, S.A., 238

test, **135**, 136, 141, 194, 335
P, 135
Bernoulli, 142, 143
Bose–Einstein distribution, 282
Cauchy condensation, 90
confidence interval of, 134
critical region of, **134**, 137
Fermi–Dirac distribution, 281–282
in statistics, 93, **134**, 164
integral, 223, **311**
Levin's, 231
lower semicomputable, **138**, 150
Martin-Löf, 135
martingale, 317
pseudo randomness, 49
ptime pseudorandom, 541
randomness, 318
recursive, 138
reference universal for uniform distribution, 140
sequential, **147**, 194, 541
sequential μ, 147
sequential Bernoulli, 162, 163
sequential for uniform distribution, 148
sequential Martin-Löf, 147
sequential martingale, 317
sequential ptime, 541
significance level of, 134
Solovay randomness, 233
statistical, 284
sum, 278
testing for randomness, 134
universal, **136**, 138, 194, 219, 220, 280, 283, 285, 334
universal Bernoulli, 142
universal for arbitrary computable P, 219
universal for uniform distribution, 139, 219, 281
universal integral, 223
universal martingale, **317**, 316–320
universal sequential, **149**, 194, 219, 223, 236, 256, 329
universal sequential Bernoulli, 162
universal sequential for the uniform measure, 151
universal sequential martingale, 317
universal sum, **278**, 278–284
universal uniform, 163
Thackeray, W.M., 319
theorem
s-m-n, **41**, 42
basic of r.e. sets, 42
binomial, 9
Blum speed-up, 550
Chinese remainder, 124
coding, 273, **273**, 277, 278, 287, 289, 301, 305–307, 334, 587, 588
coding, continuous version, 305
conditional coding, 275
conversion, **643**, 659
entropy uniqueness, 87
enumeration, **31**, 42
equality stochastic entropy and expected complexity, 187
Fermat's last, **179**, 228
Fine, 141
hierarchy, 47
incompleteness, 3, 34, **35**, 177, 195, 612, 714
incompressibility, 117
incomputability, 127
invariance, 96, 97, 99, 104, **105**, 192, 198, 202, 252
invariance for K, 193
invariance instance complexity, 571
invariance quantum Kolmogorov complexity, 703
invariance uniform complexity, 130
invariance, for $C^{t,s}$, 532
Körner–Csiszár–Marton, 654

Kamae, 125, **181**, 213
KC-characterization, 492
Liouville, 681
Matijasevich, 239, 240, 256
McMillan–Kraft, **88**, 205
Muchnik, 521, **654**, 659, 660, 717
Myhill–Nerode, 491
noiseless coding, **77**, 79, 80, 95, 199, 278, 280, 605
Occam's razor, **372**, 373, 378, 380, 381, 434
prime number, **17**, 552
recursion, 133
Savitch, 569
Schnorr, 221, **221**, 224, 225, 227–230, 233, 255
second recursion, 46
Slepian–Wolf, 654
supermartingale convergence, 321
symmetry of information (C-version), **190**, 192
symmetry of information (K-version), 247, **249**
time-bounded coding, **587**, 588
van Lambalgen, 231
theory
axiomatizable, **34**, 175–177, 186
consistent, 34
decidable, 34
sound, **34**, 175–177, 186
thermodynamics
first law, 674, 675
of computation, 629–641
second law, 674
statistical, 679
Thomas, J.A., 90, 94, 95, 324, 338, 366, 433, 439
Thurston, W., 483
Tikhomirov, V.M., 97
time–energy tradeoff, 658
Todt, G., 46
Toffoli, T., 715, 716
Torenvliet, L., 569
tournament, **452**, 452–454
ranking, 457
transitive, 452
Trevisan, L., 597
Tromp, J.T., 5, 227, 234, 255, 288, 397, 421, 425, 426, 434, 468, 483, 525, 526, 577, 641, 714, 716
Turan, G., 511
Turing
degree, 183
machine, *see* machine, Turing
quantum machine, *see* machine, quantum Turing
thesis, 24
Turing, A.M., 24, 33, 42, 92, 192
two-part description, 403
Tyszkiewicz, J., 296, 530

Ullman, J.D., 527
undecidable statement, **3**, 35, 175–177
uniform limit, 130
upper bounds
carry sequence, 444–445
combinatorics, 451–460
covering families, 457
routing in networks, 470–472
tournaments, 452
Uspensky, V.A., 97, 98, 125, 158, 194, 206, 221, 256, 290, 336, 337, 439, 600
USSR, research in former, 193
Ustinov, M.A., 427, 660

Valiant learning model, 370–382
Valiant, L.G., 94, 370, 382, 434, 459, 559, 561
Valiente, G., 665, 719
van Dam, W., 709, 722
van der Helm, P.A, 437
van der Waerden number, 460
van Melkebeek, D., 570, 597
Vapnik, V.N., 434
variation, 9
Varré, J.S., 717
Vazirani, U., 434, 530, 709
Vazirani, V., 530, 559

Vereshchagin, N.K., 125, 194, 256, 290, 421–430, 439, 519–521, 529, 626–628, 659, 660, 714, 715, 717
Vert, J.P., 718
Ville, J., 54, 58, 93, 335, 337
Vinodchandran, N.V, 586, 588, 589
Vitányi, P.M.B., 17, 92, 123, 195, 294, 334, 375, 380–382, 400, 421–431, 433–437, 439, 457, 458, 468, 469, 473, 484, 485, 489, 495, 501, 508, 509, 513, 519–521, 525–527, 529, 587, 588, 600, 626–628, 641, 658, 659, 670, 672, 673, 711, 714–720, 722
Vivant, E., 718
Vovk, V.G., 264, 329, 330, 397, 433, 436
Vyugin, M.V., 659, 660, 717
Vyugin, V.V., 193, 221, 332, 336, 423, 429, 439, 526

Wagner, K.W., 570
Wald, A., 53, 54, 93, 165
Wallace, C.S., 382, 435, 436
Wallman, H., 132
Wang, J., 598
Wang, W.G., 600
Wang, Y., 235, 596
Warmuth, M., 380, 434, 475
Watanabe, O., 206, 422, 549, 569, 570, 575–577, 597, 598, 600
Wechsung, G., 599
Wegman, M., 433
Wehner, S., 719
Wei, L., 667, 719
Weiss, B., 59
Weiss, S., 490
Whitehead, A.N., 177
Wiering, M, 599
Wigderson, A., 515, 516, 526, 570, 597
Wiles, A., 179
Williams, J.W.J., 476, 527
Willis, D.G., 327, 332
Wolf, R. de, 718
Wong, W., 719
Wood, D., 483
word
  finite, *see* string
  infinite, *see* sequence
WordNet, 670, 719

Yamanishi, K., 397, 436
Yang, E.H, 629, 715
Yang, Q., 490
Yao, A.C.C., 480, 500, 527, 529
Yee, C.N., 436
Yesha, Y., 500, 515, 525
Yianilos, P.N., 672
Yli-Harja, O., 719
Young, P., 575
Yu, B., 436
Yu, L., 215, 230, 231, 237
Yu, S., 495

Zambella, D., 232
Zator Company, 96, 333
Zeitman, R.Z., 45
Zhang, H., 672, 717
Zhang, L., 483, 525
Zhang, X., 719
Zhao, J., 599
Zhu, X., 720
Zimand, M., 530
Ziv, J., 715
Zurek, W.H., 625, 626, 633, 658, 659, 714, 716, 717, 720, 721
Zvonkin, A.K., 92, 132, 185, 191, 193–195, 255, 324, 331, 332, 334, 337, 546, 596
Zvozil, K., 722

TEXTS IN COMPUTER SCIENCE *(continued from p. ii)*

Kizza, Ethical and Social Issues in the Information Age, Second Edition

Kozen, Automata and Computability

Kozen, Theory of Computation

Li and Vitányi, An Introduction to Kolmogorov Complexity and Its Applications, Third Edition

Merritt and Stix, Migrating from Pascal to C++

Munakata, Fundamentals of the New Artificial Intelligence: Neural, Evolutionary, Fuzzy and More, Second Edition

Nerode and Shore, Logic for Applications, Second Edition

Pearce, Programming and Meta-Programming in Scheme

Revesz, Introduction to Constraint Databases

Schneider, On Concurrent Programming

Skiena and Revilla, Programming Challenges: The Programming Context Training Manual

Smith, A Recursive Introduction to the Theory of Computation

Socher-Ambrosius and Johann, Deduction Systems

Stirling, Modal and Temporal Properties of Processes

Zeigler, Objects and Systems