BIRKHÄUSER

Progress in Computer Science and Applied Logic

25

Mathematical Logic

Foundations for Information Science

Wei Li

Progress in Computer Science and Applied Logic

Volume 25

Editor:
John C. Cherniavsky, National Science Foundation

Associate Editors
Robert Constable, Cornell University
Jean Gallier, University of Pennsylvania
Richard Platek, Cornell University
Richard Statman, Carnegie-Mellon University

Mathematical Logic

Foundations for Information Science

Wei Li

Birkhäuser
Basel · Boston · Berlin

Author:

Wei Li
State Key Laboratory of Software Development Environment
Beihang University
37 Xueyuan Road, Haidian District
Beijing 100191
China
e-mail: liwei@nlsde.buaa.edu.cn

2000 Mathematics Subject Classification: 83C05, 83C35, 58J35, 58J45, 58J05, 53C80

Library of Congress Control Number: 2009940118

Bibliographic information published by Die Deutsche Bibliothek.
Die Deutsche Bibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data is available in the Internet at http://dnb.ddb.de

ISBN 978-3-7643-9976-4 Birkhäuser Verlag AG, Basel – Boston – Berlin

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in other ways, and storage in data banks. For any kind of use permission of the copyright owner must be obtained.

© 2010 Birkhäuser Verlag AG
Basel · Boston · Berlin
P.O. Box 133, CH-4010 Basel, Switzerland
Part of Springer Science+Business Media
Printed on acid-free paper produced from chlorine-free pulp. TCF∞
Printed in Germany

English version based on, 数理逻辑:基本原理与形式演算 (Mathematical Logic – Basic Principles and Formal Calculus), 978-7-03020096-9, Science Press, Beijing, China, 2007.

ISBN 978-3-7643-9976-4 e-ISBN 978-3-7643-9977-1

9 8 7 6 5 4 3 2 1 www.birkhauser.ch

Contents

Preface **ix**

Chapter 1 Syntax of First-Order Languages **1**
1.1 Symbols of first-order languages 4
1.2 Terms 6
1.3 Logical formulas 8
1.4 Free variables and substitutions 9
1.5 Gödel terms of formulas 13
1.6 Proof by structural induction 15

Chapter 2 Models of First-Order Languages **19**
2.1 Domains and interpretations 22
2.2 Assignments and models 24
2.3 Semantics of terms 24
2.4 Semantics of logical connective symbols 25
2.5 Semantics of formulas 27
2.6 Satisfiability and validity 30
2.7 Valid formulas with $\leftrightarrow$ 31
2.8 Hintikka set 33
2.9 Herbrand model 35
2.10 Herbrand model with variables 38
2.11 Substitution lemma 41
2.12 Theorem of isomorphism 42

Chapter 3 Formal Inference Systems **45**
3.1 **G** inference system 49
3.2 Inference trees, proof trees and provable sequents 52
3.3 Soundness of the **G** inference system 57
3.4 Compactness and consistency 61
3.5 Completeness of the **G** inference system 63
3.6 Some commonly used inference rules 66
3.7 Proof theory and model theory 68

Chapter 4 Computability & Representability **71**
4.1 Formal theory 72
4.2 Elementary arithmetic theory 74
4.3 *P*-kernel on $\mathbb{N}$ 76
4.4 Church-Turing thesis 80
4.5 Problem of representability 81

4.6 States of *P*-kernel 82
4.7 Operational calculus of *P*-kernel 84
4.8 Representations of statements 86
4.9 Representability theorem 95

Chapter 5 Gödel Theorems **97**
5.1 Self-referential proposition 98
5.2 Decidable sets 100
5.3 Fixed point equation in Π 104
5.4 Gödel's incompleteness theorem 107
5.5 Gödel's consistency theorem 109
5.6 Halting problem 112

Chapter 6 Sequences of Formal Theories **117**
6.1 Two examples 118
6.2 Sequences of formal theories 122
6.3 Proschemes 125
6.4 Resolvent sequences 128
6.5 Default expansion sequences 130
6.6 Forcing sequences 133
6.7 Discussions on proschemes 136

Chapter 7 Revision Calculus **139**
7.1 Necessary antecedents of formal consequences 140
7.2 New conjectures and new axioms 143
7.3 Refutation by facts and maximal contraction 144
7.4 **R**-calculus 146
7.5 Some examples 153
7.6 Special theory of relativity 155
7.7 Darwin's theory of evolution 156
7.8 Reachability of **R**-calculus 160
7.9 Soundness and completeness of **R**-calculus 163
7.10 Basic theorem of testing 164

Chapter 8 Version Sequences **169**
8.1 Versions and version sequences 171
8.2 The Proscheme `OPEN` 172
8.3 Convergence of the proscheme 176
8.4 Commutativity of the proscheme 178
8.5 Independence of the proscheme 180
8.6 Reliable proschemes 182

Chapter 9 Inductive Inference **187**
9.1 Ground terms, basic sentences, and basic instances 190
9.2 Inductive inference system **A** 192
9.3 Inductive versions and inductive process 197
9.4 The Proscheme `GUINA` 197
9.5 Convergence of the proscheme `GUINA` 204
9.6 Commutativity of the proscheme `GUINA` 206
9.7 Independence of the proscheme `GUINA` 207

Chapter 10 Workflows for Scientific Discovery **209**
10.1 Three language environments 209
10.2 Basic principles of the meta-language environment 213
10.3 Axiomatization 217
10.4 Formal methods 219
10.5 Workflow of scientific research 225

Appendix 1 Sets and Maps **229**

Appendix 2 Substitution Lemma and Its Proof **233**

Appendix 3 Proof of the Representability Theorem **237**
A3.1 Representation of the **while** statement in Π 237
A3.2 Representability of the *P*-procedure body 244

Bibliography **253**

Index **257**

Preface

Classical mathematical logic is considered to be an important component of the foundation of mathematics. It is the study of mathematical methods, especially the properties of axiom systems and the structure of proofs. The core of mathematical logic consists of defining the syntax of first-order languages, studying their models, formalizing logical inference and proving its soundness and completeness. It also covers the theory of computability and Gödel's incompleteness theorems. This process of abstraction started in the late 19th Century and was essentially completed by 1950.

In 1990, I began to give courses on mathematical logic. This teaching experience made me realize that, although deductive logic was well analyzed, the process of axiomatization had not been studied in depth. Several years later, I organized a series of seminars as an ensuing effort. The first five seminars covered classical mathematical logic and the rest were a preliminary outline of the formal theory of axiomatization.

As my understanding of mathematical logic became deeper, my desire to analyze and formalize the process of axiomatization became more intense. I also saw the influence of mathematical logic in information technology and scientific research. This inspired me to write a book for students living in the information society.

The computer was invented in the 1940's and high-level programming languages were defined and implemented soon afterwards. Computer science has developed rapidly since then. This exerted a profound influence on mathematical logic, because its concepts and theories were extensively applied. However, the development of computer science has, in turn, made new demands on mathematical logic, which have been the focus of my research and the motivation for this book. This motivation is guided by two considerations.

Firstly, mathematical logic was originally a general theory about axiom systems and proofs in mathematics, but now, its concepts and theories have been adopted by computer science and have played a principal guiding role in the design and implementation of both software and hardware.

For example, the method of structural induction was invented to define the grammar of first-order languages, but it is now used to define programming languages. This suggests that the study of mathematical logic can be applied to many areas of computer science.

Another example is given by Peano's theory of arithmetic. This is a formal theory in a first-order language, while the natural number system is a model of that theory. The distinction is essential in mathematical logic, because it is necessary in order to prove important theorems such as those of Gödel. However, many people outside this field find it hard to see the utility of making this distinction.

But in computer science, it is vital to differentiate between a high-level programming language and compiled executable codes. The difference between programs and their compiled executables is precisely the same as that made between first-order lan-

guages and their models, so the theorems of mathematical logic can be directly applied to study the properties and correctness of software systems.

These two examples show how mathematical logic is necessary to computer science, but we have also found the concepts of computer science helpful in understanding logic. For instance, students often find the process of Gödel coding difficult to grasp. To help them, we can make an analogy with computer science. In this, formulas are viewed as variable names in a programming language; the Gödel coding corresponds to the mechanism of assigning a pointer and the Gödel number corresponds to the address of the pointer, whose content is the Gödel term. This analogy helps students to understand and use these difficult concepts.

So I aspired to write a book that not only studies mathematical logic but also enlightens those who are living in the information society and are doing scientific research. This is why this book tries to illustrate the concepts, theories and methods of mathematical logic with the practical use of computers, programming languages and software, so that we can see the close relationship between mathematical logic and computer science.

The second motivation for this book is that research in computer science and technology during the last 60 years has developed many valuable methods and theories that are not covered by classical mathematical logic. I have long cherished a hope that mathematical logic could be enriched and extended to include these concepts. This aim has guided my research into investigating the following basic problems:

1. Software version

A software system is written in a programming language and its specification may be described by the formal theory of a first-order language. However, its implementation rarely completely satisfies the requirements of its designers or users. It can only be implemented through frequent exchange and close collaboration between the developers. This leads to a process of evolution through a series of versions.

It is only by distinguishing the different versions of the software that the exchange and collaboration between developers can be managed. Therefore, mathematical logic needs to incorporate the concepts of a version of a formal theory and of a version sequence, so that the evolution of formal theories can be described and studied.

2. Testing and debugging

Testing is crucial in software development. Software can only be released after it has passed rigorous tests. Many tools have been developed to assist this process. In spite of this, software testing still requires much manpower and it is a skilled craft that depends on the proficiency and experience of the testing personnel.

On the whole, software testing has two parts: designing test cases and finding and correcting software errors. Both of these require logical analysis, but this is different from the logical inference used in mathematical proof. Since mathematical proof is formally defined, we can perform it with the aid of interactive software systems. In the same way, we would like to build software tools to locate errors and to revise existing versions. If the concepts of error correction can be expressed in mathematical logic, then the goal of 'mechanization' could be realized. This research should play a guiding role in improving the efficiency of software testing.

3. The methodology of software development

The quality of software products is determined by the methodology of their development. Generally speaking, this methodology mainly consists of rules and workflows, which are managed by software tools. We would like to study this methodology as an object in mathematical logic. In this way, we could define a programming-like language to formally describe different methodologies of software development and could study their properties and prove their reliability.

4. Meta-language environment

First-order languages and their models are defined and specified in the meta-language environment and, in addition, many important theorems are proved in this environment. This will inevitably impose requirements and restrictions on the meta-language environment, so mathematical logic must specify clearly the principles that the environment must obey.

In general, any theory of mathematics or natural science is formed by a kind of evolutionary process, which is manifested as a series of different versions at different stages of development. Scientific theories are developed over a long period of time because only a limited number of experts are involved. The scale of their principles and theorems is far smaller than that of software systems and the time needed for their development is much longer. Therefore, the different versions of the theory are not so obvious as in software development.

For this reason, classical mathematical logic only takes a particular version of an axiom system as its object of study and deduces the logical consequences within that version. However, problems such as managing versions and version sequences, revision of theories, selecting methodologies of scientific research and consideration of the meta-language environment are important in the process of development of all theories. So these are all problems which mathematical logic should now define and formally analyze.

The book consists of two parts, each containing five chapters. The first part presents the core ideas of classical mathematical logic, while the second part deals with the author's work on formalizing axiomatization.

The second part includes a definition of versions of a formal theory, version sequences and their limits. It formalizes the revision of formal theories, defines the concept of proscheme, and uses it to describe a methodology for the evolution of formal theories. It goes on to study inductive inference and prescribes the principles of a meta-language environment. These are an extension and development of classical mathematical logic.

This book adopts the rigorous standards of classical mathematical logic: All concepts are strictly defined and illustrated with examples; all theorems are proved and details of proofs are provided if at all possible; all quoted conclusions and methods are referred to their original authors and sources. This book is intended to be a course book for postgraduate students of information science, but the first five chapters may be used as a textbook for undergraduate students.

Although several major revisions have been made of the draft of this book in the past few years, I do not claim that the present text is free of omissions or even errors. I would sincerely appreciate any criticisms or suggestions.

Many colleagues and students of mine read my manuscripts and contributed to the preparation of this book. Their comments and suggestions led to significant improvements in the content and presentation of the book. In particular, I would like to mention Jie Luo, Shengming Ma, Dongming Wang, and Yuping Zhang, who helped me considerably in preparing the English version, typesetting, proofreading and giving many useful suggestions. Jie Luo and Shengming Ma supplied a detailed proof of the theorem of representability in Appendix 3. My sincere thanks go to all of them for their generous support, help, and contribution. My heartfelt thanks also go to Bill Palmer for his passionate and professional efforts in language editing.

My wife Hua Meng was the first to advise me to distill my research and understanding of mathematical logic into a book. She and my daughter Xiaogeng Li looked on my writing as one of the most important events in my family. It is hard to tell how long the publication of this book would have been delayed without their loving care and constant support and encouragement. I dedicate this book to them with gratitude.

Wei Li

Beihang University, Beijing

September 2009

Chapter 1

Syntax of First-Order Languages

Programming languages such as BASIC, Pascal, and C are formal languages used for writing computer programs. A program usually implements an algorithm which describes the computational solution of a specific issue. This chapter introduces a different kind of formal language, known as a *first-order language*.

A first-order language is used to describe the properties and relationships between the objects in a specific domain. Usually, these domains are mathematical or scientific in nature. For example, the axioms, theorems, and corollaries in plane geometry, the properties of natural numbers, and the laws and principles in physics are objects that can be described by first-order languages.

We usually start describing a domain by defining the properties of its objects. Each property is described by one or more propositions.

For example, the following propositions describe aspects of number theory:

"1 is a natural number."
"No two different natural numbers have the same successor."
"If $a > 1$ and a cannot be divided by 2, then a is an odd number."

And the following describe knowledge of physics:

"A photon is a rigid body."
"The velocity of light does not depend on the velocity of the body emitting the light."
"A rigid body will continue in a state of rest or of uniform motion in a straight line unless it is acted upon by a force."

Lastly, the following describe relationships between people:

"Confucius is a human."
"Zisi is a descendant of Confucius."
"If A is a descendant of B and B is a descendant of C, then A is a descendant of C."

It should be pointed out that assertions, statements or even specifications are used instead of propositions in some other books on mathematical logic. For the sake of simplicity and uniformity, we use propositions in this book to denote the properties of the objects in a domain.

Our knowledge of a domain is composed of propositions which describe the properties of and relationships between objects. The kernel of these propositions forms an axiom system such as the axioms of Euclidean geometry or the set of laws in classical mechanics. Specifications of functional requirements for software systems are also axiom systems that describe domain knowledge.

First-order languages are specifically useful to describe axiom systems because they allow us to reason from the axioms with a symbolic calculus, which can be implemented as computer software.

Computer programs use commands or statements to specify computations. The purpose of computation is to solve a problem algorithmically. In contrast, axiom systems use propositions to describe the properties of and relationships between objects in a domain. Logical inference rules are used to deduce the logical consequences of axioms in a mechanical way. They explore the logical structure of a domain, finding all propositions that are provable from the axioms.

What do we mean when we say that a programming language is a formal language? We mean that it is constructed from an alphabet which is a set of symbols. These symbols are used to define several kinds of syntactic objects such as program declarations and statements, and each syntactic object is strictly defined by a specific grammar, which is a set of syntactic rules. Only programs written in strict accordance with the grammar can convert algorithms into mechanical operations executable on computers.

In the same way, a first-order language is also a formal language. It is based upon a set of symbols and is composed of two kinds of syntactic objects. Each syntactic object has a specific syntactic structure and is defined by a set of rules. If an axiom system is defined in strict accordance with the syntactic rules of first-order languages, we can convert logical reasoning about a domain into symbolic calculus.

The difference between first-order languages and programming languages lies in the fact that the description of the knowledge of each specific domain requires a specific first-order language, while any computable problem can be solved by programs written in any programming language.

Let us discuss what sets of *symbols* and syntactic objects a first-order language should contain. The symbols used by each first-order language should be of two types. One of them is related to specific domain knowledge and these are special symbols used by this language and are called *domain specific symbols*. The other consists of symbols common to the description of every domain, which are called *logical symbols*.

Symbols related to specific domain knowledge may be further divided into two types. One type is used to describe constants and functions and consists of *constant symbols* and *function symbols*. The other type is used to describe relationships between concepts and the symbols are called *predicate symbols*. The following are some examples of constant symbols, function symbols, and predicate symbols:

(1) Constant symbols: 0, π, and e are constants in mathematics. The acceleration of gravity (g), universal gravitational constant (G), and the velocity of light (c) are constants in physics. Confucius and Zisi (the grandson of Confucius) are both constants describing a human relationship. Every constant of a domain is described by a specific constant symbol in a first-order language for the domain.

(2) Function symbols: The successor σ of x defined by $\sigma(x) = x + 1$ is a unary function, and addition and multiplication are binary functions in number theory. $\sin x$, $\cos x$, $\ln x$, $\exp x$ are functions used in physics. Each function of a domain is described by a specific function symbol in a first-order language for the domain.

(3) Predicate symbols: "is prime," "is even," and "is odd" are some of the basic properties of natural numbers, "=", "$<$" are basic logical relations in number theory, "rigid body," "velocity," "force," etc. are basic concepts in physics, and "descendant" is a basic relation between humans. In academic research, we use natural(x) to denote "x is a natural number," even(y) to denote "y is an even number," rigid(z) to denote "z is a rigid body," $P(x,y)$ to denote "x is the descendant of y," and so on. In general, a basic property in a domain is described by a specific predicate symbol in first-order languages for the domain.

There are three other kinds of symbols, namely, *variables*, *logical connectives*, and *quantifiers*, which are needed to specify logical statements about a domain. They are called *logical symbols* in first-order languages.

Symbols of the first kind are the variables occurring in functions and predicates, such as x, y, and z in the previous examples. Conceptually, they are the same as the variables defined in programs. In first-order languages they are also called *variable symbols*.

Symbols of the second kind denote logical connectives occurring in propositions. Each proposition in a domain is composed of basic statements combined by logical connectives. There are five commonly used logical connectives:

"negation of ...," "... and ...," "... or ...,"
"if ... then ...," "... if and only if ..."

For example, in the proposition "if $1 < a$ and a cannot be divided by 2, then a is an odd number," "cannot be divided by 2" is the negation of "can be divided by 2," while "if $1 < a$ and a cannot be divided by 2" is connected by using "and." Finally, the proposition takes the form "if ... then ...." In fact, most other logical connectives may be expressed as combinations of these five logical connectives.

As in programming languages, special symbols are introduced in first-order languages to denote logical connectives:

Logical connective	negation	and	or	if ... then ...	if and only if
Special symbol	$\neg$	$\wedge$	$\vee$	$\rightarrow$	$\leftrightarrow$

By using the above symbols, the proposition "if Kongrong is a descendant of Zisi and Zisi is a descendant of Confucius, then Kongrong is a descendant of Confucius" can be described as

$$(P(\text{Kongrong, Zisi}) \wedge P(\text{Zisi, Confucius})) \rightarrow P(\text{Kongrong, Confucius}).$$

A third kind of symbol is used to describe generality. These symbols are called quantifiers and are described in first-order languages by the following symbols:

Quantifier	for all x ...	there exists an x ...
Special symbol	$\forall x$ ...	$\exists x$ ...

They deal with the instantiation or universality of a concept. In other words whether a property holds for some object or all objects in a domain. Thus, the proposition "for all x,

y, z, if x is a descendant of y and y is a descendant of z, then x is a descendant of z" can be expressed as

$$\forall x \forall y \forall z((P(x,y) \wedge P(y,z)) \rightarrow P(x,z)),$$

and read as: for all x, y, z, if $P(x,y)$ holds and $P(y,z)$ holds, then $P(x,z)$ holds. In the above formula, parentheses are used to indicate the priority of logical connectives. If $(P(x,y) \wedge P(y,z))$ does not have outer parentheses, then it is not clear whether $P(x,y) \wedge P(y,z) \rightarrow P(x,z)$ should be read as "$P(x,y)$ holds and if $P(y,z)$ holds, then $P(x,z)$ holds," or as "if $P(x,y)$ holds and $P(y,z)$ holds, then $P(x,z)$ holds." The outer parentheses of $(P(x,y) \wedge P(y,z))$ indicate that we mean the latter. Therefore, parentheses are indispensable in first-order languages.

A programming language may contain several kinds of syntactic objects, whereas a first-order language has only two kinds of syntactic objects, i.e., "terms" and "logical formulas." Terms are used to describe constants, variables, and functions, while logical formulas are used to describe propositions. We shall see later that the "terms" and "logical formulas" of first-order languages are defined by different syntactic rules.

In summary, first-order languages are formal languages used to describe propositions about knowledge domains. The aim of introducing first-order languages is to convert logical reasoning into symbolic calculus. We have explained briefly the symbols and syntactic ingredients that a first-order language should have. The purpose of this chapter is to give formal definitions of these concepts. This chapter also discusses a specific method of defining first-order languages, which is called *definition by structural induction* and the powerful mathematical proof techniques that this method induces.

It should be noted that historically, first-order languages appeared earlier than programming languages. It was during the theoretical study of first-order languages that computability was thoroughly studied and formally defined, followed by the invention of computers. In order for computer to be programmed easily, scientists began to design high-level programming languages using the theory of formal languages, which had matured through the study of first-order languages. As a result of the popularization of computers, programming languages began to be taught at high schools. The fundamental ideas and methods of first-order languages have been widely accepted, if mostly unknowingly, through the use of programming languages. Therefore, first-order languages are presented in this book in comparison with daily-used programming languages so that they may become easier to understand and master.

1.1 Symbols of first-order languages

We have already pointed out that first-order languages are formal languages describing various knowledge domains, but in order to discuss their properties, we will introduce them at an abstract level. A first-order language is defined by the following symbol sets.

Definition 1.1 (First-order languages). The set of symbols of each first-order language is composed of two kinds of symbol sets. One is identified as logical symbol sets, whereas

the other is named non-logical symbol sets or symbol sets for the knowledge domains. The logical symbol sets include the following.

V: The set of variable symbols. It consists of countable (possibly empty) variable symbols: $x_1,\ x_2,\ldots,x_n,\ldots$.

C: The set of logical connective symbols. It is composed of the symbols of logical connectives $\neg,\ \wedge,\ \vee,\ \rightarrow$ and $\leftrightarrow$ that read as "not", "and", "or", "if ... then ...", and "if and only if" respectively.

Q: The set of quantifier symbols. It includes $\forall$ and $\exists$ that read as "for all ..." and "there exists a(n) ..." respectively.

E: The set containing the *equality symbol* $\doteq$.

P : The set of parenthesis symbols. It encompasses "(" and ")" that read as "the left parenthesis" and "the right parenthesis".

In particular, each first-order language has three kinds of non-logical symbol sets of its own.

$\mathscr{L}_c$: The set of constant symbols. It consists of countable (including zero) constant symbols: $c_1,\ c_2,\ldots$.

$\mathscr{L}_f$: The set of function symbols. It is composed of countable (including zero) function symbols: $f_1,\ f_2,\ldots$. We use $fx_1\cdots x_m$ to denote an m-ary function symbol, in which $m \geqslant 1$ is the number of variable symbols as well as the number of arguments of the function.

$\mathscr{L}_P$: The set of predicate symbols. It consists of countable (including zero) predicate symbols that are represented by $P_1,P_2,\ldots$. We use $Px_1\cdots x_m$ to denote an m-ary predicate symbol, in which $m \geqslant 1$ is the number of variable symbols as well as the number of arguments of the predicate.

All first-order languages have the same sets of logical symbols, whereas different first-order languages have different non-logical symbol sets. We should point out that $\doteq$ is also a predicate symbol. Hereafter we shall use $\mathscr{L}$ to represent a first-order language. Since all first-order languages have the same sets of logical symbols, $\mathscr{L}$ essentially represents the sets of non-logical symbols of the first-order language.

In what follows, we shall give an example of a first-order language that describes elementary arithmetic. We will use this frequently in later chapters of the book.

Example 1.1 (Elementary arithmetic language $\mathscr{A}$). The language of elementary arithmetic is a first-order language that will be denoted by $\mathscr{A}$ henceforth. Its constant symbol set, function symbol set, and predicate symbol set are $\{0\}$, $\{S,\ +,\ \cdot\}$ and $\{<\}$ respectively.

The purpose of introducing S is to represent the successor function or "plus 1" function in arithmetic. The binary function symbols $+$ and $\cdot$ stand for the addition and multiplication in arithmetic respectively. The predicate symbol $<$ denotes the "less than" relation between two natural numbers.

Although first-order languages and programming languages have a similar foundation, the way symbols are used in each is different. Firstly, a programming language is more general purpose; any algorithm can be expressed, and new constants and functions can be created for this purpose. A first-order language, on the other hand, is defined to describe one specific domain of knowledge and the constants, functions and predicates are determined by that domain.

Secondly, programming languages allow only a finite number of identifier symbols while first-order languages permit a countably infinite number of separate symbols. As we shall see, knowledge about some mathematical domains can only be captured using an infinite number of symbols.

For the convenience of description and usage, we prescribe that the symbols used in $V, \mathscr{L}_c, \mathscr{L}_f$ and $\mathscr{L}_P$ are different from one another in this book. We also use the lowercase letters x, y, z, ... to denote variable symbols, f, g, h, ... to denote function symbols and the uppercase letters P, Q, R, ... to denote predicate symbols, and thus conform to the conventions of mathematics and knowledge domains. In addition, we will simply refer to the constant symbols and predicate symbols as constants and predicates hereafter if no misunderstanding of the context is incurred.

1.2 Terms

Terms are one of the two kinds of syntactic objects of first-order languages. The terms of first-order languages are defined by the same method as that of arithmetic expressions of programming languages, except that the former are more general and allow countably infinite constant, variable, and function symbols.

Definition 1.2 (Terms). The *terms* of a first-order language $\mathscr{L}$ are defined inductively by the following three rules.

T_1: Each constant symbol is a term.

T_2: Each variable symbol is a term.

T_3: If $t_1, \ldots, t_n$ are terms and f is an n-ary function symbol, then $ft_1 \cdots t_n$ is a term.

The rules in Definition 1.2 are named as *T-rules*. Henceforth we shall use $\mathscr{L}_T$ to denote the set of terms of the first-order language $\mathscr{L}$.

Definition 1.2 is a structural inductive definition that can also be represented in the following form:

$$t ::= c \mid x \mid ft_1 \cdots t_n.$$

Here $\mid$ denotes "or" and $::=$ denotes "is inductively defined as". The above form is called the *Backus normal form* [Backus, 1959].

Example 1.2 (Terms of $\mathscr{A}$). The symbol strings

$$S0,\ Sx_1,\ +S0SSx,\ \cdot x_1 + Sx_1x_2,\ SS <$$

are all terms of $\mathscr{A}$ except $SS<$.

For any finite string composed of symbols of $\mathscr{A}$, we can determine if it is a term of $\mathscr{A}$ by invoking the T-rules in finite steps. In what follows, let us prove that $+S0SSx$ is a term.

(1) 0 is a term. (By T_1.)

(2) x is a term. (By T_2.)

(3) $S0$ is a term. (By T_3 since S is a unary function symbol and 0 is a term according to (1).)

(4) Sx is a term. (By T_3 since S is a unary function symbol and the variable x is a term according to (2).)

(5) SSx is a term. (By T_3 since S is a unary function symbol and Sx is a term according to (4).)

(6) $+S0SSx$ is a term. (By T_3 since $+$ is a binary function symbol and both $S0$ and SSx are terms according to (3) and (5).)

The intention of introducing terms is to describe the constants, variables and functions in a knowledge domain. Definition 1.2 says that each term is just a symbol string whose symbols are from the symbol sets V, $\mathscr{L}_c$ and $\mathscr{L}_f$, and whose constructions are in strict accordance with the T-rules. Definition 1.2 does not concern the meaning of terms at all. When discussing the semantics of first-order languages in Chapter 2, we shall see that terms are interpreted as constants, variables or functions of a domain.

In Example 1.2, the function symbol S can be used repeatedly. This is a characteristic of structural inductive definitions. Hereafter we write

$$S^0 0 \text{ for } 0 \text{ and } S^{n+1}0 \text{ for } S(S^n 0), \text{ and thus } S^n 0 \text{ for } \underbrace{SS\cdots S}_{n}0.$$

$S^n 0$ is only an abbreviation whose superscript n stands for "making successor operations n times."

We should also point out that, except for the lack of parentheses, the representations of terms in first-order languages are basically the same as those of constants, variables and functions in programming languages and textbooks in mathematics. The representations for the terms $S0$ and Sx_1 are called *prefix representations*. They are usually written as $S(0)$ and $S(x_1)$ in programming languages and textbooks in mathematics. The conventional representations of $+Sx_1x_2$ and $\cdot\, x_1 + Sx_1x_2$ are $S(x_1)+x_2$ and $x_1 \cdot (S(x_1)+x_2)$ respectively. In this book, we shall use the conventional representations of these frequently used functions for the convenience of reading and understanding, provided that no misunderstandings of the context are incurred. Strictly speaking, these are no longer the terms of $\mathscr{A}$ specified by Definition 1.2 but their aliases.

1.3 Logical formulas

The other kind of syntactic objects in first-order languages are logical formulas. They are the 'first class' syntactic objects of first-order languages and are defined by structural induction.

Definition 1.3 (Logical formulas). The *logical formulas* of a first-order language $\mathscr{L}$, or called formulas for short, are represented by uppercase letters $A,\ B,\ \ldots$ and are defined inductively by the following five F-rules.

F_1 : If t_1 and t_2 are terms, then $t_1 \doteq t_2$ is a formula.

F_2 : If $t_1,\ldots,t_n$ are n terms and R is an n-ary predicate, then

$$Rt_1\cdots t_n$$

is a formula.

F_3 : If A is a formula, then $(\neg A)$ is a formula.

F_4 : If A and B are formulas, then

$$(A\wedge B),\ (A\vee B),\ (A\to B),\ (A\leftrightarrow B)$$

are all formulas.

F_5 : If A is a formula and x is a variable, then

$$\forall xA \ \text{ and } \ \exists xA$$

are also formulas. In this case, x is called a *bound variable*.

The formulas defined by the rules F_1 and F_2 are identified as *atomic formulas*, whereas the formulas defined by the rules F_3, F_4 and F_5 are called *composite formulas*.

The formula $(\neg A)$ reads as the "negation of formula A" or "not A". $(A\wedge B)$, $(A\vee B)$, $(A\to B)$ and $(A\leftrightarrow B)$ read as the "conjunction of formulas A and B," "disjunction of formulas A and B," "A implies B," and "A is equivalent to B" respectively. $\forall xA$ and $\exists xA$ are identified as quantified formulas with A being the body of the formula. $\forall xA$ and $\exists xA$ read as "for all x, A" and "there exists an x such that A" respectively. The redundant parentheses in the formula can be omitted without changing the meaning of a formula. Henceforth we use the notation $\mathscr{L}_F$ to denote the set of the formulas in a first-order language $\mathscr{L}$.

The Backus normal form defined by the above structural induction is:

$$A \ ::= \ t_1 \doteq t_2 \mid Rt_1\cdots t_n \mid \neg A \mid A\wedge B \mid A\vee B \mid A\to B \mid A\leftrightarrow B \mid \forall xA \mid \exists xA.$$

Example 1.3 (The formulas of $\mathscr{A}$). According to Definition 1.3, we can determine if the symbol strings

$$\forall x\neg(Sx\doteq 0) \quad \text{and} \quad \forall x\forall y(<xy\to(\exists z(y\doteq +xz)))$$

are formulas of $\mathscr{A}$. In what follows, we prove that the symbol string $\forall x\neg(Sx\doteq 0)$ is a formula of $\mathscr{A}$.

(1) $Sx \doteq 0$ is a formula. (By F_1 since both Sx and 0 are terms according to Example 1.2.)

(2) $\neg(Sx \doteq 0)$ is a formula. (By F_3 and (1).)

(3) $\forall x\neg(Sx \doteq 0)$ is a formula. (By F_5 and (2).)

Similarly, we can prove that the symbol string $\forall x\forall y(<xy \rightarrow (\exists z(y \doteq +xz)))$ is also a formula.

According to Definition 1.3, each logical formula is a finite symbol string constructed in strict accordance with the F-rules. Definition 1.3 tells how a formula is defined syntactically, but it does not concern its meaning. Logical formulas describe propositions about the domain of knowledge. For example, the formula $\forall x\neg(Sx \doteq 0)$ denotes the proposition "each natural number is greater than or equal to 0" and the formula $\forall x\forall y(<xy \rightarrow (\exists z(y \doteq +xz)))$ expresses the proposition "if $x < y$, then there must exist a natural number z such that $y = x+z$." In the next chapter, we will introduce a method of interpreting each symbolic formula by a semantic proposition about a domain.

An advantage of introducing first-order languages is that by symbolizing the constants, functions, equations, predicates, logical connectives and quantifiers in propositions of a domain, the logical structures implicit in propositions can be made explicit. In this way, the logical reasoning about the propositions can be converted into symbolic calculus.

1.4 Free variables and substitutions

Local variables are allowable in programming languages with each variable having a specific scope within which the variable is bound and available. In addition, programmers are allowed to use formal parameters in the declarations of procedures and functions. The formal parameters are a type of variable which are substituted by real parameters when the procedures and functions are called. The ideas of local variables, formal parameters, real parameters and substitutions in programming languages coincide with those of bound variables, free variables and substitutions in first-order languages. In first-order languages, variables may be bound by quantifier symbols. Let us look at an example first.

Example 1.4. Suppose that $x,\ y,\ z$ are three different variables in the formula

$$A:\ \exists x((P(x,y) \wedge \forall y R(x,y)) \rightarrow Q(x,z)),$$

where P, R, Q are three binary predicates. The variable x in P, R, Q is bound by the outmost quantifier symbol $\exists$ with its scope being $((P(x,y) \wedge \forall y R(x,y)) \rightarrow Q(x,z))$. The variable y in R is bound by the quantifier $\forall$ with its scope being $R(x,y)$. Nonetheless, y in P and z in Q are not bound by any quantifiers in the formula and they occur free in the formula A; they are the free variables of A.

Definition 1.4 (Free variables of terms). Suppose that t is a term of $\mathscr{L}$ with $FV(t)$ being the set of free variables of t. According to the syntactic structure of terms, $FV(t)$ is

defined in a structurally inductive way as follows.

$$\begin{aligned} FV(x) &= \{x\}, && x \text{ is a variable.} \\ FV(c) &= \varnothing, && c \text{ is a constant symbol.} \\ FV(ft_1\cdots t_n) &= FV(t_1)\cup\cdots\cup FV(t_n). \end{aligned}$$

If $x \in FV(t)$, then we identify x as a *free variable* of t or say that x occurs free in t. If $FV(t) = \varnothing$, then we call t a *ground term* (or *closed term*).

Definition 1.5 (Free variables of formulas). Suppose that A is a formula of $\mathscr{L}$, $FV(A)$ being the set of free variables of A. We define the set of free variables of A, $FV(A)$, inductively as follows.

(1) $FV(t_1 \doteq t_2) = FV(t_1)\cup FV(t_2)$.

(2) $FV(Pt_1\cdots t_n) = FV(t_1)\cup\cdots\cup FV(t_n)$.

(3) $FV(\neg A) = FV(A)$.

(4) $FV(A * B) = FV(A)\cup FV(B)$, where $*$ stands for any of $\wedge, \vee, \rightarrow, \leftrightarrow$.

(5) $FV(\forall xA) = FV(A) - \{x\}$.

(6) $FV(\exists xA) = FV(A) - \{x\}$.

If $x \in FV(A)$, then we identify x as a *free variable* of the formula A or say that x occurs free in A. If $FV(A) = \varnothing$, then we call A a *sentence*, which is a formula encompassing no free variables.

Example 1.5. According to Definitions 1.4 and 1.5, we can determine the free variables in $\exists x((P(x,y)\wedge\forall yR(x,y))\rightarrow Q(x,z))$ as follows.

$$\begin{aligned} & FV(\exists x((P(x,y)\wedge\forall yR(x,y))\rightarrow Q(x,z))) \\ &= FV((P(x,y)\wedge\forall yR(x,y))\rightarrow Q(x,z)) - \{x\} \\ &= (FV(P(x,y)\wedge\forall yR(x,y))\cup FV(Q(x,z))) - \{x\} \\ &= (FV(P(x,y))\cup FV(\forall yR(x,y))\cup\{x,z\}) - \{x\} \\ &= (\{x,y\}\cup(FV(R(x,y)) - \{y\})\cup\{x,z\}) - \{x\} \\ &= (\{x,y\}\cup(\{x,y\} - \{y\})\cup\{x,z\}) - \{x\} \\ &= (\{x,y\}\cup\{x\}\cup\{x,z\}) - \{x\} \\ &= (\{x,y,z\}) - \{x\} \\ &= \{y,z\}. \end{aligned}$$

We should point out that y here is the variable y in $P(x,y)$, not the one in $R(x,y)$.

In programming languages, a formal parameter used in the declaration of a function can be substituted by a real parameter. In the same way, a free variable in a term or formula can be substituted by a term, creating a new instance of that expression. The following definition makes this procedure precise.

Definition 1.6 (Substitution of terms). Let s and t be terms. Denote by $s[t/x]$ the term obtained from s by substituting the term t for the free variable x of s. According to the structure of terms, $s[t/x]$ is inductively defined as follows.

$$\begin{aligned} y[t/x] &= y, && \text{if } y \neq x.\\ y[t/x] &= t, && \text{if } y = x.\\ c[t/x] &= c, && c \text{ is a constant symbol.}\\ ft_1\cdots t_n[t/x] &= ft_1[t/x]\cdots t_n[t/x]. \end{aligned}$$

Note that the equal sign $=$ in the above definition refers to the equality of elements in a set, which is different from $\doteq$ in first-order languages. The equality symbol $\doteq$ is a specific predicate symbol of first-order languages.

Definition 1.7 (Substitution of formulas). Let A be a formula containing a free variable x. $A[t/x]$ stands for the formula obtained from A by substituting the term t for the free variable x of A. It is abbreviated sometimes to $A[t]$. According to the syntactic structure of formulas, $A[t/x]$ is inductively defined as follows.

(1) $((t_1 \doteq t_2)[t/x]) = (t_1[t/x] \doteq t_2[t/x])$.

(2) $Rt_1\cdots t_n[t/x] = Rt_1[t/x]\cdots t_n[t/x]$.

(3) $(\neg A)[t/x] = \neg(A[t/x])$.

(4) $(A * B)[t/x] = A[t/x] * B[t/x]$, where $*$ stands for any of $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$.

(5) $(\forall xA)[t/x] = \forall xA$.

(6) $(\exists xA)[t/x] = \exists xA$.

(7) $(\forall yA)[t/x] = \forall yA[t/x]$, if $y \notin FV(t)$.

(8) $(\exists yA)[t/x] = \exists yA[t/x]$, if $y \notin FV(t)$.

(9) $(\forall yA)[t/x] = \forall zA[z/y][t/x]$, if $y \in FV(t)$, $z \notin FV(t)$, z does not occur in A.

(10) $(\exists yA)[t/x] = \exists zA[z/y][t/x]$, if $y \in FV(t)$, $z \notin FV(t)$, z does not occur in A.

In rules (9) and (10), the conditions $z \notin FV(t)$ and z not occurring in A indicate that the variable z is a new variable with respect to t and A, that is, z is neither a free variable of t nor a free variable or bound variable of A.

Example 1.6 (Substitution). Let $t = fc$ with f being a unary function symbol and c being a constant symbol. We substitute t for the free variable y of the formula in Example 1.5 as follows.

$$\begin{aligned} &(\exists x(P(x,y) \wedge \forall yR(x,y)) \rightarrow Q(x,z)))[fc/y]\\ =\ &\exists x(((P(x,y) \wedge \forall yR(x,y)) \rightarrow Q(x,z))[fc/y])\\ =\ &\exists x((P(x,y) \wedge \forall yR(x,y))[fc/y] \rightarrow Q(x,z)[fc/y])\\ =\ &\exists x((P(x,y)[fc/y] \wedge (\forall yR(x,y))[fc/y] \rightarrow Q(x,z))\\ =\ &\exists x((P(x,fc) \wedge \forall yR(x,y)) \rightarrow Q(x,z)). \end{aligned}$$

Definition 1.7 provides three groups of substitution rules for quantified formulas. The first group consists of rules (5) and (6), which prescribe that we can only substitute for the free variables in a quantified formula. The second group is composed of rules (7) and (8), which indicate that if the bound variable of a quantified formula is not a free variable of the term t, then the substitution in the quantified formula amounts to a substitution in its body. The third group consists of rules (9) and (10), whose usage is demonstrated by the following example.

Example 1.7. Suppose that $A = \exists y(y < x)$ and let $t = y$. Consider a substitution $A[t/x]$. Since $x \neq y$, if we invoke the second group of rules, then we shall have

$$(\exists y(y < x))[y/x] = \exists y(y < y),$$

which does not coincide with our experience. In fact, if we interpret A as

"for any integer x, there exists a y such that $y < x$ holds",

then the proposition is true for integers. We certainly hope that the proposition is still true after a substitution for x. Nonetheless after the substitution the proposition becomes

"there exists an integer y such that $y < y$ holds",

which is false. The problem lies in the fact that y substituted for x is a bound variable of A. Generally speaking, if the free variables of t are not the bound variables of A, which is the condition $y \notin FV(t)$ for the second group of rules, then we make the substitution according to the second group of rules. If a free variable of t is by any chance also a bound variable of A, then the condition $y \in FV(t)$ for the third group of rules holds for this variable.

In this case, if we still make a substitution according to the second group of rules, then we shall make the mistake as described in the above example. The solution is to introduce a new variable z that is neither a free variable of t nor a free variable or a bound variable of A. When making the substitution, we first substitute z for the bound variable y of the quantifier such that the free variable of t is no longer the bound variable of A. Then we make the substitution $[t/x]$ according to the second group of rules and the mistake is avoided. This is the motivation for rules (9) and (10).

According to rule (10), the correct solution for the above example is

$$\begin{aligned}(\exists y(y < x))[y/x] &= (\exists z(y < x)[z/y])[y/x] \\ &= \exists z(z < x)[y/x] \\ &= \exists z(z < y).\end{aligned}$$

The result of the substitution can be interpreted as

"for any integer y, there exists an integer z such that $z < y$ holds,"

which lives up to our expectation.

In summary, if A is the quantified formula $\forall yB$ or $\exists yB$, then there are two groups of rules for making the substitution $A[t/x]$. If $y \notin FV(t)$, then we say that t is *free* for A with respect to y and we use the rules (7) and (8). If $y \in FV(t)$, then we say that t is *bound* by A with respect to y and we can only use rules (9) and (10).

1.5 Gödel terms of formulas

Although the terms and formulas of first-order languages are two different categories of syntactic objects, it is sometimes possible to convert one into the other. In this section, we show how this can be done in the language of elementary arithmetic. This method is called *Gödel coding* [Shoenfield, 1967]. The basic idea is to first code every formula A by a natural number $\&A$, called the *Gödel number* of A. Then the natural number $\&A$ is made to correspond to a term $S^{\&A}0$, called the *Gödel term* of A. The integration of these two steps represents each formula A in $\mathscr{A}$ by a term $S^{\&A}0$ and the representation is bijective.

Gödel coding is analogous to the mechanism of indirect addressing in computer instructions and pointers in programming languages. Let us illustrate this analogy using the language C. Suppose that x is an integer variable and p is an integer pointer, with $\&x$ denoting the address of x. After the execution of the statement $p = \&x$, p points to the address of x and $*p$ represents the content stored in the address $\&x$.

The analogy with Gödel coding amounts to regarding each formula A (a symbol string) in a first-order language as the name of a variable in C, whose Gödel number $\&A$ is the storage address of the variable A and the Gödel term $S^{\&A}0$ is the content stored at the address $\&A$.

Gödel coding is defined inductively. First, we define the concept of ordinal number in Gödel coding.

Definition 1.8 (Ordinal number). Suppose that $a_1, a_2, \ldots, a_n$ are natural numbers. $< a_1, a_2, \ldots, a_n >$ is called the *ordinal number* of $a_1, a_2, \ldots, a_n$ and it represents the natural number $p_1^{a_1+1} p_2^{a_2+1} \cdots p_n^{a_n+1}$ with $p_1, \ldots, p_n$ being the first n prime numbers. Namely,

$$< a_1, a_2, \ldots, a_n >= p_1^{a_1+1} p_2^{a_2+1} \cdots p_n^{a_n+1},$$

where a_i $(0 < i \leqslant n)$ is called the *i-th element* of this ordinal number.

An ordinal number is a natural number. Any element a_i of an ordinal number will still be an ordinal number. Nonetheless, not every natural number is an ordinal number and, as an example, 0 is not an ordinal number.

Definition 1.9 (Gödel coding). The *Gödel coding* of $\mathscr{A}$ is a map $\& : \mathscr{A} \mapsto \mathbb{N}$. $\&$ maps each symbol, term or formula of $\mathscr{A}$ to a natural number. According to the syntactic structure of symbols, terms and formulas of $\mathscr{A}$, $\&$ is inductively defined as follows.

(1) Symbols

$$\begin{aligned} \&(0) &= 1, & \qquad \&(\neg) &= 13, \\ \&(S) &= 3, & \qquad \&(\vee) &= 15, \\ \&(+) &= 5, & \qquad \&(\wedge) &= 17, \\ \&(\cdot) &= 7, & \qquad \&(\rightarrow) &= 19, \\ \&(\doteq) &= 9, & \qquad \&(\forall) &= 21, \\ \&(<) &= 11, & \qquad \&(\exists) &= 23. \end{aligned}$$

(2) Variables

$$\&(x_n) = 25 + 2 \cdot n, \qquad n \in \mathbb{N}.$$

It should be noted that the number 25 could be replaced by any odd number greater than 23 to allow us to introduce more symbols. This will be seen in Chapter 5.

(3) Terms

$$\&(St) =< \&S, \&t >,$$
$$\&(t_1 * t_2) =< \&(*), \&t_1, \&t_2 >, \quad \text{where } * \text{ stands for any of } +, \cdot.$$

(4) Formulas

$$\&(t_1 * t_2) =< \&(*), \&t_1, \&t_2 >, \qquad \text{where } * \text{ stands for any of } <, \doteq,$$
$$\&(\neg A) =< \&(\neg), \&A >,$$
$$\&(A * B) =< \&(*), \&A, \&B >, \qquad \text{where } * \text{ stands for any of } \wedge, \vee, \rightarrow, \leftrightarrow,$$
$$\&(\forall x_n A) =< \&(\forall), \&(x_n), \&A >,$$
$$\&(\exists x_n A) =< \&(\exists), \&(x_n), \&A > .$$

Example 1.8 (Gödel number). According to the rules of Gödel coding, we can determine effectively the Gödel number of each formula. For example, let A be a formula of the form

$$\forall x_3 \exists x_1 \; x_3 \doteq x_1 + x_2.$$

The Gödel number of A is

$$\begin{aligned}
&\&(\forall x_3 \exists x_1 \; x_3 \doteq x_1 + x_2) \\
&= \langle \&(\forall), \&(x_3), \&(\exists x_1 \; x_3 \doteq x_1 + x_2) \rangle \\
&= \langle 21, 31, \&(\exists x_1 \; x_3 \doteq x_1 + x_2) \rangle \\
&= \langle 21, 31, \langle 23, 27, \&(x_3 \doteq x_1 + x_2) \rangle \rangle \\
&= \langle 21, 31, \langle 23, 27, \langle 9, 31, \langle 5, 27, 29 \rangle \rangle \rangle \rangle \\
&= 2^{21+1} \cdot 3^{31+1} \cdot 5^{\langle 23, 27, \langle 9, 31, \langle 5, 27, 29 \rangle \rangle \rangle + 1} \\
&= 2^{21+1} \cdot 3^{31+1} \cdot 5^{2^{23+1} \cdot 3^{27+1} \cdot 5^{\langle 9, 31, \langle 5, 27, 29 \rangle \rangle + 1} + 1} \\
&= 2^{21+1} \cdot 3^{31+1} \cdot 5^{2^{23+1} \cdot 3^{27+1} \cdot 5^{2^{9+1} \cdot 3^{31+1} \cdot 5^{2^{5+1} \cdot 3^{27+1} \cdot 5^{29+1} + 1} + 1} + 1}.
\end{aligned}$$

The following lemma indicates that Gödel coding establishes a one-to-one correspondence between $\mathscr{A}$ and Gödel numbers.

Lemma 1.1. *Gödel coding is a one-to-one map from $\mathscr{A}$ to the set of Gödel numbers.*

Proof. The conclusion follows directly from the unique factorization theorem of prime numbers and the fact that the ordinal numbers are even so the odd variable codes will never coincide with them. □

Definition 1.10 (Gödel term). Let A be a formula of $\mathscr{A}$ and the Gödel number of A is $\&A$. The *Gödel term* of A is $S^{\&A}0$.

Example 1.9 (Gödel term). The Gödel term of the formula $\forall x_3 \exists x_1\ x_3 \doteq x_1 + x_2$ is

$$S^{2^{2\cdot 1+1}\cdot 3^{3\cdot 1+1}\cdot 5^{2^{2\cdot 3+1}\cdot 3^{2\cdot 7+1}\cdot 5^{2^{2\cdot 9+1}\cdot 3^{3\cdot 1+1}\cdot 5^{2^{2\cdot 5+1}\cdot 3^{2\cdot 7+1}\cdot 5^{2^{2\cdot 9+1}+1}+1}+1}+1}}0.$$

If $\mathscr{L}$ is a first-order language extending $\mathscr{A}$ which contains extra symbols, then we can still define their Gödel numbers and Gödel terms using the above method. We will see in Chapter 5 that the original intention of Gödel was to represent the self-referential statements in first-order languages so as to prove the incompleteness of formal theories. Nonetheless the idea of Gödel coding inspired the development of indirect addressing in computer hardware as well as the pointers in programming languages. In this sense, Gödel is the pioneer of these mechanisms.

1.6 Proof by structural induction

In the previous sections, the terms, formulas, free variables and substitutions of first-order languages are all defined by structural induction. In this section, we show how to use the inductive nature of these definitions to prove general properties of formulas in first-order languages.

Let's take the definition of formulas as an example. By structural induction, we first define the atomic formulas, which are equations and predicates, and then define the composite formulas by three F-rules (actually seven rules). These rules tell us how a composite formula is constructed from its components. Each F-rule can be written in a mathematical form. For instance, the rule on the disjunction formula in F_4 is "if A and B are formulas, then $A \vee B$ is a formula", which can be written in the form of a 'fraction'

$$\frac{A \quad B}{A \vee B}.$$

We should point out that A and B in the numerator of the fraction represent any logical formulas. Hence the above rule is a 'schema' to generate disjunction formulas. In general, each rule in a definition through structural induction can be written in the form of a 'fraction' as follows:

$$\frac{X_1 \ \cdots \ X_n}{X},$$

where the uppercase letters $X_1, \ldots, X_n$, X represent well-formed objects. The objects $X_1, \ldots, X_n$ in the numerator of the fraction are identified as the *premise* and the denominator X of the fraction is called the *conclusion* of the rule. This rule can be interpreted as: if the premise $X_1, \ldots, X_n$ holds, then the conclusion X holds.

In mathematical investigations, we often need to prove that a class of objects possess a certain property, which is usually the most difficult part of the whole investigation. Nowadays there are still many mathematical conjectures with their rigorous proofs pending. Nonetheless, if an object is defined by structural induction, then the proof of its properties may become rather simple and even turn into a kind of routine schema. The reason is that it suffices to verify under such circumstances that the atomic objects possess the property and each composite object possesses the property, from which we can deduce that all objects possess the property.

The composite object is a conclusion of a certain rule according to the definition by structural induction. Thus it suffices to prove that, for every rule defining composite objects, the premises having the property implies the conclusion also has the property. This kind of proof method is called the proof by structural induction, or structural induction for short. It can be strictly stated as follows.

Method 1.1 (Structural induction). Suppose that a set $\mathbf{Z}$ is defined by a group of rules. To prove that the set $\mathbf{Z}$ possesses a property Ψ, we only need to prove the following.

I_1: Each atomic object that is directly defined possesses the property Ψ;

I_2: For each rule

$$\frac{X_1 \ \cdots \ X_n}{X},$$

if $X_1, \ldots, X_n$ all possess the property Ψ, then we can prove that X also possesses the property Ψ.

I_1 is called the *induction basis*. The condition "if $X_1, \ldots, X_n$ all possess the property Ψ" specified in I_2 is identified as the *induction hypothesis*.

The proof method of structural induction can be applied to the proofs of terms and formulas, which can be summarized in the following proof schema.

Method 1.2 (Proof that terms possess the property Ψ). To prove that each term possesses the property Ψ, we only need to prove:

T_1': Each variable possesses the property Ψ;

T_2': Each constant possesses the property Ψ;

T_3': If terms $t_1, \ldots, t_n$ all possess the property Ψ and f is an n-ary function symbol, then $ft_1 \cdots t_n$ also possesses the property Ψ.

Method 1.3 (Proof that formulas possess the property Ψ). To prove that each formula possesses the property Ψ, we only need to prove:

F_1': $t_1 \doteq t_2$ possesses the property Ψ;

F_2': For any n-ary predicate symbol R and terms $t_1, \ldots, t_n$, $Rt_1 \cdots t_n$ possesses the property Ψ;

F_3': If A possesses the property Ψ, then so does $(\neg A)$;

F_4': If A and B both possess the property Ψ, then so do $(A \wedge B)$, $(A \vee B)$, $(A \to B)$ and $(A \leftrightarrow B)$;

F_5': If A possesses the property Ψ, then so do both $\forall xA$ and $\exists xA$.

Let us look at the following example.

Example 1.10. For any given first-order language $\mathscr{L}$, every formula in $\mathscr{L}$ contains an equal number of left parentheses "(" and right parentheses ")".

Proof. We prove the conclusion first on terms by structural induction.

T_1': Every variable x has no parenthesis and thus the conclusion holds.

T_2': Every constant c also has no parenthesis and thus the conclusion holds as well.

T_3': For any term $ft_1 \cdots t_n$ with f being an n-ary function symbol, every term t_i $(i = 1, \ldots, n)$ contains an equal number of left and right parentheses according to the assumption of structural induction. As per T_3' of Method 1.2, no new parenthesis is added to the term $ft_1 \cdots t_n$ and the number of left or right parentheses contained in it equals the total number of left or right parentheses contained in $t_1, \ldots, t_n$. Thus the conclusion holds for terms.

The proof by structural induction on formulas proceeds as follows.

F_1': The conclusion holds for $t_1 \doteq t_2$. Since t_1 and t_2 are terms, the conclusion holds for t_1 and t_2 according to the first part of the proof, and no new parenthesis is added to the formula $t_1 \doteq t_2$.

F_2': The conclusion holds for $Rt_1 \cdots t_n$. The reason is that $t_1, \ldots, t_n$ are all terms and as per the first part of the proof, the conclusion holds for $t_1, \ldots, t_n$; R is an n-ary predicate that contains no parenthesis in itself and thus no new parenthesis is added to the formula $Rt_1 \cdots t_n$.

F_3': Suppose that A is a formula that contains an equal number of left and right parentheses. According to Definition 1.3, $(\neg A)$ contains an equal number of left and right parentheses.

F_4': Suppose that the conclusion holds for both the formulas A and B. If we assume that A contains n left parentheses and n right parentheses and B contains m left parentheses and m right parentheses, then according to Definition 1.3 the formula $(A \wedge B)$ contains $n + m + 1$ left parentheses and $n + m + 1$ right parentheses. Thus the conclusion holds for $(A \wedge B)$. Similarly we can prove that the conclusion holds for $(A \vee B)$, $(A \to B)$ and $(A \leftrightarrow B)$ as well.

F_5': Suppose that formula A contains an equal number of left and right parentheses. According to the definition, the numbers of the left parentheses and right parentheses contained in $\forall xA$ or $\exists xA$ equal the numbers of those contained in A respectively.

The conclusion is proved. □

In fact, any property that can be proved by structural induction can also be proved by mathematical induction. In this sense, we say that the proofs using the structural induction method are rational. The bridge connecting the structural induction method and the mathematical induction method is the rank of terms and formulas.

Definition 1.11 (Rank of terms). The *rank* of a term t is a natural number denoted as $\mathrm{rk}(t)$ and it can be inductively defined as follows.

(1) $\mathrm{rk}(c) = 1$.

(2) $\mathrm{rk}(x) = 1$.

(3) $\mathrm{rk}(ft_1 \cdots t_n) = \max\{\mathrm{rk}(t_1), \ldots, \mathrm{rk}(t_n)\} + 1$.

Here $\max\{k_1, \ldots, k_n\}$ stands for the maximum of $k_1, \ldots, k_n$, with $k_1, \ldots, k_n$ being natural numbers.

Definition 1.12 (Rank of formulas). The *rank* of a formula A is a natural number denoted as $\mathrm{rk}(A)$ and it can be inductively defined as follows.

(1) $\mathrm{rk}(Pt_1 \cdots t_n) = 1$.

(2) $\mathrm{rk}(t_1 \doteq t_2) = 1$.

(3) $\mathrm{rk}(\neg A) = \mathrm{rk}(A) + 1$.

(4) $\mathrm{rk}(A * B) = \max\{\mathrm{rk}(A), \mathrm{rk}(B)\} + 1$, where $*$ stands for any of $\vee, \wedge, \rightarrow, \leftrightarrow$.

(5) $\mathrm{rk}(\forall xA) = \mathrm{rk}(A) + 1$.

(6) $\mathrm{rk}(\exists xA) = \mathrm{rk}(A) + 1$.

The method of proof by structural induction will have extensive applications in this book, because the syntax of first-order languages is defined by structural induction. Because the syntax of most programming languages is also defined by structural induction (using the Backus normal form), this method can be used for proving many kinds of properties of computer programs. More generally, the properties of any object that is defined by structural induction, in principle, can all be proved by structural induction. All these proofs follow a kind of routine schema, which makes it possible to complete such proofs by well-designed software systems. In fact, definition by structural induction forms the basis for computer-aided and computer-automated proof systems.

Since definitions by structural induction have such an advantage, wouldn't mathematical proofs become much simpler if every mathematical object were defined by structural induction? Regrettably, not every mathematical object can be defined in this way and later on we will encounter some objects for which this is not possible. The problem of identifying which objects can be inductively defined and which cannot is difficult.

Chapter 2

Models of First-Order Languages

As we mentioned in the previous chapter, terms and formulas are all symbol strings. To make use of a first-order language, the terms and formulas need to be interpreted as saying something meaningful about a domain. This semantic interpretation gives meaning to the symbol strings and is called a model of the language. In this chapter, we will build a general theory of semantics for first-order languages. The key ideas are as follows:

(1) Object languages and meta-languages. In Chapter 1 we used two languages. The first-order languages we defined are called *object languages* and the language we used to explain first-order languages we call *meta-languages*. The first-order languages are defined and explained in the *meta-language*. For instance, in Example 1.3, in the logical formula $\exists z(y \doteq +xz)$, "$\doteq$" is a symbol of the first-order language and the formula can be interpreted by the proposition "there exists a natural number z such that $y = x + z$ holds," where "$=$" in the proposition is the equality relation as used in high-school algebra. The equality $=$ belongs to the meta-language.

The object language and its meta-language occur together wherever scientific research takes place. For instance, in a text book on physics, which talks about specific concepts such as mass, acceleration and force, we need to introduce symbols such as m, a, and F to denote these concepts so that we can specify laws of physics such as $F = m \cdot a$. These special symbols and equations constitute the object language of physics, or physics for short. The object language uses this terminology to precisely specify the laws and principles that natural phenomena obey. The natural language that we use to explain these symbols and equations is the meta-language of physics.

As another example, when we are learning Latin, Latin is the object language and the English which is used to interpret the Latin becomes the meta-language. Hence, in a Latin-English dictionary Latin words are in the object language and the English interpretation belongs to the meta-language.

Generally speaking, an object language restricts the scope of its usage by introducing special terminology, whereas a meta-language explains this terminology by using existing knowledge. First-order languages are the object languages of this book, the meta-language that we use to describe them is English. Our existing knowledge allows us to understand first-order languages from their description in the meta-language.

(2) The relativity of object languages and meta-languages. A language can be an object language in one context and become a meta-language in another context. For example, in a manual of C, the programming language C is the object language and C programs are its syntactic objects, while the natural language used to explain C statements is the meta-language. Only through explanations using natural language can we understand the meaning of each C statement. However, the language C becomes the meta-language

when it is used to interpret Java programs. Through an interpreter written in the language C a Java program can be executed by computers.

Generally speaking, when we have acquired a profound knowledge of a language through studying it as an object language, it may be used as a meta-language to interpret and explain the terminology of another object language and to prove relations and properties of this language. This is a fundamental method used in scientific research. From this point of view, we can regard first-order languages as meta-languages and use them to interpret the object language that describes the domain and to prove logical relations and the properties of its objects.

(3) Two key components of interpretations. A meta-language is used to interpret object languages. To precisely interpret the symbols and objects of an object language, one needs two key ingredients:

The first requirement is a specific knowledge domain whose elements are identified with the object symbols in the language. This is usually a mathematical system, simply called a *domain*. The other requirement is a specific method of interpretation that maps symbols and objects in the object language to their corresponding elements in the domain. For example, let the first-order language be an object language, f be the symbol of its binary function, and P be the symbol of its binary predicate. According to the definition in Chapter 1, we know that the symbol string

$$A: \ \forall x \forall y \forall z (Pxy \rightarrow Pfxzfyz)$$

is a formula. If we are asked what it means, it is unlikely that we can give an immediate answer. If we choose the system of natural numbers as the domain, assume that the variables x, y, z can only take natural numbers, interpret the binary function symbol f as addition of natural numbers, i.e., fxz denotes $x+z$, and interpret the binary predicate P as the "less than" relation between natural numbers, then Pxy denotes $x < y$. Moreover, we interpret the quantifier symbol and bound variable $\forall x$ as "for all natural numbers x," the logical connective symbol $\rightarrow$ as "if ... then ...". With these interpretations, the formula A can be interpreted as the following true proposition about the domain of natural numbers:

$$\text{for all natural numbers } x, y, z, \text{ if } x < y, \text{ then } (x+z) < (y+z).$$

We can see from this example that the semantics provided by the meta-language should contain not only the domain but also an interpretation. A domain and an interpretation combined together define a *model* of a first-order language. In Chapter 1 we viewed the terms and formulas of a first-order language as symbol strings with definite syntactic structures. After choosing a domain and an interpretation, constant symbols and function symbols are interpreted as elements and functions in the domain, predicate symbols are interpreted as basic concepts and relations in the domain, and formulas are interpreted as propositions about the domain. In this case, we say that the semantics of each term of the language is an element or a function of the domain, each logical formula is interpreted as a proposition about the domain, and the semantics of this logical formula is the truth of the proposition.

(4) The variability of domains and interpretations. One object language may have many models. For example, in the formula A in (3), we can take the field of real

numbers as its domain, change the scope of variables to the set of all real numbers, interpret f as multiplication over the field of real numbers, interpret P as the "less than" relation over the field of real numbers, and the interpretations for the logical connectives $\forall$ and $\rightarrow$ remain unchanged. In this case, formula A is interpreted as the following proposition about the field of real numbers:

$$\text{for all real numbers } x, y, z, \text{ if } x < y, \text{ then } x \cdot z < y \cdot z.$$

Over the field of real numbers, this proposition is no longer true because the proposition does not hold when z is a negative number. This illustrates that there can be different domains and interpretations for the same first-order language. This is the variability of the domains and interpretations of an object language.

(5) The invariability of semantics of logical connectives. From the above example, we can see that, for different domains and interpretations, the semantics of terms and formulas of a first-order language can be completely different. However, in the previous two examples the interpretation for logical connectives remains the same. In other words, the semantics of logical connectives is independent of domains and interpretations. This semantic invariability of logical connectives is indispensable if we want to convert logical reasoning about domain knowledge into a symbolic calculus.

(6) The dual nature of a language in the same domain and interpretation. We discussed previously that a language is an object language with respect to a domain and an interpretation and is a meta-language with respect to another object language. What we want to point out here is that a language can be both an object language and a meta-language. A typical example is the Oxford English dictionary. The English entries in the dictionary are the objects of study and belong to the object language, while the text used to interpret each English entry is also English, but it belongs to the meta-language.

More generally, the terms of a first-order language form a set of symbol strings. This set can be viewed as a domain of a model for the language, in which the elements are simply symbol strings and the interpretation of function symbols are maps from strings to strings. This is what we mean by "the dual nature of the object language". In this chapter, we shall show how to do this by defining the Herbrand domain. The fact that the object language has this dual nature as a domain is a key to the proof of completeness for formal inference based on a first-order language.

Distinguishing object languages from their meta-languages gives clarity to thought. This is an essential difference between scientific investigation and daily discourse and therefore can be seen as a major milestone in the development of the theory of knowledge.

We shall see later that this methodology not only helps eliminate the ambiguities of propositions in meta-language but can further convert logical reasoning to symbolic calculus. Generally speaking, if an object can be described by a certain language, then one can design an object language to investigate this object and determine the semantics of the object language by introducing models.

This chapter presents the main concepts of models of first-order languages. In Section 2.1 the concepts of domains, interpretations, and structures as well as the principle of excluded middle in domains will be introduced. The concepts of assignments and models will be given in Section 2.2 and the semantics of terms will be discussed in Section 2.3.

Section 2.4 will present the semantics of logical connectives. The latter remain invariant in first-order languages as well as their models and meta-languages. The semantics of logical formulas will be discussed in Section 2.5. The satisfiability and validity of formulas and sets of formulas will be given in Section 2.6. Section 2.7 is devoted to valid formulas about the equivalence symbol. In Sections 2.8 to 2.10, Hintikka sets, the Herbrand domain and the satisfiability of Hintikka sets will be introduced. The substitution lemma will be presented in Section 2.11 and the proof of this lemma will be given in Appendix 2. Finally, isomorphism between models is discussed in Section 2.12.

2.1 Domains and interpretations

As mentioned previously, to make the terms and formulas of a first-order language meaningful, we need to determine a domain and an interpretation that specifies the meaning of constant symbols, function symbols and predicate symbols in the domain. The purpose of this section is to give a mathematical description of domains and interpretations.

A domain is a mathematical system denoted by $\mathbb{M}$. It consists of three parts. The first is a nonempty set M. The second is a nonempty set of functions, each of which has M or the Cartesian product of several M's as its domain and has M as its range. The third is a nonempty set of propositions, each of which represents a relation between the elements and functions of M. The natural number system $\mathbb{N}$, the rational number system $\mathbb{Q}$, and the real number system $\mathbb{R}$ are all typical examples of domains. For simplicity, we will often follow convention and not make any discrimination between the domain $\mathbb{M}$ and its set of elements M.

Before defining these concepts in detail we should mention an important assumption, which we adopt in this book, called the *principle of excluded middle*.

Principle 2.1 (Principle of excluded middle). *Each proposition in a domain $\mathbb{M}$ is either true or false and there is no other choice.*

The principle of excluded middle is a basic assumption in classical mathematical logic, whose status is equivalent to that of the postulate of parallels in plane geometry or that of the Galilei transformation in classical mechanics.

Interpretation is a mapping that interprets each constant symbol in the first-order language as an element in M, each n-ary function symbol as an n-ary function in M, and each n-ary predicate symbol as an n-ary relation in M. A domain coupled with an interpretation is called a *structure*, which is defined as follows:

Definition 2.1 (Structure of $\mathscr{L}$). The *structure* $\mathbf{M}$ of a first-order language $\mathscr{L}$ is a pair

$$\mathbf{M} = (M, I),$$

with the following properties.

(1) M is a nonempty set identified as a *domain*.

(2) I is a map from $\mathscr{L}$ to M called an *interpretation* and is denoted by $I : \mathscr{L} \to M$, which satisfies:

(i) for each constant symbol c in $\mathscr{L}$, $I(c)$ is an element in M;

(ii) for each n-ary function symbol f in $\mathscr{L}$, $I(f)$ is an n-ary function in M;

(iii) for each n-ary predicate symbol P in $\mathscr{L}$, $I(P)$ is an n-ary relation on elements of M.

For the convenience of writing, $I(c)$, $I(f)$ and $I(P)$ are often denoted as $c_{\mathbf{M}}$, $f_{\mathbf{M}}$ and $P_{\mathbf{M}}$. They are the interpretations of the constant symbol c, function symbol f and predicate symbol P respectively in $\mathbf{M}$, or as their *semantics* with respect to $\mathbf{M}$.

Example 2.1 (Structure of $\mathscr{A}$). We will illustrate all these semantic concepts through the language of elementary arithmetic $\mathscr{A}$ given in Example 1.1 and the subsequent three examples.

The symbol sets of $\mathscr{A}$ are the set of the constant symbol $\{0\}$, the set $\{S,\ +,\ \cdot\}$ of function symbols and the set of the predicate symbols $\{<\}$. We define a pair $\mathbf{N} = (\mathbb{N}, I)$ with the domain $\mathbb{N}$ being the set of natural numbers. Let s be the successor function on $\mathbb{N}$, i.e., $s(x) = x+1$, $+$ and $\cdot$ represent addition and multiplication on $\mathbb{N}$ respectively, and $<$ signifies the "less than" relation on $\mathbb{N}$. We further define the map I of interpretation as follows:

$$\begin{aligned} I(0) &= 0, \\ I(S) &= s, \\ I(+) &= +, \\ I(\cdot) &= \cdot, \\ I(<) &= <. \end{aligned}$$

The symbols 0, S, $+$, $\cdot$ and $<$ on the left-hand side of the above equalities are the constant symbol, function symbols and predicate symbol of the object language $\mathscr{A}$ respectively, while 0, s, $+$, $\cdot$ and $<$ on the right-hand side of the above equalities are mathematical entities used in the domain, i.e., the constant 0, the successor function s, addition, multiplication and the "less than" relation on $\mathbb{N}$ respectively.

The interpretation I maps the constant symbol 0 to the natural number 0, and the unary function symbol S to the successor operation s on $\mathbb{N}$. It maps the binary function symbols $+$ and $\cdot$ to addition and multiplication respectively and the binary predicate symbol $<$ to the "less than" relation on $\mathbb{N}$. Hence $\mathbf{N} = (\mathbb{N}, I)$ is a structure of the language of elementary arithmetic $\mathscr{A}$. The interpretation is defined in the meta-language and $=$ used in the definition is the equal sign of the meta-language.

The various symbols in first-order languages and their interpretation as entities in structures are easy to confuse. Many authors distinguish between them by using different typefaces. For instance, the Oxford English dictionary prints the words of the object language (the entries) in boldface, whereas the explanations in the meta-language are written with normal weight. We purposely do not adopt this method in this book because we sometimes want to emphasize the dual nature of a language discussed above. Instead, we suggest that the reader pay attention to the context in which these symbols are used and be aware of their different possible meanings.

2.2 Assignments and models

We have pointed out in the first chapter that the free variables of first-order languages are analogous to the formal parameters of procedural declarations in programming languages. The procedural declarations with formal parameters are not executable in programming languages. It is only when the procedure is called and its formal parameters are replaced by real parameters that the procedure becomes executable. Similarly, for first-order languages, if there are free variables in terms and formulas, then the semantics of these terms and formulas depend on the values assigned to their free variables even if the domain and interpretation are fixed. The assignment of variables is defined as follows.

Definition 2.2 (Assignment). An *assignment* σ is a map whose domain is the variable set V and whose range is M. It is denoted by $\sigma : V \to M$. σ assigns an element $a \in M$ to each variable x in $\mathscr{L}$ such that $\sigma(x) = a$. The set of all assignments is denoted by $[V \to M]$.

A structure together with an assignment is called a model of a first-order language.

Definition 2.3 (Model). For a given first-order language $\mathscr{L}$ with an associated structure $\mathbf{M}$ and assignment σ, the pair $(\mathbf{M}, \sigma)$ is called a *model* of $\mathscr{L}$.

Example 2.2 (Model of $\mathscr{A}$). As in Example 2.1, $\mathbf{N} = (\mathbb{N}, I)$. Let the assignment be $\sigma(x_n) = n$ such that $(\mathbf{N}, \sigma)$ is a model of $\mathscr{A}$. In this model, not only the constant symbol, function symbols and predicate symbol of $\mathscr{A}$ have interpretations in the set of natural numbers but each variable of $\mathscr{A}$ also has a definite natural number as its value.

For each assignment σ, we can define a new assignment that will be used frequently in this book:

Definition 2.4 (Assignment $\sigma[x_i := a]$). Suppose that $a \in M$ and $\sigma : V \to M$ is an assignment. The *assignment* $\sigma[x_i := a]$ is defined as follows:

$$\sigma[x_i := a](y) = \begin{cases} \sigma(y), & \text{if} \quad y \neq x_i, \\ a, & \text{if} \quad y = x_i. \end{cases}$$

This definition indicates that the assignment $\sigma[x_i := a]$ assigns a to the variable x_i whereas for other variables the assignment is the same as σ.

2.3 Semantics of terms

For a first-order language $\mathscr{L}$, once its model $(\mathbf{M}, \sigma)$ is specified, its variables and constant symbols are interpreted as elements in M, its function symbols are interpreted as functions on M, and thus its terms are interpreted as elements in the domain M accordingly. In other words, its terms are "designated" as elements in the domain M. This defines the semantics of terms as follows:

Definition 2.5 (Semantics of terms). Given a first-order language $\mathscr{L}$, an associated structure $\mathbf{M} = (M, I)$ and assignment $\sigma : V \to M$, the semantics of a term t in the model $(\mathbf{M}, \sigma)$ is an element in M that is denoted by $t_{\mathbf{M}[\sigma]}$ and is inductively defined as follows:

(1) $x_{\mathbf{M}[\sigma]} = \sigma(x)$, x is a variable;

(2) $c_{\mathbf{M}[\sigma]} = c_{\mathbf{M}}$, c is a constant symbol;

(3) $(ft_1 \cdots t_n)_{\mathbf{M}[\sigma]} = f_{\mathbf{M}}((t_1)_{\mathbf{M}[\sigma]}, \ldots, (t_n)_{\mathbf{M}[\sigma]})$.

(1) shows that the semantics of a variable x is the value of the assignment σ at x, which is an element of M. (2) shows that the semantics of a constant symbol c is $c_{\mathbf{M}}$, i.e., $I(c)$, which is the value of I at c in the structure $\mathbf{M}$ and is also an element of M. (3) shows that the semantics of the term $ft_1 \cdots t_n$ is still an element of M, which can be obtained as follows: interpret f as an n-ary function $f_{\mathbf{M}}$ in M, i.e., $I(f)$, and evaluate $(t_i)_{\mathbf{M}[\sigma]}$ for $1 \leqslant i \leqslant n$ respectively to obtain n values in the domain M; then evaluate the function $f_{\mathbf{M}}$ at $((t_1)_{\mathbf{M}[\sigma]}, \ldots, (t_n)_{\mathbf{M}[\sigma]})$.

Example 2.3 (Terms of $\mathscr{A}$). In the model $(\mathbf{N}, \sigma)$ given in Example 2.2, the function symbol $+$ in $\mathscr{A}$ is interpreted as the addition of natural numbers and S is interpreted as the successor operation of natural numbers. Thus the term $+x_1Sx_7$ is interpreted as 9, or in other words, the semantics of this term in the model $(\mathbf{N}, \sigma)$ is 9. The calculation process to obtain this interpretation is

$$\begin{aligned}(+x_1Sx_7)_{\mathbf{N}[\sigma]} &= (x_1)_{\mathbf{N}[\sigma]} + (Sx_7)_{\mathbf{N}[\sigma]} \\ &= 1 + ((x_7)_{\mathbf{N}[\sigma]} + 1) \\ &= 1 + (7 + 1) \\ &= 9.\end{aligned}$$

The interpretation of the term $+S0SSx_1$ in the model $(\mathbf{N}, \sigma)$ is 4, whose calculation process is

$$\begin{aligned}(+S0SSx_1)_{\mathbf{N}[\sigma]} &= (S0)_{\mathbf{N}[\sigma]} + (SSx_1)_{\mathbf{N}[\sigma]} \\ &= s(0) + (SS(x_1))_{\mathbf{N}[\sigma]} \\ &= 1 + s(1) + 1 \\ &= 1 + 2 + 1 \\ &= 4.\end{aligned}$$

2.4 Semantics of logical connective symbols

Up to now, for a given model, we already know how to determine the semantics of each term or predicate. Nonetheless, this is not enough to determine the precise proposition which interprets a given formula and whether the proposition is true or not. It is only after the semantics of logical connective symbols are strictly defined that these two problems can be solved. For instance, the formula

$$\exists x(x < 4 \vee x < 2)$$

should be interpreted as the following proposition:

"There exists a natural number x such that either $x < 4$ or $x < 2$ holds."

Here we adopted the conventional interpretations of the quantifier symbol $\exists$ and logical connective symbol $\vee$. In other words, we interpreted $\vee$ as "or" and $\exists$ as "there exists". Nevertheless, the truth of the proposition in the domain $\mathbb{N}$ still cannot be determined because, in reality, there are two different understandings of "or". One is the "exclusive or", which is "or" in the exclusive sense. In this sense, "A or B holds" is regarded as "either A holds or B holds and there is no other choice". Under such circumstances the formula "$x < 2$ or $x < 4$" is true only when $x = 2,\ 3$. The other is the "inclusive or", which is "or" in the inclusive sense. In this sense, "A or B holds" is regarded as "at least one of A and B holds". Thus, when $x = 0, 1, 2, 3$, the formula "$x < 2$ or $x < 4$" is true. The variation between these two results is a consequence of the different possible meanings of the logical connective symbol $\vee$.

In order to avoid any confusion in the interpretation of logical connective symbols, we have to define their semantics consistently and strictly. Our approach is to define their semantics as truth functions whose domain is the set of truth values or a Cartesian product of two sets of truth values and whose range is also a set of truth values.

Definition 2.6 (Set of truth values). We define the *set of truth values* as the set $\{\mathrm{T}, \mathrm{F}\}$, which we call a Boolean set, that contains only two elements with T representing truth and F representing falsity.

Definition 2.7 (Semantics of logical connective symbols). For first-order languages, the function of the logical connective symbol $\neg$ is $\mathbf{B}_\neg$, whose variable X can only take truth values T or F. The function values $\mathbf{B}_\neg(X)$ are defined by the following table:

X	T	F
$\mathbf{B}_\neg(X)$	F	T

Suppose that the binary functions $\mathbf{B}_\vee$, $\mathbf{B}_\wedge$, $\mathbf{B}_\rightarrow$ and $\mathbf{B}_\leftrightarrow$ are functions of the logical connective symbols $\vee$, $\wedge$, $\rightarrow$ and $\leftrightarrow$ respectively. They are defined by the following table:

X	Y	$\mathbf{B}_\vee(X,Y)$	$\mathbf{B}_\wedge(X,Y)$	$\mathbf{B}_\rightarrow(X,Y)$	$\mathbf{B}_\leftrightarrow(X,Y)$
T	T	T	T	T	T
T	F	T	F	F	F
F	T	T	F	T	F
F	F	F	F	T	T

The table of $\mathbf{B}_\neg$ shows that if X is true, i.e., it takes the value T, then the value of $\mathbf{B}_\neg(X)$ is F, i.e., it is false. Conversely, if X takes the value F, then the value of $\mathbf{B}_\neg(X)$ is T.

We use $\mathbf{B}_\vee(X,Y)$ to illustrate the semantics of the logical connective symbol $\vee$. The truth values of the variables X and Y only have four different combinations (T, T), (T, F), (F, T) and (F, F). When X and Y take the value (T, F), downwards along the column beginning with $\mathbf{B}_\vee$ and rightwards along the row beginning with (T, F), the intersection point T of the row and column is the value of $\mathbf{B}_\vee$(T, F). $\mathbf{B}_\vee$, defined by the truth table, can

also be defined as the following:

$$\mathbf{B}_{\vee}(X,Y)=\begin{cases}\mathrm{T}, & \text{if } X=\mathrm{T} \text{ and } Y=\mathrm{T},\\ \mathrm{T}, & \text{if } X=\mathrm{T} \text{ and } Y=\mathrm{F},\\ \mathrm{T}, & \text{if } X=\mathrm{F} \text{ and } Y=\mathrm{T},\\ \mathrm{F}, & \text{if } X=\mathrm{F} \text{ and } Y=\mathrm{F}.\end{cases}$$

The functions $\mathbf{B}_{\wedge}$, $\mathbf{B}_{\rightarrow}$ and $\mathbf{B}_{\leftrightarrow}$ can all be defined by similar methods.

The above definitions show that the arguments of these logical functions are simply the truth values of propositions. They are independent of the structures and assignments of first-order languages and they are only determined by logical connective symbols.

2.5 Semantics of formulas

Each formula is interpreted as a proposition in the domain after a model is given with the semantics of logical connective symbols being defined. The semantics of this formula in the model is naturally defined as the truth value of its corresponding proposition because each proposition in the model is either true or false. For a given model, we can define the semantics of logical formulas via the method of structural induction. The semantics of atomic formulas is directly determined by the structure and assignment, whereas the semantics of composite formulas is determined by the truth values of its subformulas as well as the semantics of logical connective symbols.

Definition 2.8 (Semantics of formulas). Let $\mathbf{M}$ and σ be a structure and assignment associated with a first-order language $\mathscr{L}$ and let A be a formula of $\mathscr{L}$. The semantics of the formula A in the model $(\mathbf{M},\sigma)$ is a truth value denoted by $A_{\mathbf{M}[\sigma]}$, defined by the structural induction as follows:

(1) $(Pt_1\cdots t_n)_{\mathbf{M}[\sigma]}=P_{\mathbf{M}}((t_1)_{\mathbf{M}[\sigma]},\ldots,(t_n)_{\mathbf{M}[\sigma]})$;

(2) $(t_1\doteq t_2)_{\mathbf{M}[\sigma]}=\begin{cases}\mathrm{T}, & \text{if } (t_1)_{\mathbf{M}[\sigma]}=(t_2)_{\mathbf{M}[\sigma]},\\ \mathrm{F}, & \text{otherwise;}\end{cases}$

(3) $(\neg A)_{\mathbf{M}[\sigma]}=\mathbf{B}_{\neg}(A_{\mathbf{M}[\sigma]})$;

(4) $(A\vee B)_{\mathbf{M}[\sigma]}=\mathbf{B}_{\vee}(A_{\mathbf{M}[\sigma]},B_{\mathbf{M}[\sigma]})$;

(5) $(A\wedge B)_{\mathbf{M}[\sigma]}=\mathbf{B}_{\wedge}(A_{\mathbf{M}[\sigma]},B_{\mathbf{M}[\sigma]})$;

(6) $(A\rightarrow B)_{\mathbf{M}[\sigma]}=\mathbf{B}_{\rightarrow}(A_{\mathbf{M}[\sigma]},B_{\mathbf{M}[\sigma]})$;

(7) $(A\leftrightarrow B)_{\mathbf{M}[\sigma]}=\mathbf{B}_{\leftrightarrow}(A_{\mathbf{M}[\sigma]},B_{\mathbf{M}[\sigma]})$;

(8) $(\forall x_i A)_{\mathbf{M}[\sigma]}=\begin{cases}\mathrm{T}, & \text{if for every } a\in M,\ A_{\mathbf{M}[\sigma[x_i:=a]]}=\mathrm{T} \text{ holds,}\\ \mathrm{F}, & \text{otherwise;}\end{cases}$

(9) $(\exists x_i A)_{\mathbf{M}[\sigma]} = \begin{cases} \mathrm{T}, & \text{if there exists an } a \in M \text{ such that } A_{\mathbf{M}[\sigma[x_i:=a]]} = \mathrm{T} \text{ holds,} \\ \mathrm{F}, & \text{otherwise.} \end{cases}$

If $A_{\mathbf{M}[\sigma]}$ is true, then we say that the formula A is true in the model $(\mathbf{M}, \sigma)$.

In Definition 2.8, (1) shows that under the structure $\mathbf{M}$ and assignment σ, P is interpreted as an n-ary relation $P_{\mathbf{M}}$ in the domain M, t_i is interpreted as an element $(t_i)_{\mathbf{M}[\sigma]}$ of M, and thus predicate symbol $Pt_1 \cdots t_n$ is interpreted as a truth value that indicates whether the n-ary relation $P_{\mathbf{M}}$ holds at the point $((t_1)_{\mathbf{M}[\sigma]}, \ldots, (t_n)_{\mathbf{M}[\sigma]})$ in the model $(\mathbf{M}, \sigma)$.

(2) shows that for the given structure $\mathbf{M}$ and assignment σ, the semantics of the formula $(t_1 \doteq t_2)$ is true under the model $(\mathbf{M}, \sigma)$ if $(t_1)_{\mathbf{M}[\sigma]}$ and $(t_2)_{\mathbf{M}[\sigma]}$ are equal in $\mathbf{M}$. It is false otherwise.

(3) shows that under the structure $\mathbf{M}$ and assignment σ, the truth value of the formula $\neg A$ is exactly opposite to that of the formula A.

In (4), $A_{\mathbf{M}[\sigma]}$, $B_{\mathbf{M}[\sigma]}$ and $(A \vee B)_{\mathbf{M}[\sigma]}$ in the equality represent the semantics of A, B and $A \vee B$, i.e., their truth values, respectively under the structure $\mathbf{M}$ and assignment σ. According to the definition of $\mathbf{B}_{\vee}$, we have

$$(A \vee B)_{\mathbf{M}[\sigma]} = \begin{cases} \mathrm{T}, & \text{if } A_{\mathbf{M}[\sigma]} = \mathrm{T} \text{ and } B_{\mathbf{M}[\sigma]} = \mathrm{T}, \\ \mathrm{T}, & \text{if } A_{\mathbf{M}[\sigma]} = \mathrm{T} \text{ and } B_{\mathbf{M}[\sigma]} = \mathrm{F}, \\ \mathrm{T}, & \text{if } A_{\mathbf{M}[\sigma]} = \mathrm{F} \text{ and } B_{\mathbf{M}[\sigma]} = \mathrm{T}, \\ \mathrm{F}, & \text{if } A_{\mathbf{M}[\sigma]} = \mathrm{F} \text{ and } B_{\mathbf{M}[\sigma]} = \mathrm{F}. \end{cases}$$

(5)–(7) can be illustrated in a similar way.

(8) shows that for any given structure $\mathbf{M}$ and assignment σ, if the interpretation of the formula A is always true after assigning *any* element a of M to x_i in the formula A, then $\forall x_i A$ is true. It is false otherwise.

(9) shows that for any given structure $\mathbf{M}$ and assignment σ, if there exists an element a of M which makes the interpretation of the formula A true after assigning a to x_i in the formula A, then $\exists x_i A$ is true. It is false otherwise.

Note that $A_{\mathbf{M}[\sigma[x_i:=a]]}$ represents the truth value of the formula A under the structure $\mathbf{M}$ and assignment $\sigma[x_i := a]$. It is different from $A[a/x_i]$ introduced in Chapter 1, which is a formula of first-order languages. Instead, it is the formula obtained by substituting the constant symbol a for the free variable x_i in A. In a word, the first is a truth value expressing the semantics of A under the assignment $\sigma[x_i := a]$, whereas the other is a logical formula substituting x_i by the constant symbol a.

Example 2.4 (Formulas of $\mathscr{A}$). According to the structure $\mathbf{N}$ and assignment σ defined in Example 2.2, the term $(+x_1 S x_7)$ of $\mathscr{A}$ is interpreted as 9, that is $(+x_1 S x_7)_{\mathbf{N}[\sigma]} = 9$. Since we also have $(x_9)_{\mathbf{N}[\sigma]} = 9$, we know from (2) in Definition 2.8 that

$$(+x_1 S x_7 \doteq x_9)_{\mathbf{N}[\sigma]} = \mathrm{T}$$

holds. This indicates that the formula $+x_1 S x_7 \doteq x_9$ holds in the model $(\mathbf{N}, \sigma)$.

We know from $\sigma(x_n) = n$ that $(x_2)_{\mathbf{N}[\sigma]} = 2$ and $(x_4)_{\mathbf{N}[\sigma]} = 4$ hold.

Suppose that x is a new variable. For the assignment $\sigma[x := 1]$, i.e., x assigned the natural number 1, it is not difficult to verify that

$$(< xx_4)_{\mathbf{N}[\sigma[x:=1]]} = \mathrm{T} \text{ and } (< xx_2)_{\mathbf{N}[\sigma[x:=1]]} = \mathrm{T}$$

hold. Further, by the definition of $\mathbf{B}_\vee$, we know that

$$(< xx_4 \vee < xx_2)_{\mathbf{N}[\sigma[x:=1]]} = \mathrm{T}.$$

According to (9) in Definition 2.8,

$$(\exists x(< xx_4 \vee < xx_2))_{\mathbf{N}[\sigma]} = \mathrm{T}$$

holds. This equality indicates that in the model $(\mathbf{N}, \sigma)$, the formula $\exists x(< xx_4 \vee < xx_2)$ is interpreted as the following proposition:

"There exists a natural number x such that $x < 4$ or $x < 2$"

and this proposition is true. Thus in the model $(\mathbf{N}, \sigma)$, the formula $\exists x(< xx_4 \vee < xx_2)$ is true.

The above example suggests three steps in determining the semantics of a formula. First, define a model (i.e., identify a domain), define an interpretation and introduce an assignment of symbols to variables. Secondly, determine inductively the semantics of each term in the formula starting from constant symbols and variables according to Definition 2.5. Thirdly, determine inductively the proposition by which the formula is interpreted in the domain, starting from atomic formulas according to Definition 2.8, and determine the truth of the proposition whose truth value is the semantics of the formula under the model.

Up to now we have introduced the domain $\mathbb{N}$ and structure $\mathbf{N}$ of the elementary arithmetic language $\mathscr{A}$ as well as the semantics of its terms and logical formulas.

Readers familiar with computer programming will have realized that the relationship between a program and its executable code is analogous to that between a first-order language and a model with the interpretation playing the role of a compiler.

For instance, if we view C as an object language then we can also see all sequences of executable code as its domain, which we will denote by $\mathbb{C}$. The compiler, C_I of C can then be regarded as an interpretation map. C and its compiler C_I can be discussed using natural language. This is its meta-language. In this analogy, each C program is a syntactic object and the semantic interpretation of this object is the machine code generated by the compiler C_I. Thus $(\mathbb{C}, C_I)$ can be regarded as a structure or a model of C.

Similarly, we may view compiled Java codes as an object language and then the C routines that execute these codes would be viewed as the domain. This illustrates again how a language like C can be both an object language and a semantic domain.

In summary, a first-order language, its model and their meta-language can be seen as a conceptual framework consisting of object language, model, and meta-language. A programming language, its implementing language and natural language form an identical framework. It is essential to keep the distinctions in this framework clear to avoid ambiguity and to ensure the correctness of implementation in software development.

2.6 Satisfiability and validity

We introduced in the previous sections the concept of model, i.e., the concepts of the domain, interpretation and assignment. We also discussed how to determine the semantics of terms and formulas for a given model. In this section we shall discuss another issue, that is, for a given formula A or formula set Γ, whether there exists a model $\mathbf{M}$ such that A or Γ is true under this model. Furthermore, we shall discuss whether A or Γ is true under any model. These two issues are described formally as the satisfiability and validity of formulas and formula sets respectively.

Definition 2.9 (Satisfiability). Given a first-order language $\mathscr{L}$ and its formula A and formula set Γ, if there exists a model $(\mathbf{M},\sigma)$ such that

$$A_{\mathbf{M}[\sigma]} = \mathrm{T}$$

holds, then formula A is said to be *satisfiable* under the model $(\mathbf{M},\sigma)$, or A is satisfiable for short. We also say that the model $(\mathbf{M},\sigma)$ satisfies A. This is denoted by $\mathbf{M} \models_\sigma A$. If A is a sentence, then it is denoted by $\mathbf{M} \models A$.

If every formula in Γ is satisfiable under the model $(\mathbf{M},\sigma)$, i.e.,

$$\mathbf{M} \models_\sigma A$$

holds for all $A \in \Gamma$, then we say that the formula set Γ is *satisfiable* under the model $(\mathbf{M},\sigma)$, or the formula set Γ is satisfiable for short. We can also say that the model $(\mathbf{M},\sigma)$ satisfies the formula set Γ or $(\mathbf{M},\sigma)$ is a model of Γ. This is denoted by $\mathbf{M} \models_\sigma \Gamma$. If Γ is a set consisting of sentences, then we denote it as $\mathbf{M} \models \Gamma$.

Definition 2.10 (Validity). A formula A is called *valid* if A is satisfiable under any model $(\mathbf{M},\sigma)$ of $\mathscr{L}$, that is, $\mathbf{M} \models_\sigma A$ holds for any structure $\mathbf{M}$ and any assignment σ. It is denoted by $\models A$.

A formula set Γ is called *valid* if each formula of Γ is valid. This is denoted by $\models \Gamma$.

A valid formula, also called a *tautology*, is irrelevant to models, and is true in any model.

Example 2.5. The formula

$$A \vee \neg A$$

is a valid formula.

For the formula $A \vee \neg A$, suppose that $(\mathbf{M},\sigma)$ is an arbitrary model. According to the principle of excluded middle in Section 2.1, the proposition $A_{\mathbf{M}}[\sigma]$ is either true or false. If $A_{\mathbf{M}}[\sigma]$ is true, then by (4) in Definition 2.8, $(A \vee \neg A)_{\mathbf{M}}[\sigma]$ is true; otherwise, if $A_{\mathbf{M}}[\sigma]$ is false, then the proposition $(\neg A)_{\mathbf{M}}[\sigma]$ is true by (3) in Definition 2.8. Then according to (4) in Definition 2.8, the proposition $(A \vee \neg A)_{\mathbf{M}}[\sigma]$ is still true. Thus the formula $A \vee \neg A$ is always true under any model $(\mathbf{M},\sigma)$.

Example 2.6. The formula

$$\forall x(x \doteq x)$$

is a valid formula.

Proof. For an arbitrary given model $(\mathbf{M}, \sigma)$,

$$(x)_{\mathbf{M}[\sigma]} = (x)_{\mathbf{M}[\sigma]}$$

holds. According to (2) in Definition 2.8,

$$(x \doteq x)_{\mathbf{M}[\sigma]} = \mathrm{T}$$

holds. Thus for every $a \in M$,

$$(x \doteq x)_{\mathbf{M}[\sigma[x:=a]]} = \mathrm{T}$$

holds. According to (8) in Definition 2.8, $\forall x(x \doteq x)$ is a valid formula. □

Definition 2.11 (Logical consequence). Let $\mathbf{M}$ be an arbitrary structure and σ be an arbitrary assignment. For any given formula A and any given formula set Γ,

$$\text{if } \mathbf{M} \models_\sigma \Gamma \text{ holds, then } \mathbf{M} \models_\sigma A \text{ holds.}$$

In this circumstance, A is a *logical consequence* or *semantic conclusion* of Γ. It is denoted as $\Gamma \models A$. It is also said that $\Gamma \models A$ is *valid*.

In Definitions 2.9, 2.10 and 2.11, $\models$ appears in four different forms which denote different semantic relationships. They are:

$$\mathbf{M} \models_\sigma A, \quad \mathbf{M} \models A, \quad \models A, \quad \Gamma \models A.$$

A simple method to discriminate the semantics of these forms is as follows: When both $\mathbf{M}$ and σ appear in a form, the semantic relation holds only for the given $\mathbf{M}$ and the given σ; when σ does not appear in a form, the semantic relation holds for every σ; when neither $\mathbf{M}$ nor σ appears in a form, the semantic relation holds for every $\mathbf{M}$ and every σ. $\Gamma \models A$ means that for every $\mathbf{M}$ and every σ, if Γ is true, then A is true as well. The following lemma will be used later in proving the completeness of logical inference systems.

Lemma 2.1. *If $\Gamma \models A$, then the formula set $\Gamma \cup \{\neg A\}$ is not satisfiable.*

Proof. We prove the lemma by contradiction. Suppose that there exist a structure $\mathbf{M}$ and an assignment σ such that they satisfy the formula set $\Gamma \cup \{\neg A\}$. Then both $\mathbf{M} \models_\sigma \Gamma$ and $\mathbf{M} \models_\sigma \neg A$ hold. Since $\Gamma \models A$ holds, if $\mathbf{M} \models_\sigma \Gamma$ holds by Definition 2.11, then $\mathbf{M} \models_\sigma A$ must hold. This leads to a contradiction, because according to the principle of excluded middle, A and $\neg A$ cannot hold simultaneously under the same model. □

2.7 Valid formulas with ↔

Valid formulas with the logical connective symbol ↔ are of special importance in that they define the equivalence between logical connective symbols and thus the number of logical connective symbols employed can be reduced.

Lemma 2.2 (Valid formulas with $\leftrightarrow$). *The following formulas with $\leftrightarrow$ are valid formulas:*

(1) $\models (A \wedge B) \leftrightarrow \neg(\neg A \vee \neg B)$;

(2) $\models (A \rightarrow B) \leftrightarrow \neg A \vee B$;

(3) $\models (A \leftrightarrow B) \leftrightarrow \neg(\neg(\neg A \vee B) \vee \neg(\neg B \vee A))$;

(4) $\models \forall x A \leftrightarrow \neg\exists x \neg A$.

Proof. First of all let us prove that the first formula is valid.

By (7) in Definition 2.8, it suffices to prove that for every structure $\mathbf{M}$ and every assignment σ,

$$(A \wedge B)_{\mathbf{M}[\sigma]} \text{ and } (\neg(\neg A \vee \neg B))_{\mathbf{M}[\sigma]}$$

share the same truth value. We build the following table according to Definition 2.8:

$A_{\mathbf{M}[\sigma]}$	T	T	F	F
$B_{\mathbf{M}[\sigma]}$	T	F	T	F
$(\neg A)_{\mathbf{M}[\sigma]}$	F	F	T	T
$(\neg B)_{\mathbf{M}[\sigma]}$	F	T	F	T
$(\neg A \vee \neg B)_{\mathbf{M}[\sigma]}$	F	T	T	T
$(\neg(\neg A \vee \neg B))_{\mathbf{M}[\sigma]}$	T	F	F	F
$(A \wedge B)_{\mathbf{M}[\sigma]}$	T	F	F	F
$((A \wedge B) \leftrightarrow \neg(\neg A \vee \neg B))_{\mathbf{M}[\sigma]}$	T	T	T	T

For any given $\mathbf{M}$ and σ, the truth values of the propositions $A_{\mathbf{M}[\sigma]}$ and $B_{\mathbf{M}[\sigma]}$ have only four combinations and they are (T, T), (T ,F), (F, T) and (F, F). The above table indicates that for these four combinations, the formula $(A \wedge B) \leftrightarrow \neg(\neg A \vee \neg B)$ is always true. Similarly, we can prove the validity of the other three formulas. □

According to (7) in Definition 2.8, the formula $A \leftrightarrow B$ is valid if and only if for every structure $\mathbf{M}$ and every assignment σ, $A_{\mathbf{M}[\sigma]}$ and $B_{\mathbf{M}[\sigma]}$ share the same truth value. In this case, we say that the formulas A and B are *equivalent*. Although A and B are different symbol strings in terms of syntactic rules, the propositions $A_{\mathbf{M}[\sigma]}$ and $B_{\mathbf{M}[\sigma]}$ share the same truth value for every model $(\mathbf{M}, \sigma)$ in terms of their semantics. Thus, if $A \leftrightarrow B$ is valid, then A can always be substituted by B wherever it appears. Such substitutions do not affect the interpretations of the formulas A and B. (1)–(4) in Lemma 2.2 indicate that wherever $A \wedge B$, $A \rightarrow B$, $A \leftrightarrow B$, and $\forall x A$ appear, they can always be substituted by the formulas $\neg(\neg A \vee \neg B)$, $\neg A \vee B$, $\neg(\neg(\neg A \vee B) \vee \neg(\neg B \vee A))$, and $\neg\exists x \neg A$ respectively. This implies that it suffices for a first-order language to use only three logical connective symbols, i.e., $\neg$, $\vee$, and $\exists$. Other combinations of logical connective symbols such as $\{\neg, \vee, \forall\}$, $\{\neg, \wedge, \exists\}$, $\{\neg, \wedge, \forall\}$, $\{\neg, \rightarrow, \forall\}$, and $\{\neg, \rightarrow, \exists\}$ have the same effect.

Thus, when we prove theorems by structural induction, in order to make the proof concise, it suffices to consider two logical connective symbols and one quantifier symbol

only. For instance, it suffices to only consider $\neg$, $\vee$, and $\exists$ and the proofs for the logical connective symbols $\wedge$, $\rightarrow$, and $\leftrightarrow$ and the quantifier symbol $\forall$ can be omitted. The justification is that the formulas containing these logical connective and quantifier symbols can be replaced by equivalent formulas containing $\neg$, $\vee$, and $\exists$.

2.8 Hintikka set

From the previous section we know that to prove the satisfiability of a formula set, it suffices to find a model $(\mathbf{M}, \sigma)$, i.e., a domain M, an interpretation I and an assignment σ, such that each formula in the formula set is true under the model $(\mathbf{M}, \sigma)$. As an example, we will introduce a specific set of sentences identified as the *Hintikka set* in this and the next section, and prove that every Hintikka set is satisfiable. The importance of discussing Hintikka sets is twofold: first, their satisfiability is the key to proving the completeness of the formal inference system of first-order languages in Chapter 3; secondly, the method for proving the satisfiability of Hintikka sets is stereotypical. In the proof, we construct a model called the *Herbrand model* for the language which embodies the dual nature of the object language that was discussed in the beginning of this chapter.

Definition 2.12 (Herbrand domain). Let $\mathscr{L}$ be a given first-order language with H being a nonempty subset consisting of the terms of $\mathscr{L}$. H is defined by structural induction as follows:

(1) if c is a constant symbol, then $c \in H$;

(2) if f is an n-ary function symbol and the terms $t_1, \ldots, t_n \in H$, then $ft_1 \cdots t_n \in H$.

H is identified as the *Herbrand domain* or *Herbrand universe* or *term domain* of $\mathscr{L}$, and the elements in H are called *Herbrand terms* or *ground terms*.

Definition 2.12 indicates that the Herbrand domain is a set consisting of the terms of a first-order language $\mathscr{L}$ that contains no variables. It is a subset of the set of terms. For instance, for the elementary arithmetic language $\mathscr{A}$, x, $Sx + S0$ and $SS0 + S0$ are all terms of $\mathscr{A}$. The first two terms are not ground terms and only $SS0 + S0$ is a ground term, an element of the Herbrand domain of $\mathscr{A}$.

Definition 2.13 (Hintikka set). Suppose that H is the Herbrand domain of a first-order language $\mathscr{L}$. We say that Ω is the *Hintikka set* with respect to H if Ω is a set of formulas whose elements satisfy the following seven conditions [Smullyan, 1968]. For all formulas:

(1) If A is an atomic formula, then either $A \in \Omega$ or $\neg A \in \Omega$[1].

(2) The formula $\neg\neg A \in \Omega$ if $A \in \Omega$.

(3) The formula $A \vee B \in \Omega$ if $A \in \Omega$ or $B \in \Omega$. The formula $\neg(A \vee B) \in \Omega$ if $\neg A \in \Omega$ and $\neg B \in \Omega$.

[1]For the equality symbol, we prescribe that $t \doteq t \in \Omega$. The technical details of the equality symbol can be found in [Gallier, 1986].

(4) The formula $A \wedge B \in \Omega$ if $A \in \Omega$ and $B \in \Omega$. The formula $\neg(A \wedge B) \in \Omega$ if $\neg A \in \Omega$ or $\neg B \in \Omega$.

(5) The formula $A \rightarrow B \in \Omega$ if $\neg A \in \Omega$ or $B \in \Omega$. The formula $\neg(A \rightarrow B) \in \Omega$ if $A \in \Omega$ and $\neg B \in \Omega$.

(6) The formula $\exists x A \in \Omega$ if there exists a term $t \in H$ such that $A[t/x] \in \Omega$. The formula $\neg\exists x A \in \Omega$ if $\neg A[t/x] \in \Omega$ holds for every $t \in H$.

(7) The formula $\forall x A \in \Omega$ if for every $t \in H$, we always have $A[t/x] \in \Omega$. The formula $\neg\forall x A \in \Omega$ if there exists a term $t \in H$ such that $\neg A[t/x] \in \Omega$.

It is obvious that for any given first-order language, there exists at least one Hintikka set. For example, the set Ω can be defined as follows: let every atomic sentence belong to Ω, then take the items of the above definition as generation rules to construct Ω. Then Ω is a Hintikka set of the first-order language.

We should point out that all the formulas in the above definition are actually sentences. The reason is that a Herbrand domain is a set of terms that contain no variables. All the formulas in Sections 2.8 and 2.9 are sentences because they contain no free variables. In Section 2.10, we shall discuss Herbrand domains that contain free variables.

A Hintikka set has the following property.

Lemma 2.3. *Let Ω be a Hintikka set of $\mathscr{L}$. Then for every formula A of $\mathscr{L}$, either $A \in \Omega$ holds or $\neg A \in \Omega$ holds.*

Proof. We prove the lemma by structural induction.

First, if A is an atomic formula, then according to (1) in Definition 2.13, either $A \in \Omega$ or $\neg A \in \Omega$. For A being a composite formula, we prove the lemma by structural induction on A as follows.

(1) Suppose that A is $\neg B$. According to the assumption of the structural induction, either $B \in \Omega$ or $\neg B \in \Omega$. By (2) in Definition 2.13, this amounts to either $\neg(\neg B) \in \Omega$ or $\neg B \in \Omega$.

(2) Suppose that A is $B \vee C$. According to the assumption of the structural induction, there are four possibilities for the formulas B and C: $B \in \Omega$ and $C \in \Omega$, $B \in \Omega$ and $\neg C \in \Omega$, $\neg B \in \Omega$ and $C \in \Omega$, and $\neg B \in \Omega$ and $\neg C \in \Omega$. According to the definition of the Hintikka set, the first three cases amount to either B or C being in Ω, i.e., $B \vee C \in \Omega$. The fourth case is $\neg B \in \Omega$ and $\neg C \in \Omega$. In this case, (3) in Definition 2.13 indicates that $\neg(B \vee C) \in \Omega$. Thus either $B \vee C \in \Omega$ holds, or $\neg(B \vee C) \in \Omega$ holds.

(3) Suppose that A is $B \wedge C$. According to the assumption of the structural induction, there are four possibilities for the formulas B and C: $B \in \Omega$ and $C \in \Omega$, $B \in \Omega$ and $\neg C \in \Omega$, $\neg B \in \Omega$ and $C \in \Omega$, and $\neg B \in \Omega$ and $\neg C \in \Omega$. According to the definition of the Hintikka set, the first case is $B \in \Omega$ and $C \in \Omega$, i.e., $B \wedge C \in \Omega$. The last three cases amount to either $\neg B$ or $\neg C$ being in Ω. In these cases, (4) in Definition 2.13 indicates that $\neg(B \wedge C) \in \Omega$. Thus either $B \wedge C \in \Omega$ holds, or $\neg(B \wedge C) \in \Omega$ holds.

(4) The proof for A being $B \to C$ is similar to that for A being $B \vee C$.

(5) Suppose that A is $\exists xB$. According to the assumption of the structural induction, for every $t \in H$, either $B[t/x] \in \Omega$ or $\neg B[t/x] \in \Omega$. If there exists a $t \in H$ such that $B[t/x] \in \Omega$, then according to (6) in Definition 2.13, $\exists xB \in \Omega$. If $\neg B[t/x] \in \Omega$ holds for every $t \in H$, then according to (6) in Definition 2.13, this means that $\neg\exists xB \in \Omega$ holds. Thus either $\exists xB \in \Omega$ holds, or $\neg\exists xB \in \Omega$ holds.

(6) Suppose that A is $\forall xB$. According to the assumption of the structural induction, for every $t \in H$, either $B[t/x] \in \Omega$ or $\neg B[t/x] \in \Omega$. If there exists a $t \in H$ such that $\neg B[t/x] \in \Omega$, then according to (7) in Definition 2.13, $\neg\forall xB \in \Omega$. If $B[t/x] \in \Omega$ holds for every $t \in H$, then according to (7) in Definition 2.13, this means that $\forall xB \in \Omega$ holds. Thus either $\forall xB \in \Omega$ holds, or $\neg\forall xB \in \Omega$ holds. □

2.9 Herbrand model

In this section we prove the satisfiability of the Hintikka set. According to Definition 2.9, we have to find a model $(\mathbf{M}, \sigma)$ such that every formula in the Hintikka set is true under $(\mathbf{M}, \sigma)$. This model is called the Herbrand model. The basic idea for constructing the Herbrand model is as follows. First of all, for a given first-order language $\mathscr{L}$, the Herbrand domain H of $\mathscr{L}$ is a set. It should be noted that even though each element of H is a term and a symbol string, yet it is still an element of the *set* H. Therefore we can use H as both the domain and the range to define the functions in the Herbrand model. We can then define propositions of $\mathbb{H}$ by assigning truth values to atomic formulas of $\mathscr{L}$, and further construct $\mathbb{H}$ for $\mathscr{L}$.

Definition 2.14 (Function of $\mathbb{H}$). Let H be a Herbrand domain and f be an arbitrary n-ary function symbol of $\mathscr{L}$. We call f_H an *n-ary function* of $\mathbb{H}$ if its domain is $H \times \cdots \times H$ and its range is H such that

$$f_H(t_1, \ldots, t_n) = ft_1 \cdots t_n.$$

Definition 2.14 indicates that f_H is an n-ary function of $\mathbb{H}$. When the values of the n variables of f_H are n elements $t_1, \ldots, t_n$ of H respectively, the value of the function f_H is also an element $ft_1 \cdots t_n$ of H.

Definition 2.15 (Proposition of $\mathbb{H}$). Let P be an n-ary predicate of $\mathscr{L}$. For n elements $t_1, \ldots, t_n$ of H, we define

$$P_H(t_1, \ldots, t_n) = Pt_1 \cdots t_n$$

and call P_H an *n-ary relation* or *atomic proposition* of $\mathbb{H}$.

A proposition of $\mathbb{H}$ is either an atomic proposition or a composite proposition composed of propositions connected by the logical connectives $\neg$, $\wedge$, $\vee$, $\to$, $\leftrightarrow$ and quantifiers $\forall$ and $\exists$.

For $\mathbb{H}$, the key is how to determine the truth of a proposition. According to Lemma 2.3, for every Hintikka set Ω and every formula A of $\mathscr{L}$, either $A \in \Omega$ holds or $\neg A \in \Omega$ holds. Thus we prescribe that every formula in the set Ω is a true proposition of $\mathbb{H}$.

Definition 2.16 (Truth of propositions of $\mathbb{H}$). Suppose that H is a Herbrand domain and Ω is a Hintikka set with respect to H. For each proposition A of $\mathbb{H}$, we define A being true if $A \in \Omega$. Thus $\mathbb{H}$ becomes a domain with respect to Ω and is denoted by $\mathbb{H}_\Omega$.

So far, we have defined the Herbrand domain H, functions, and propositions of $\mathbb{H}$. Having introduced the Hintikka set, we also defined the truth of propositions of $\mathbb{H}$, from which we obtained $\mathbb{H}_\Omega$. If we further define an interpretation map, then we obtain a model of $\mathscr{L}$ which is called the *Herbrand model* of $\mathscr{L}$ with respect to the Hintikka set Ω. The interpretation map can be defined by interpreting the constant symbols, function symbols and predicate symbols of the first-order language $\mathscr{L}$ as constants, functions and atomic propositions of $\mathbb{H}$ which are defined in Definitions 2.12, 2.14 and 2.15.

It should be mentioned that according to our previous convention, we shall not discriminate among $\mathbb{H}$, $\mathbb{H}_\Omega$ and H unless stated specifically. Now let us first define the Herbrand model which contains no variables.

Definition 2.17 (Herbrand model with respect to Ω). Suppose that $\mathscr{L}$ is a first-order language with H being its Herbrand domain and Ω being a Hintikka set of $\mathscr{L}$ with respect to H. I_H is an interpretation map from $\mathscr{L}$ to H and is defined as follows:

(1) $I_H(c) = c$;

(2) $I_H(f) = f_H$;

(3) $I_H(P) = P_H$.

We call (H, I_H) a *Herbrand structure* of $\mathscr{L}$ with respect to Ω and denote it as $\mathbf{H}$. Let $\sigma : V \to H$ be an arbitrary assignment of $\mathscr{L}$ with respect to the Herbrand domain H. We call $(\mathbf{H}, \sigma)$ a *Herbrand model* of $\mathscr{L}$ with respect to the Hintikka set Ω and denote it as $\mathbf{H}_\Omega$.

In Definition 2.17, the equation (1) defines the semantics of the constant symbol c. The symbol c on the left-hand side of the equation is a constant symbol, whereas c on the right-hand side is an element of the Herbrand domain H. The equation (1) represents that the constant symbol c is interpreted as an element c of H. The symbol f on the left-hand side of the equation (2) is a function symbol and f_H on the right-hand side is the interpretation of f. Note that f_H is a map that maps n elements $t_1, \ldots, t_n$ of H to the element $ft_1 \cdots t_n$ of H. Similarly, P on the left-hand side of the equation (3) is a predicate symbol and P_H on the right-hand side is the interpretation of P. P_H is a map that maps n elements $t_1, \ldots, t_n$ of H to the element $Pt_1 \cdots t_n$ of H. $Pt_1 \cdots t_n$ is an atomic proposition in the domain $\mathbb{H}$ that describes the relation between these elements. The truth of the relation depends on whether it belongs to the set Ω.

Lemma 2.4. *If a term $t \in H$, then $t_{\mathbf{H}_\Omega[\sigma]} = t$ holds.*

Proof. We prove the lemma by structural induction.

(1) If $t = c$, by the definition of the Herbrand domain, $c_{\mathbf{H}_\Omega[\sigma]} = c$.

(2) If $t = ft_1 \cdots t_n$, then

$$\begin{aligned}
&(ft_1 \cdots t_n)_{\mathbf{H}_\Omega[\sigma]} \\
&= f_H(t_{1\mathbf{H}_\Omega[\sigma]}, \ldots, t_{n\mathbf{H}_\Omega[\sigma]}) && \text{(by the definition of semantics of terms)} \\
&= f_H(t_1, \ldots, t_n) && \text{(by the induction hypothesis on } t_1, \ldots, t_n) \\
&= ft_1 \cdots t_n && \text{(by the definition of } f_H).
\end{aligned}$$

□

Lemma 2.5. *Let Ω be a Hintikka set of a first-order language $\mathscr{L}$. Then for every formula A,*

$$\mathbf{H}_\Omega \models_\sigma A \text{ holds if and only if } A \in \Omega \text{ holds.}$$

Proof. It suffices to prove that for every formula A, $A_{\mathbf{H}_\Omega[\sigma]} = \mathrm{T}$ if and only if $A \in \Omega$. We prove this lemma by mathematical induction on the rank $\mathrm{rk}(A)$ of the formula A.

If $\mathrm{rk}(A) = 1$, then A is an atomic formula. Let A be $Pt_1 \cdots t_n$ and we have

$$\begin{aligned}
& (Pt_1 \cdots t_n)_{\mathbf{H}_\Omega[\sigma]} = \mathrm{T} \\
\text{if and only if } & P_H(t_{1\mathbf{H}_\Omega[\sigma]}, \ldots, t_{n\mathbf{H}_\Omega[\sigma]}) = \mathrm{T} && \text{(by the semantics of predicates)} \\
\text{if and only if } & P_H(t_1, \ldots, t_n) = \mathrm{T} && \text{(by Lemma 2.4)} \\
\text{if and only if } & Pt_1 \cdots t_n \in \Omega \text{ holds} && \text{(by Definition 2.16).}
\end{aligned}$$

Suppose that the lemma applies to any formula A with $\mathrm{rk}(A) \leqslant k$. Consider the case where $\mathrm{rk}(A) = k+1$. In this case, A can only be one of $\neg B$, $B \vee C$, $B \wedge C$, $B \to C$, $B \leftrightarrow C$, $\forall xB$ and $\exists xB$ and the lemma holds with respect to the formulas B and C. Here it suffices to prove the lemma in the following cases.

(1) If A is $\neg B$, then

$$\begin{aligned}
& (\neg B)_{\mathbf{H}_\Omega[\sigma]} = \mathrm{T} \\
\text{if and only if } & B_{\mathbf{H}_\Omega[\sigma]} = \mathrm{F} && \text{(by the semantics of } \neg) \\
\text{if and only if } & B \notin \Omega && \text{(by the induction hypothesis)} \\
\text{if and only if } & \neg B \in \Omega \text{ holds} && \text{(by Lemma 2.3).}
\end{aligned}$$

(2) If A is $B \vee C$, then we have

$$\begin{aligned}
& (B \vee C)_{\mathbf{H}_\Omega[\sigma]} = \mathrm{T} \\
\text{if and only if } & \mathbf{B}_\vee(B_{\mathbf{H}_\Omega[\sigma]}, C_{\mathbf{H}_\Omega[\sigma]}) = \mathrm{T} && \text{(by the semantics of } \vee) \\
\text{if and only if } & B_{\mathbf{H}_\Omega[\sigma]} = \mathrm{T} \text{ or } C_{\mathbf{H}_\Omega[\sigma]} = \mathrm{T} && \text{(by the definition of } \mathbf{B}_\vee) \\
\text{if and only if } & B \in \Omega \text{ or } C \in \Omega \text{ holds} && \text{(by the induction hypothesis)} \\
\text{if and only if } & B \vee C \in \Omega \text{ holds} && \text{(by Definition 2.13 and Lemma 2.3).}
\end{aligned}$$

(3) If A is $\exists xB$, then according to the definition of the semantics of $\exists$, $(\exists xB)_{\mathbf{H}_\Omega[\sigma]} = \mathrm{T}$ holds if and only if there exists a $t \in H$ such that $B_{\mathbf{H}_\Omega[\sigma[x:=t]]} = \mathrm{T}$ holds.

It should be noted that the assignment $\sigma[x := t]$ assigns the value t to the variable x, and the term t of $\mathscr{L}$ is an element of the Herbrand domain. Hence the formula $B[t/x]$, which is obtained by substituting the term t for the variable x in B, is interpreted in $\mathbf{H}_\Omega$ as $B_{\mathbf{H}_\Omega[\sigma[x:=t]]}$. Thus $B_{\mathbf{H}_\Omega[\sigma[x:=t]]} = (B[t/x])_{\mathbf{H}_\Omega[\sigma]}$ holds.

Accordingly, $(\exists xB)_{\mathbf{H}_\Omega[\sigma]} = \mathrm{T}$ holds if and only if there exists a $t \in H$ such that $(B[t/x])_{\mathbf{H}_\Omega}[\sigma] = \mathrm{T}$ holds. Since $\mathrm{rk}(B[t/x]) = \mathrm{rk}(B) = k$, according to the assumption of the induction, there exists a $t \in H$ such that $(B[t/x])_{\mathbf{H}_\Omega}[\sigma] = \mathrm{T}$ holds if and only if $B[t/x] \in \Omega$ holds.

By Definition 2.13 and Lemma 2.3, there exists a $t \in H$ such that $B[t/x] \in \Omega$ holds if and only if $\exists xB \in \Omega$ holds. □

Theorem 2.1 (Satisfiability of Hintikka sets). *If Ω is a Hintikka set of a first-order language $\mathscr{L}$, then Ω is satisfiable and the Herbrand model $\mathbf{H}_\Omega$ of $\mathscr{L}$ is a model that satisfies Ω.*

Proof. The conclusion can be directly deduced from Lemma 2.5. □

Careful readers must have found that we did not discuss the semantics of the equality symbol $\doteq$ when the semantics and proofs of atomic formulas in Herbrand models were involved. In fact, to prove that Lemma 2.5 also applies to the equality symbol, more technical subtleties are involved. The reason is that, if the atomic formula $t_1 \doteq t_2$ belongs to Ω, then according to the semantics of $\doteq$ in Definition 2.8, $t_{1\mathbf{H}_\Omega[\sigma]} = t_{2\mathbf{H}_\Omega[\sigma]}$ should hold.

Nonetheless by Lemma 2.4, we also have $t_{1\mathbf{H}_\Omega[\sigma]} = t_1$ and $t_{2\mathbf{H}_\Omega[\sigma]} = t_2$. That is, we should have $t_1 = t_2$. Since H is a Herbrand domain whose elements are terms of $\mathscr{L}$, t_1 and t_2 should have the same syntactic structure.

We know that in elementary arithmetic we have $x_1 + Sx_2 \doteq S(x_1 + x_2)$, from which we can obtain $+S0S0 \doteq SS0$. Here $+S0S0$ and $SS0$ are terms (symbol strings) in different syntactic structure. Then how do we deal with the scenario that two elements in different syntactic forms are equal?

In mathematics, the solution is to introduce an equivalence relation and define equivalence classes according to the equivalence relation. Then we construct a domain whose elements are the equivalence classes. Specifically, we define $t_1 \sim t_2$ if $t_1 \doteq t_2 \in \Omega$ and prove that $\sim$ is an equivalence relation. As per this equivalence relation, we define equivalence classes $\bar{t} = \{t' \mid t \sim t'\}$ as well as the domain $\overline{\mathbb{H}}$ whose elements are the equivalence classes.

In this way the terms in different syntactic structures and satisfying $t \doteq t' \in \Omega$ are all represented by one element $\bar{t}$ in the domain $\overline{\mathbb{H}}$. Consequently, $\mathbf{H}_\Omega$ in Lemma 2.5 is changed into $\overline{\mathbf{H}}_\Omega$ and Lemma 2.5 becomes:

$$\overline{\mathbf{H}}_\Omega \models_\sigma A \text{ holds if and only if } A \in \Omega \text{ holds.}$$

2.10 Herbrand model with variables

In the domain of a Herbrand model $\mathbf{H}_\Omega$, every element is a Herbrand term of $\mathscr{L}$. They are terms of $\mathscr{L}$ with no variables. Recall that, as we discussed in the beginning of Section

2.9, symbol strings are also elements of a set. We can generalize the definition of the Herbrand model so that the model contains all the terms of $\mathscr{L}$, i.e., it contains the terms with free variables.

Generally speaking, constant symbols are interpreted as elements in a domain through an interpretation map I and variables are also interpreted as elements in the domain through an assignment σ. Since σ is a map as well, there is a lot in common between constant symbols and variables from the viewpoint of models.

Thus we make the following definitions:

For a language $\mathscr{L}$, we define a language $\mathscr{L}^+$ such that $\mathscr{L}_C^+ \supset \mathscr{L}_C$ and each variable x in $\mathscr{L}$ corresponds to a constant symbol c_x in the constant symbol set of $\mathscr{L}^+$ such that different variables of $\mathscr{L}$ correspond to different constant symbols of $\mathscr{L}^+$. Under this definition the constant symbol sets of the two first-order languages have the following relationship:

$$\mathscr{L}_C^+ = \mathscr{L}_C \cup \{c_x \mid x \in V \text{ is a variable of } \mathscr{L},\ c_x \notin \mathscr{L}_C\}.$$

For the language $\mathscr{L}^+$, we similarly define the Herbrand domain H^+ and the Hintikka set Ω^+. In this way we obtain the Herbrand model $\mathbf{H}_{\Omega^+}$ of $\mathscr{L}^+$ with respect to the Hintikka set Ω^+. This model can be transformed in the following way to a Herbrand model $\mathbf{H}^-_{\Omega^+}$ of $\mathscr{L}$, whose terms and formulas contain variables.

Definition 2.18 (t^+ and s^-). Let t be a term of a language $\mathscr{L}$. We define the Herbrand term t^+ of $\mathscr{L}^+$ as follows.

(1) If t is a constant symbol of $\mathscr{L}$, then we define t^+ as t.

(2) If t is a variable x, then we define t^+ as c_x.

(3) If t is $ft_1 \cdots t_n$, then we define t^+ as $ft_1^+ \cdots t_n^+$.

For any Herbrand term s of a language $\mathscr{L}^+$, we define the term s^- of $\mathscr{L}$ as follows.

(1) If s is a constant symbol of $\mathscr{L}^+$ and $s \in \mathscr{L}_C$, then we define s^- as s.

(2) If s is a constant symbol c_x of $\mathscr{L}^+$, then we define s^- as x, where x is a variable of $\mathscr{L}$.

(3) If s is $fs_1 \cdots s_n$, then we define s^- as $fs_1^- \cdots s_n^-$.

Lemma 2.6. *If t is a term of a first-order language $\mathscr{L}$ and s is a Herbrand term of the first-order language $\mathscr{L}^+$, then we have*

$$(t^+)^- = t, \quad (s^-)^+ = s.$$

Proof. The conclusion can be proved by structural induction. □

Corollary 2.1 (One-to-one correspondence). *There exists a one-to-one correspondence between the terms of a first-order language $\mathscr{L}$ and the Herbrand terms of the first-order language $\mathscr{L}^+$.*

Proof. The conclusion readily follows from Lemma 2.6. □

Definition 2.19 (A^+ and A^-). For every formula A of $\mathscr{L}$, we define the sentence A^+ of $\mathscr{L}^+$ in the following way.

(1) If A is $Pt_1\cdots t_n$, then $(Pt_1\cdots t_n)^+$ is the sentence $Pt_1^+\cdots t_n^+$.

(2) If A is $\neg B$, then $(\neg B)^+$ is the sentence $\neg B^+$.

(3) If A is $B\vee C$, then $(B\vee C)^+$ is the sentence $B^+\vee C^+$.

(4) If A is $B\wedge C$, then $(B\wedge C)^+$ is the sentence $B^+\wedge C^+$.

(5) If A is $B\to C$, then $(B\to C)^+$ is the sentence $B^+\to C^+$.

(6) If A is $\exists xB$, then $(\exists xB)^+$ is the sentence $\exists xB^+$. Here $^+$ only applies to the free variables in B that are not x.

(7) If A is $\forall xB$, then $(\forall xB)^+$ is the sentence $\forall xB^+$. Here $^+$ only applies to the free variables in B that are not x.

For every sentence A of $\mathscr{L}^+$, we define the formula A^- of $\mathscr{L}$ in the following way.

(1) If A is $Pt_1\cdots t_n$, then $(Pt_1\cdots t_n)^-$ is the formula $Pt_1^-\cdots t_n^-$.

(2) If A is $\neg B$, then $(\neg B)^-$ is the formula $\neg B^-$.

(3) If A is $B\vee C$, then $(B\vee C)^-$ is the formula $B^-\vee C^-$.

(4) If A is $B\wedge C$, then $(B\wedge C)^-$ is the formula $B^-\wedge C^-$.

(5) If A is $B\to C$, then $(B\to C)^-$ is the formula $B^-\to C^-$.

(6) If A is $\exists xB$, then $(\exists xB)^-$ is the formula $\exists xB^-$.

(7) If A is $\forall xB$, then $(\forall xB)^-$ is the formula $\forall xB^-$.

Similarly, we can prove the following lemma by structural induction.

Lemma 2.7. *If A and B are a formula and a sentence of $\mathscr{L}$ and $\mathscr{L}^+$ respectively, then we have*

$$(A^+)^- = A, \quad (B^-)^+ = B.$$

Definition 2.20. With respect to the Herbrand model $\mathbf{H}_{\Omega^+}$ of a language $\mathscr{L}^+$, we define the Herbrand model with free variables $\mathbf{H}^-_{\Omega^+}$ of $\mathscr{L}$ as follows.

(1) The domain of $\mathbf{H}^-_{\Omega^+}$ is composed of s^-, where s is a Herbrand term of the language $\mathscr{L}^+$.

(2) For every f, $(ft_1\cdots t_n)_{\mathbf{H}^-_{\Omega^+}} = ft_1\cdots t_n$.

(3) $(Pt_1\cdots t_n)_{\mathbf{H}^-_{\Omega^+}} = Pt_1\cdots t_n$.

Lemma 2.6 indicates that the domain of $\mathbf{H}^-_{\Omega^+}$ is actually the set of terms of $\mathscr{L}$.

Definition 2.21 (Semantics of the formulas in $\mathbf{H}^-_{\Omega^+}$). For every formula A in $\mathscr{L}$, $\mathbf{H}^-_{\Omega^+} \models A$ is defined to be held if $\mathbf{H}_{\Omega^+} \models A^+$ holds.

According to the Hintikka set Ω^+ of $\mathscr{L}^+$, we can define the formula set $(\Omega^+)^- = \{A^- \mid A \in \Omega^+\}$ of $\mathscr{L}$. Lemma 2.7 indicates that the following lemma holds.

Lemma 2.8. $\mathbf{H}^-_{\Omega^+} \models A$ *holds if and only if* $A \in (\Omega^+)^-$.

By Definition 2.19 and Lemmas 2.6 and 2.7, we can prove the following lemma by structural induction.

Lemma 2.9. *If the sentence set Ω^+ of a language $\mathscr{L}^+$ is a Hintikka set, then the formula set $(\Omega^+)^-$ of the language $\mathscr{L}$ is a Hintikka set as well.*

Using Lemma 2.8, we can prove the following theorem directly.

Theorem 2.2 (Satisfiability of Hintikka sets with variables). *Every Hintikka set that contains variables is satisfiable.*

We can define directly the Herbrand domain with variables, Hintikka set with variables and Herbrand model with variables. The Herbrand model with variables can be used to prove the satisfiability of the Hintikka set with variables. This is exactly what some other researchers did. In fact, it suffices for us to add the following in Definition 2.12: if x is a variable of $\mathscr{L}$, then $x \in H$. In this book we define the Herbrand model without variables and the Herbrand model with variables separately. There are two reasons for this: first, such definitions are easier for beginners; secondly, the Herbrand domains without variables are indispensable to the problem of inductive inference in Chapter 9.

2.11 Substitution lemma

In proving Lemma 2.5 in Section 2.9, in the case where the formula is $\exists xB$, we used the following property of a Herbrand model:

$$B_{\mathbf{H}_\Omega[\sigma[x:=t_{\mathbf{H}_\Omega[\sigma]}]]} = (B[t/x])_{\mathbf{H}_\Omega[\sigma]}.$$

Here $\sigma[x := t_{\mathbf{H}_\Omega[\sigma]}]$ on the left-hand side of the formula is an assignment, whereas $t_{\mathbf{H}_\Omega[\sigma]}$ in the formula is an element of H. The above equation indicates that, in the model $(\mathbf{H}_\Omega, \sigma)$, there are two different ways to interpret the formula $B[t/x]$. The first way is to find the interpretation of t in the model $(\mathbf{H}_\Omega, \sigma)$ first, and then interpret the formula B, where the free variable x in B is replaced by the interpretation of t directly. This is the implication of the left-hand side of the above equation.

The second interpretation is to substitute the term t for the free variable x in B first, and then interpret B. This is the implication of the right-hand side of the above equality. The equality indicates that the results of these two ways are the same. In this section we prove that this property holds for all first-order languages. This is the substitution lemma stated as follows.

Lemma 2.10 (Substitution lemma). *Let $\mathscr{L}$ be a first-order language with* $\mathbf{M}$ *and* σ *being a structure and an assignment of $\mathscr{L}$ respectively. Let t, t' and A be two terms and a formula of $\mathscr{L}$ respectively. Then the following equations hold:*

$$(t[t'/x])_{\mathbf{M}[\sigma]} = t_{\mathbf{M}[\sigma[x:=t'_{\mathbf{M}[\sigma]}]]},$$
$$(A[t/x])_{\mathbf{M}[\sigma]} = A_{\mathbf{M}[\sigma[x:=t_{\mathbf{M}[\sigma]}]]}.$$

It is worthwhile to note that when defining the semantics of formulas, we once pointed out that the symbol $[t/x]$ is a substitution operation of first-order languages. It is an operation of symbol strings according to syntactic rules. Nonetheless $\sigma[x := t_{\mathbf{M}[\sigma]}]$ is an assignment. It represents that the variable x is assigned the element $t_{\mathbf{M}[\sigma]}$ of the domain M as its value.

The difference between $A[t/x]$ and $A_{\mathbf{M}}[\sigma[x := t_{\mathbf{M}[\sigma]}]$ is that the former, a symbol string, is the formula A of a first-order language in which variable x is substituted by the term t (also a symbol string), whereas the latter, a truth value, is the interpretation of A in the domain in which x is assigned the value $t_{\mathbf{M}[\sigma]}$.

Let us take the second equality of the substitution lemma as an example to clarify the meaning of the lemma. $(A[t/x])_{\mathbf{M}[\sigma]}$ on the left-hand side of the equality represents that we first substitute the term t for the free variable x in the formula A and then interpret the formula in the model $(\mathbf{M},\sigma)$. $A_{\mathbf{M}[\sigma[x:=t_{\mathbf{M}[\sigma]}]]}$ on the right-hand side of the equality represents that we first interpret the term t and the formula A in the model $(\mathbf{M},\sigma)$ and then assign the interpretation of t to the free variable x in A to determine the truth of the proposition.

The second equation in the substitution lemma shows that these two approaches lead to the same result. According to Definitions 1.6 and 1.7, a substitution is a symbolic operation and may also be called the *substitution calculus*. The above lemma indicates that a substitution made by an interpretation is commutative with an interpretation followed by an evaluation. This commutativity ensures the rationality or soundness of the substitution calculus.

The proof of the substitution lemma is typical in that it not only is related to the substitutions of terms and formulas of first-order languages, which is a symbolic operation following syntactic rules, but also uses the concepts of interpretations, assignments, and models of first-order languages. The key is that the proof per se employs structural induction. A detailed proof of the lemma is provided in Appendix 2.

2.12 Theorem of isomorphism

We define isomorphic structures and models. In order to do so, we need to introduce the function operator denoted by $\circ$: Let function $f : V \to M$ and function $g : M \to H$. $g \circ f : V \to H$ is a function defined by $g \circ f(x) = g(f(x))$.

Definition 2.22 (Isomorphism and Isomorphic structure). *Let $(\mathbf{M}_1,\sigma_1)$ and $(\mathbf{M}_2,\sigma_2)$ be a model of first-order language $\mathscr{L}$. A map $\pi : \mathbf{M}_1 \to \mathbf{M}_2$ is called an isomorphism of $\mathbf{M}_1$ onto $\mathbf{M}_2$, which is written as $\pi : \mathbf{M}_1 \cong \mathbf{M}_2$ if*

(1) π *is a bijection of* $\mathbf{M}_1$ *onto* $\mathbf{M}_2$*,*

(2) if c *is a constant symbol of* $\mathscr{L}$*, then*

$$\pi(c_{\mathbf{M}_1}) = c_{\mathbf{M}_2},$$

(3) if f *is a* n*-ary function symbol of* $\mathscr{L}$ *and* $a_1, \ldots, a_n \in \mathbf{M}_1$*, then*

$$\pi(f_{\mathbf{M}_1}(a_1, \ldots, a_n)) = f_{\mathbf{M}_2}(\pi(a_1), \ldots, \pi(a_n)),$$

(4) if P *is a* n*-ary predicate symbol of* $\mathscr{L}$ *and* $a_1, \ldots, a_n \in \mathbf{M}_1$*, then*

$$\pi(P_{\mathbf{M}_1}(a_1, \ldots, a_n)) = P_{\mathbf{M}_2}(\pi(a_1), \ldots, \pi(a_n)),$$

(5) $\pi \circ \sigma_1 = \sigma_2$.

$\mathbf{M}_1$ *and* $\mathbf{M}_2$ *are said to be isomorphic with respect to* π*, which is written as* $\mathbf{M}_1 \cong \mathbf{M}_2$ *if there exists an isomorphism* $\pi : \mathbf{M}_1 \cong \mathbf{M}_2$.

Example 2.7. Consider the language of elementary arithmetic $\mathscr{A}$ defined in Example 1.1 and its structure $\mathbf{N} : (\mathbb{N}, I)$ defined in Example 2.1. Let $\mathbb{N}_{mod\ 2}$ be the set of even numbers and let the interpretation map $I_{mod\ 2}$ defined by

$$I_{mod\ 2}(x) = 2x.$$

It can be verified that $(\mathbb{N}_{mod\ 2}, I_{mod\ 2})$ is a structure of $\mathscr{A}$. Let us define the map π by $\pi(n) = 2n$. We can verify that $\pi : \mathbb{N} \to \mathbb{N}_{mod\ 2}$ is isomorphism and $\mathbf{N}$ and $\mathbf{N}_{mod\ 2}$ are isomorphic with respect to π.

For isomorphic structures, we have the following theorem.

Theorem 2.3. *For a given first-order language* $\mathscr{L}$*. If* $(\mathbf{M}_1, \sigma_1)$ *and* $(\mathbf{M}_2, \sigma_2)$ *are isomorphic models of* $\mathscr{L}$ *with respect to* $\pi : \mathbf{M}_1 \cong \mathbf{M}_2$*, then for any formula* A *in* $\mathscr{L}$*,*

$$\mathbf{M}_1 \models A \text{ if and only if } \mathbf{M}_2 \models A.$$

Proof. We prove this theorem by structural induction. Since $(\mathbf{M}_1, \sigma_1)$ and $(\mathbf{M}_2, \sigma_2)$ are isomorphic, the assignment $\sigma_2 = \pi \circ \sigma_1$. To prove the theorem, we should prove

1. for every term t, $t_{\mathbf{M}_1[\sigma_1]} = t_{\mathbf{M}_2[\sigma_2]}$ holds,
2. for every formula A, $\mathbf{M}_1 \models_{\sigma_1} A$ holds if and only if $\mathbf{M}_2 \models_{\sigma_2} A$ holds.

We can easily prove that the theorem holds for terms. Let us prove the second condition for a formula A. To do so, consider the cases of atomic formulas, and the composite formulas involving $\neg$, $\vee$, and $\exists$.

1. A is $t_1 \doteq t_2$.

$$\begin{array}{lll} & \mathbf{M}_1 \models_{\sigma_1} t_1 \doteq t_2 & \\ \iff & t_{1\mathbf{M}_1[\sigma_1]} = t_{2\mathbf{M}_1[\sigma_1]} & \text{by the semantics of } \doteq \\ \iff & \pi(t_{1\mathbf{M}_1[\sigma_1]}) = \pi(t_{2\mathbf{M}_1[\sigma_1]}) & \text{since } \pi \text{ is a bijection} \\ \iff & t_{1\mathbf{M}_2[\pi\circ\sigma_1]} = t_{2\mathbf{M}_2[\pi\circ\sigma_1]} & \text{by Definition 2.22} \\ \iff & \mathbf{M}_2 \models_{\pi\circ\sigma_1} t_1 \doteq t_2 & \text{by the definition of } \models \\ \iff & \mathbf{M}_2 \models_{\sigma_2} t_1 \doteq t_2 & \text{by } \sigma_2 = \pi\circ\sigma_1 \end{array}$$

2. A is $Pt_1\cdots t_n$.

$$\begin{array}{lll} & \mathbf{M}_1 \models_{\sigma_1} Pt_1\cdots t_n & \\ \iff & P_{\mathbf{M}_1[\sigma_1]} t_{1\mathbf{M}_1[\sigma_1]}\cdots t_{n\mathbf{M}_1[\sigma_1]} = \mathrm{T} & \text{by the semantics of } Pt_1\cdots t_n \\ \iff & \pi(P_{\mathbf{M}_1[\sigma_1]} t_{1\mathbf{M}_1[\sigma_1]}\cdots t_{n\mathbf{M}_1[\sigma_1]}) = \pi(\mathrm{T}) & \text{since } \pi \text{ is a bijection} \\ \iff & P_{\mathbf{M}_2[\pi\circ\sigma_1]} t_{1\mathbf{M}_2[\pi\circ\sigma_1]}\cdots t_{n\mathbf{M}_2[\pi\circ\sigma_1]} = \mathrm{T} & \text{by Definition 2.22} \\ \iff & \mathbf{M}_2 \models_{\pi\circ\sigma_1} Pt_1\cdots t_n & \text{by the definition of } \models \\ \iff & \mathbf{M}_2 \models_{\sigma_2} Pt_1\cdots t_n & \text{by } \sigma_2 = \pi\circ\sigma_1 \end{array}$$

3. A is $\neg B$.

$$\begin{array}{lll} & \mathbf{M}_1 \models_{\sigma_1} \neg B & \\ \iff & B_{\mathbf{M}_1[\sigma_1]} = \mathrm{F} & \text{by the semantics of } \neg \\ \iff & \pi(B_{\mathbf{M}_1[\sigma_1]}) = \pi(\mathrm{F}) & \text{since } \pi \text{ is a bijection} \\ \iff & B_{\mathbf{M}_2[\pi\circ\sigma_1]} = \mathrm{F} & \text{by induction hypothesis} \\ \iff & \mathbf{M}_2 \models_{\pi\circ\sigma_1} \neg B & \text{by the definition of } \models \\ \iff & \mathbf{M}_2 \models_{\sigma_2} \neg B & \text{by } \sigma_2 = \pi\circ\sigma_1 \end{array}$$

4. A is $\exists xB$.

$$\begin{array}{lll} & \mathbf{M}_1 \models_{\sigma_1} \exists xB & \\ \iff & \text{there exists an } a\in\mathbf{M}_1 \text{ such that } B_{\mathbf{M}_1[\sigma_1[x:=a]]} = \mathrm{T} & \text{by the semantics of } \exists \\ \iff & \text{there exists an } a\in\mathbf{M}_1 \text{ such that } \pi(B_{\mathbf{M}_1[\sigma_1[x:=a]]}) = \pi(\mathrm{T}) & \text{since } \pi \text{ is a bijection} \\ \iff & \text{there exists an } a\in\mathbf{M}_1 \text{ such that } B_{\mathbf{M}_2[\pi\circ\sigma_1[x:=a]]} = \mathrm{T} & \text{by induction hypothesis} \\ \iff & (\exists xB)_{\mathbf{M}_2[\pi\circ\sigma_1[x:=a]]} = \mathrm{T} & \text{by the semantics of } \exists \\ \iff & \mathbf{M}_2 \models_{\pi\circ\sigma_1} \exists xB & \text{by the definition of } \models \\ \iff & \mathbf{M}_2 \models_{\sigma_2} \exists xB & \text{by } \sigma_2 = \pi\circ\sigma_1 \end{array}$$

□

Corollary 2.2. *If* $\mathbf{M}_1$ *and* $\mathbf{M}_2$ *are isomorphic with respect to the isomorphism* $\pi : \mathbf{M}_1 \cong \mathbf{M}_2$ *and* A *is a formula of* $\mathscr{L}$ *and* $a \in \mathbf{M}_1$*, then*

$$\mathbf{M}_1 \models_\sigma A[a/x] \quad \textit{if and only if} \quad \mathbf{M}_2 \models_{\pi\circ\sigma} A[a/x].$$

The following important theorem is also a direct corollary.

Theorem 2.4. *Let* $\mathbf{M}$ *be a model of first-order language* $\mathscr{L}$*. There exists a Herbrand model* $\mathbf{H}_\Omega$ *with respect to a Hintikka set* Ω *of* $\mathscr{L}$ *such that* $\mathbf{M}$ *and* $\mathbf{H}_\Omega$ *are isomorphic.*

Chapter 3

Formal Inference Systems

In Chapter 2, we introduced the concept of logical consequence. We call a formula A a *logical consequence* of a formula set Γ if, in every model that satisfies Γ, A is also satisfied. The definition of logical consequence refers to the logical semantics of formulas. Essentially, it means that, however we interpret the terms in the formulas, the truth of A always follows the truth of Γ. This is consistent with our normal understanding of the concept of logical consequence.

Consider plane geometry as an example. If we use Γ to denote the set of Euclidean postulates and A to denote the proposition that "*the sum of the interior angles of any triangle is equal to* 180°," then A is a logical consequence of Γ. This means that, no matter how we interpret the concepts of plane geometry, such as points and lines, as long as they obey the Euclidean postulates, the sum of the interior angles of any triangle is always equal to 180°.

According to Definition 2.11, to determine that A is a logical consequence of Γ, it is necessary to verify that, for all structures and assignments, if Γ is true, then so is A. However, mathematicians use a different method, i.e., they employ mathematical proofs or logical deductions to achieve the same goal.

In plane geometry every theorem is a *proved consequence* of the postulates (which are assumed) rather than a logical consequence verified by considering all models. The statement "*the sum of the interior angles of any triangle equals to* 180°" is a theorem which is proved mathematically, or is deduced from the set of postulates by logical inference rules. The proof itself depends only on Euclidean postulates and is enough to guarantee that the theorem is true for every triangle. We do not need to verify it by checking all triangles.

We generally assume that *in any theory of mathematics or natural science, every proposition that has been deduced logically or proved mathematically is also a logical consequence of the theory in the above sense*.

If a proposition has been deduced logically from the postulates, it is said to be a *proved consequence* of them. The equivalence of proved consequences and logical consequences is a basic assumption in mathematics and natural science. With this assumption, mathematical proof becomes a powerful approach for both determining logical consequences and avoiding verifying all possible applications.

This chapter shows that the above assumption is valid for first-order languages. For this purpose, we shall define what *proved consequence* means in first-order languages, and then prove that the basic assumption holds for first-order languages.

To do so, let us first take a theorem about triangles in plane geometry as an example to analyze the structure of mathematical proof.

(1) Mathematical Proof. Every mathematical theory starts with a set of propositions which is usually called its *axiom system*. These are the basis of lemmas, theorems and all proved consequences of the theory. For plane geometry, these are the set of Euclidian postulates. For classical mechanics, we could say that its axiom system contains Newton's three laws of motion, the principle of relativity and the law of universal gravity. An axiom system may be described by a formula set Γ. A proved consequence may be described by a formula. To denote the relationship between axioms and proved consequences, we introduce the symbol $\vdash$, read as "derives" or "deduces", and introduce the form $\Gamma \vdash A$ called a *sequent*, where Γ represents the axioms and A represents the proposition to be proved. If A is deduced from Γ or proved under Γ, then we say that the sequent $\Gamma \vdash A$ is *provable*. Otherwise, we say that $\Gamma \vdash A$ does not hold or is not provable.

(2) Structure of Mathematical Proofs. A proof of a theorem in mathematics is a piece of text containing a series of arguments. In each proof, the axioms, lemmas and theorems proved in previous arguments form the premises of the next step and the proposition to be proved is the conclusion. Like the overall structure of the proof, the internal structure of each step also consists of previously proved premises and a conclusion.

In what follows, we will analyze a simple proof in geometry. We will use P to denote the proposition "a polygon is a triangle" and Q to denote the proposition "the sum of interior angles of a polygon is equal to 180°." Then, the theorem "if a polygon is a triangle, then the sum of its interior angles is equal to 180°" can be written $P \to Q$, whereas its converse-negative proposition "if the sum of interior angles of a polygon is not equal to 180°, then this polygon is not a triangle" can be written $\neg Q \to \neg P$. Suppose that the former theorem is proved. The proof of the converse-negative proposition is given in the following four steps:

1. The premise is $P \to Q$ and the proposition to be proved is $\neg Q \to \neg P$. For simplicity, we can write this as a sequent. This means we have to prove

$$P \to Q \vdash \neg Q \to \neg P$$

holds.

2. Recall the semantics of $\to$ given in Chapter 2: to prove that $\neg Q \to \neg P$ holds, it suffices to prove that if both $P \to Q$ and $\neg Q$ hold, then $\neg P$ holds. Thus, the new premise becomes $P \to Q$ and $\neg Q$, and the goal of proof becomes $\neg P$. Written in the form of a sequent, this amounts to proving that

$$P \to Q, \neg Q \vdash \neg P$$

holds.

3. If $P \to Q$ occurs in the premise, it is presupposed to hold. According to the semantics of $\to$, this means $\neg P$ holds or Q holds. Therefore, as long as it is proved that $\neg P$ holds in either case, the converse-negative proposition is proved. Expressing this as a sequent, this is equivalent to proving that the following two sequents

$$\neg P, \neg Q \vdash \neg P \text{ and } Q, \neg Q \vdash \neg P$$

hold.

4. For the left-hand sequent: since its conclusion $\neg P$ occurs in the premise, which is presupposed to hold, the whole sequent holds. For the right-hand sequent: the premise contains both Q and $\neg Q$, which implies that the premise is contradictory, then any conclusion can be deduced. Therefore, this sequent also holds.

So the converse-negative proposition $\neg Q \to \neg P$ is proved.

Paragraphs 1 to 4 are the proof of the theorem, presented in the form of sequents.

(3) Logical inferences are the calculus of logical connective symbols. Let us analyze the composition of the above proof.

First of all, if P denotes "two corresponding sides of two triangles are equal and the angles included by these two sides are also equal" and Q denotes "two triangles are congruent," then $P \to Q$ denotes the theorem of congruence of triangles. Its converse-negative proposition is "if two triangles are not congruent, then these two triangles have at least one of two corresponding sides which are not equal or whose included angles are not equal," i.e., $\neg Q \to \neg P$.

If the proof of this converse-negative proposition is written out in full, we shall see that its structure is exactly the same as that for the proof given in (2).

As another example, if we use P to denote "interior alternate angles are equal" and Q to denote "two lines are parallel," then $\neg Q \to \neg P$, i.e., "if two lines are not parallel, then interior alternate angles are not equal" is a proved consequence. Its converse-negative proposition is "if interior alternate angles are equal, then two lines are parallel," i.e., $P \to Q$. Again, the structure of its full proof is the same as that in (2).

This indicates the following:

1. The proof of the converse-negative propositions does not depend on what P and Q denote.
2. The proof only depends on the logical connective symbols in the premise and the goal to be proved.
3. Every step in the proof is an operation on the logical connective symbols. In this case, these are $\to$ or $\neg$.

For instance, the second paragraph in (2) says that to prove that $P \to Q \vdash \neg Q \to \neg P$ holds, it suffices to prove that $P \to Q, \neg Q \vdash \neg P$ holds. It can be described by the following fraction:

$$\frac{P \to Q, \neg Q \vdash \neg P}{P \to Q \vdash \neg Q \to \neg P}.$$

This fraction describes an operation on the $\to$ on the right of $\vdash$ in the denominator: split $\neg Q \to \neg P$ on the right of the symbol $\vdash$ in the denominator sequent, move $\neg Q$ to the left of the symbol $\vdash$, keep $\neg P$ on the right of $\vdash$, and finally take the new sequent $P \to Q, \neg Q \vdash \neg P$ obtained from the operation as the numerator of the fraction.

This fraction is usually called an inference rule of $\to$, and reads as: $\neg Q \to \neg P$ can be deduced from $P \to Q$ if and only if $\neg P$ can be deduced from $P \to Q$ and $\neg Q$.

Similarly, the goal of paragraph 3 is to prove that $P \to Q, \neg Q \vdash \neg P$ holds. As we saw in paragraph 3, $P \to Q$ in the premise holds if and only if $\neg P$ holds or Q holds. Therefore, it suffices to prove that $\neg P$ can be deduced from the premise, no matter whether $\neg P$ holds

or Q holds. In other words, it suffices to prove that $P \to Q, \neg Q \vdash \neg P$ can be proved when both $\neg P, \neg Q \vdash \neg P$ and $Q, \neg Q \vdash \neg P$ hold. This inference step can be represented by the following fraction:

$$\frac{\neg P, \neg Q \vdash \neg P \qquad Q, \neg Q \vdash \neg P}{P \to Q, \neg Q \vdash \neg P}.$$

This fraction can also be viewed as an operation for $\to$ occurring on the left of $\vdash$ in the denominator. The operation deletes $\to$ from the left of $\vdash$ and produces two new sequents $\neg P, \neg Q \vdash \neg P$ and $Q, \neg Q \vdash \neg P$. This fraction reads as: $P \to Q, \neg Q \vdash \neg P$ holds if and only if both $\neg P, \neg Q \vdash \neg P$ and $Q, \neg Q \vdash \neg P$ hold.

We can see from these examples that the operation described by the fractions can be done in a mechanical way. These fractions can be viewed as rules in a symbolic calculus.

It can be seen from the above discussion that every step in a mathematical proof is an operation on some logical connective occurring in its premises or conclusion.

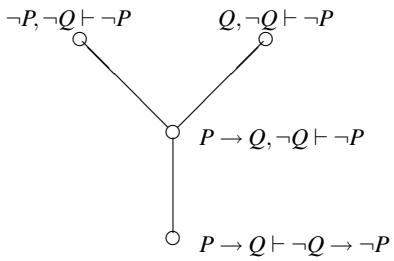
Such an operation can be described precisely, within the syntax of first-order languages, by defining rules of symbolic calculation on the logical connective symbols and quantifier symbols in the sequent. These rules are called formal inference rules and each rule can be written as a fraction.

The numerator and denominator of the fraction are composed of sequents. The form of each fraction is determined by the semantics of a logical connective symbol or a quantifier symbol. Each fraction expresses the statement: its numerator holds if and only its denominator holds. If a logical connective symbol, such as $\to$, appears on the left of $\vdash$ in the denominator sequent, this means that it appears in the premise of the proof; if it appears on the right of $\vdash$, this means that it appears in a conclusion of the proof.

The set of all formal logical inference rules make up a logical inference system.

(4) Mathematical proofs are trees composed of inference rules. If we represent each step of the proof given in (2) with a corresponding fraction and connect all the fractions together, then the proof of the converse-negative proposition has the following tree structure:

$$\cfrac{\cfrac{\neg P, \neg Q \vdash \neg P \qquad Q, \neg Q \vdash \neg P}{P \to Q, \neg Q \vdash \neg P}}{P \to Q \vdash \neg Q \to \neg P}$$



The root of this tree is at the bottom and the leaves are at the top. Associated with each node of the tree is a sequent. The root node $P \to Q \vdash \neg Q \to \neg P$ is the goal to be proved, i.e., the conclusion of proof or proved consequence. Each node of the tree and the nodes of its next layer follow an inference rule for some logical connective symbol. The

sequents $\neg P, \neg Q \vdash \neg P$ and $Q, \neg Q \vdash \neg P$ are in the leaf nodes of the tree. If all the sequents in the leaf nodes hold, then the tree structure represents a valid proof.

(5) The soundness and completeness of inference systems. At the beginning of this chapter, we stated the basic assumption of mathematics, that "proved consequences are logical consequences". In other words, if a proposition has been proved, then it holds and is applicable in any circumstances in which the set of axioms of the theory is also satisfied.

Having introduced the above concept of sequent, this basic assumption can be expressed as:

$$\text{if } \Gamma \vdash A \text{ is provable, then } \Gamma \models A \text{ holds.}$$

If this statement is true of a formal inference system, it is called *sound*. The complementary principle can be expressed as:

$$\text{if } \Gamma \models A \text{ holds, then } \Gamma \vdash A \text{ is provable.}$$

and an inference system for which this holds is called *complete*.

In this chapter, we prove that first-order languages are both sound and complete. More precisely, if a formal inference system is defined for first-order languages as a symbolic calculus on all logical connective symbols and quantifier symbols, then the system is both sound and complete.

Section 3.1 will provide a set of inference rules for logical connective symbols and quantifier symbols, called the **G** *inference system*, or **G** *system* for short. This system is a modified version of the formal inference system proposed by Gentzen [1969], while we use the symbol system of Gallier [1986]. In Section 3.2, the concepts of inference tree and proof tree will be introduced and some examples about inference trees and proof trees will be given. In Sections 3.3, 3.4, and 3.5, we will define the concepts of soundness, compactness, consistency, and completeness for formal inference systems of first-order languages, and prove that the **G** system is both sound and complete. In Section 3.6 it will be proved that most of the logical inference rules frequently used in mathematics and natural science are all derived rules of the **G** system. The basic concepts introduced in the first three chapters will be summarized in Section 3.7. The content of this chapter is usually referred to as the *proof theory* of first-order languages.

3.1 G inference system

In this section we introduce a formal inference system, called the **G** *inference system*, or the **G** *system* for short. We choose the **G** inference system in this book because its inference rules are both simple and symmetric. Furthermore, the connections between the inference rules and the semantics of the logical connective symbols and quantifier symbols in the rules are quite intuitive.

The **G** system is composed of axioms and inference rules with sequents being its basic objects. For each logical connective symbol or quantifier symbol, there are two inference rules in the **G** system: the left rule and the right rule. Each inference rule is

a fraction whose denominator is a sequent, called the *conclusion* of the rule, and whose numerator comprises one or two sequents, called the *premise* of the rule. The rule reads as: the sequent in its denominator is provable if and only if each sequent in its numerator is provable.

Before introducing the **G** inference system, we shall explain the symbols that will be used hereafter. In the previous two chapters we used the uppercase Greek letters Γ, Δ, Λ and Θ to denote finite sets of logical formulas including the empty set. For instance, Γ denotes the set $\{A_1,\ldots,A_m\}$ and Δ the set $\{B_1,\ldots,B_n\}$. In a sequent it is more convenient to write formula sets as formula sequences. For example, the formula set $\{A_1,\ldots,A_m\}$ can be simply written as $A_1,\cdots,A_m$ in a sequent. Thus we use Γ,A,Δ to denote $A_1,\ldots,A_m,A,B_1,\ldots,B_n$. Sequences such as A,Γ,Δ and Γ,Δ,A denote finite formula sets.

In what follows we introduce the inference rules of the **G** system, or **G** *rules* for short. Their meanings will be specified in Section 3.3. Let $\Gamma,\Delta,\Lambda,\Theta$ be finite formula sets with A and B being formulas in the following definitions.

Definition 3.1 (Sequent). The form

$$\Gamma \vdash \Delta$$

is called a *sequent* with Γ as its *antecedent* and Δ as its *succedent*.

Definition 3.2 (Axiom). The sequent

$$\Gamma,A,\Delta \vdash \Lambda,A,\Theta$$

is called the **G** *axiom*.

It says that because the descendent to be proved contains at least one formula in the antecedent, the axiom sequent is self-evident.

Definition 3.3 ($\neg$-rules).

$$\neg\text{-}L:\ \frac{\Gamma,\Delta \vdash A,\Lambda}{\Gamma,\neg A,\Delta \vdash \Lambda} \qquad\qquad \neg\text{-}R:\ \frac{A,\Gamma \vdash \Lambda,\Theta}{\Gamma \vdash \Lambda,\neg A,\Theta}$$

The $\neg$-L rule indicates that in order to prove $\Gamma,\neg A,\Delta \vdash \Lambda$ holds, we have to prove $\Gamma,\Delta \vdash A,\Lambda$ holds, and vice versa. The $\neg$-R rule indicates that in order to prove $\Gamma \vdash \Lambda,\neg A,\Theta$ holds, we have to prove $A,\Gamma \vdash \Lambda,\Theta$ holds, and vice versa.

Definition 3.4 ($\vee$-rules).

$$\vee\text{-}L:\ \frac{\Gamma,A,\Delta \vdash \Lambda \qquad \Gamma,B,\Delta \vdash \Lambda}{\Gamma,A\vee B,\Delta \vdash \Lambda} \qquad\qquad \vee\text{-}R:\ \frac{\Gamma \vdash \Lambda,A,B,\Theta}{\Gamma \vdash \Lambda,A\vee B,\Theta}$$

The $\vee$-L rule indicates that, to prove that the sequent $\Gamma,A\vee B,\Delta \vdash \Lambda$ holds, we have to prove that both $\Gamma,A,\Delta \vdash \Lambda$ and $\Gamma,B,\Delta \vdash \Lambda$ hold, and vice versa. The $\vee$-R rule can be interpreted in a similar way. The following rules of the **G** system on the logical connective symbols and quantifier symbols can all be interpreted in this way and we will not elaborate on them.

Definition 3.5 ($\wedge$-rules).

$$\wedge\text{-}L: \frac{\Gamma, A, B, \Delta \vdash \Lambda}{\Gamma, A \wedge B, \Delta \vdash \Lambda} \qquad \wedge\text{-}R: \frac{\Gamma \vdash \Lambda, A, \Theta \qquad \Gamma \vdash \Lambda, B, \Theta}{\Gamma \vdash \Lambda, A \wedge B, \Theta}$$

Definition 3.6 ($\rightarrow$-rules).

$$\rightarrow\text{-}L: \frac{\Gamma, \Delta \vdash A, \Lambda \qquad B, \Gamma, \Delta \vdash \Lambda}{\Gamma, A \rightarrow B, \Delta \vdash \Lambda} \qquad \rightarrow\text{-}R: \frac{A, \Gamma \vdash B, \Lambda, \Theta}{\Gamma \vdash \Lambda, A \rightarrow B, \Theta}$$

Definition 3.7 ($\forall$-rules).

$$\forall\text{-}L: \frac{\Gamma, A[t/x], \forall x A(x), \Delta \vdash \Lambda}{\Gamma, \forall x A(x), \Delta \vdash \Lambda} \qquad \forall\text{-}R: \frac{\Gamma \vdash \Lambda, A[y/x], \Theta}{\Gamma \vdash \Lambda, \forall x A(x), \Theta}$$

The $\forall$-*L* rule indicates that if the sequent $\Gamma, \forall x A(x), \Delta \vdash \Lambda$ holds, then the sequent $\Gamma, A[t/x], \forall x A(x), \Delta \vdash \Lambda$ holds, and vice versa. The $\forall$-*R* rule can be interpreted similarly.

The function of $A[t/x]$ in the numerator of the $\forall$-*L* rule is to eliminate the quantifier symbol $\forall$. The term t in the rules should match a term in a formula on the right-hand side of $\vdash$ so as to form an instance of the axiom. Note that we retain the $\forall x A(x)$ in the numerator so that we may use it later to construct another axiom instance. This is also true of the $\exists$-*R* rule below.

Definition 3.8 ($\exists$-rules).

$$\exists\text{-}L: \frac{\Gamma, A[y/x], \Delta \vdash \Lambda}{\Gamma, \exists x A(x), \Delta \vdash \Lambda} \qquad \exists\text{-}R: \frac{\Gamma \vdash \Lambda, A[t/x], \exists x A(x), \Theta}{\Gamma \vdash \Lambda, \exists x A(x), \Theta}$$

In the $\forall$-*R* and $\exists$-*L* rules, the variable y is either x or an eigen-variable with "eigen" referring to y not occurring free in Γ, Λ, Δ and A.

The formulas $A \wedge B$, $A \vee B$, $A \rightarrow B$, $\neg A$, $\forall x A(x)$, and $\exists x A(x)$ in the denominators of the above rules are called the *principal formulas* of the rules, whereas the formulas A, B, $A[t/x]$, and $A[y/x]$ in the numerators of the above rules are called their *side formulas* [Gallier, 1986]. The principal formulas of a **G** rule are those formulas in its denominator that are to be decomposed, whereas the side formulas are those formulas introduced in its numerator.

The $\wedge$-*L* and $\vee$-*R* rules indicate that in the sequent $A_1, \ldots, A_m \vdash B_1, \ldots, B_n$, the commas on the left-hand side of $\vdash$ can be regarded as the logical connective symbol $\wedge$, whereas the commas on the right-hand side of $\vdash$ can be regarded as the logical connective symbol $\vee$.

Strictly speaking, the formulas A and B in each inference fraction are not formulas in the first-order language. Instead, they are a kind of variables, which can be replaced by formulas instead of terms. Hence they should be called *formula variables*. Thus each **G** rule is an inference schema, which can also be called a **G** *rule schema*. When applying these inference schemas, A and B must be replaced by formulas of first-order languages, whereas the sets Γ, Δ, Λ, and Θ must be replaced by formula sets. After these substitutions, an inference fraction becomes an instance of the corresponding rule schema.

Definition 3.9 (Instances of inference rules). Let $\mathscr{L}$ be a first-order language. After substituting formula variables A, B and set variables Γ, Δ in a **G** inference rule by formulas and formula sets of $\mathscr{L}$, we call the inference fraction obtained an *instance of the inference rule*.

Lemma 3.1. *The cut rule*

$$\frac{\Gamma \vdash A, \Lambda \quad \Delta, A \vdash \Theta}{\Gamma, \Delta \vdash \Lambda, \Theta}$$

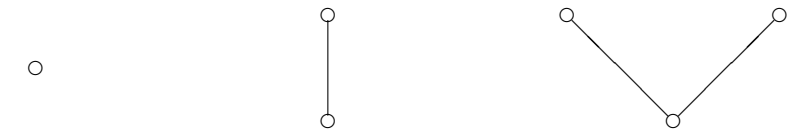
holds.

Proof. The cut rule can be deduced from the **G** inference system. The proof of the lemma is rather long and tedious and readers may refer to the proof of [Gallier, 1986]. □

The cut rule shows that if $\Gamma \vdash A, \Lambda$ and $\Delta, A \vdash \Theta$ are provable, then $\Gamma, \Delta \vdash \Lambda, \Theta$ is also provable. This lemma indicates that a proof using the cut rule can be substituted by a proof using the **G** rules only. In this sense the cut rule serves as a procedure or function in programming. The cut rule is treated in some books (e.g., [Gallier, 1986]) as a rule of the formal inference system of first-order languages. In this book we shall also take this approach and treat the cut rule as a rule of the **G** system. This will simplify the proof of the completeness of the **G** system.

3.2 Inference trees, proof trees and provable sequents

Each **G** inference rule introduced in the previous section can be represented as a tree. The nodes of the tree are sequents with its root being the conclusion and leaves being the premises of the inference rule. If a rule has only one premise, then it is a single branch tree; if the rule has two premises, then it is a tree with two branches. An axiom itself is a single node tree that is the leaf of the tree as well.



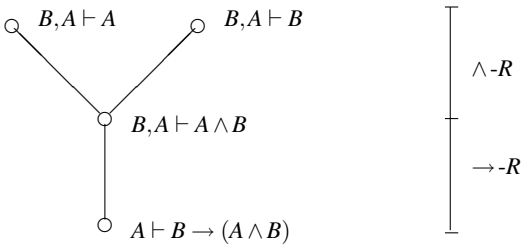
Axiom A rule with a single premise A rule with double premises

The following is an example of obtaining a proof tree via **G** rules.

Example 3.1 (Provable sequent). We prove that the sequent $A \vdash B \to (A \wedge B)$ holds. First we apply the $\to$-*R* rule to $\to$ on the right-hand side of $\vdash$ in the sequent. Then we apply the $\wedge$-*R* rule to $\wedge$ on the right-hand side to obtain the following proof tree:

$$\dfrac{\dfrac{B, A \vdash A \qquad B, A \vdash B}{B, A \vdash A \wedge B}\,(2)}{A \vdash B \to (A \wedge B)}\,(1)$$

Here the fraction (1) is obtained by applying the $\rightarrow$-*R* rule to $\rightarrow$ in $A \vdash B \rightarrow (A \wedge B)$ and the fraction (2) is obtained by applying the $\wedge$-*R* rule to $\wedge$ in $B, A \vdash A \wedge B$. The proof can be represented as the following proof tree:



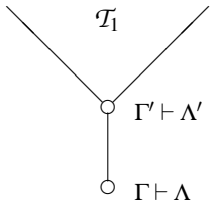
$A \vdash B \rightarrow (A \wedge B)$ holds because both $B, A \vdash A$ and $B, A \vdash B$ are instances of the axiom.

Generally speaking, to prove that any given $\Gamma \vdash A$ holds, we need to delete the logical connective symbols and quantifier symbols in Γ and A one by one via **G** inference rules and construct a tree from the root to the leaves. Each node of the tree is a sequent and each of its subtrees has either a single branch or two branches, as instances of inference rules with a single or double premises respectively. If every leaf of the tree is an instance of the **G** axiom, then $\Gamma \vdash A$ holds. In the course of proof, the **G** rules are only relevant to the logical connective symbols and quantifier symbols in Γ and A and are 'mechanical' operations on these symbols. As a result, proofs of this kind are called *formal proofs*. The following are formal definitions of inference trees, proof trees, and formal proofs.

Definition 3.10 (Inference trees, proof trees and formal proofs). For a given sequent $\Gamma \vdash \Lambda$, a tree $\mathcal{T}$ is called the *inference tree* of $\Gamma \vdash \Lambda$ if each node of $\mathcal{T}$ is a sequent and satisfies the following three conditions.

(1) A single node tree is an inference tree if its node is an instance of a sequent.

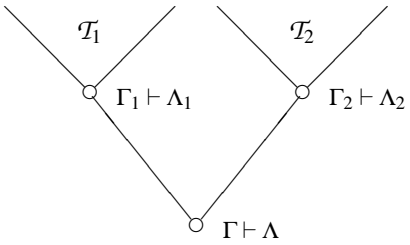
(2) If $\mathcal{T}_1$ is an inference tree whose root is a sequent $\Gamma' \vdash \Lambda'$, then the tree structure



$$\frac{\Gamma' \vdash \Lambda'}{\Gamma \vdash \Lambda}\ (a)$$

is an inference tree of $\Gamma \vdash \Lambda$ if and only if the fraction (a) is an instance of some rule of the **G** system.

(3) If $\mathcal{T}_1, \mathcal{T}_2$ are inference trees and the sequents of their roots are $\Gamma_1 \vdash \Lambda_1$ and $\Gamma_2 \vdash \Lambda_2$ respectively, then the tree



$$\frac{\Gamma_1 \vdash \Lambda_1 \qquad \Gamma_2 \vdash \Lambda_2}{\Gamma \vdash \Lambda}\ (b)$$

is an inference tree of $\Gamma \vdash \Lambda$ if and only if the fraction (b) is an instance of some rule of the **G** system.

If $\mathcal{T}$ is a finite inference tree of a sequent $\Gamma \vdash \Lambda$ whose leaves are all instances of the **G** axiom, then we say that $\mathcal{T}$ is a *proof tree* of $\Gamma \vdash \Lambda$.

A sequent is called *provable* if its proof tree exists. The proof tree is called a *formal proof* of the sequent and Λ is called the *formal consequence* of Γ. Otherwise, the sequent $\Gamma \vdash \Lambda$ is *unprovable*.

As per Definition 3.10, if the inference tree of $\Gamma \vdash \Lambda$ exists but one of its leaves is neither decomposable nor an instance of the axiom, then $\Gamma \vdash \Lambda$ is unprovable.

Example 3.2 (Inference trees). Consider the sequent $\neg P \to Q \vdash S \to R$. Its inference tree is as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{P,S \vdash R}{P \vdash S \to R}\,(3)}{\vdash \neg P, S \to R}\,(2) \qquad \dfrac{\dfrac{Q,S \vdash R}{Q \vdash S \to R}\,(4)}{}}{\neg P \to Q \vdash S \to R}\,(1)}{}$$

The fraction (1) is obtained by applying the $\to$-L rule to the denominator $\neg P \to Q \vdash S \to R$ with two sequents in its numerator: $\vdash \neg P, S \to R$ and $Q \vdash \neg P, S \to R$. Applying the $\neg$-R rule to $\vdash \neg P, S \to R$ we obtain the fraction (2). A further application of the $\to$-R rule to the numerator $P \vdash S \to R$ of the fraction (2) leads to $P, S \vdash R$, which yields the fraction (3). We apply the $\to$-R rule to $\to$ on the right-hand side of $\vdash$ in the sequent $Q \vdash S \to R$ to obtain $Q, S \vdash R$, which is the fraction (4). As per Definition 3.10, this tree is an inference tree. Since neither of its leaves $P, S \vdash R$ and $Q, S \vdash R$ is an instance of the axiom, this tree is an inference tree but not a proof tree.

Example 3.3 (Applications of the $\forall$-rules and $\exists$-rules). Consider the sequent $\exists x P(x) \wedge Q(a) \vdash \forall y P(f(y))$. Here a is a constant symbol and f a unary function symbol. P and Q

are unary predicate symbols. The inference tree of the sequent is:

$$\dfrac{\dfrac{\dfrac{P(y_1), Q(a) \vdash P(f(y_2))}{P(y_1), Q(a) \vdash \forall y P(f(y))}\,(3)}{\exists x P(x), Q(a) \vdash \forall y P(f(y))}\,(2)}{\exists x P(x) \wedge Q(a) \vdash \forall y P(f(y))}\,(1)$$

(1) is obtained by applying the $\wedge$-L rule to $\wedge$ on the left-hand side of $\vdash$ in the denominator and (2) is obtained by applying the $\exists$-L rule to $\exists$ on the left-hand side of $\vdash$ in the denominator with y_1 being an eigen-variable with respect to $P, Q(a), \forall y P(f(y))$. (3) is obtained by applying the $\forall$-R rule to $\forall$ on the right-hand side of $\vdash$ in the denominator with y_2 being another eigen-variable with respect to P, Q. Here $P(y_1)$ and $P(f(y_2))$ are different atomic formulas because y_1 and $f(y_2)$ are different terms. Thus $P(y_1), Q(a) \vdash P(f(y_2))$ is not an instance of the **G** axiom. Hence this tree is an inference tree instead of a proof tree. We shall show in the following sections that the sequent $\exists x P(x) \wedge Q(a) \vdash \forall y P(f(y))$ is an unprovable sequent.

Example 3.4 (Applications of the $\forall$-rules and $\exists$-rules [continued]). Consider the sequent $\forall x P(x) \wedge \exists y Q(y) \vdash P(f(v)) \wedge \exists z Q(z)$ whose inference tree is as follows:

$$\dfrac{\dfrac{\dfrac{P(f(v)), \forall x P(x), \exists y Q(y) \vdash P(f(v))}{\forall x P(x), \exists y Q(y) \vdash P(f(v))}\,(3) \qquad \dfrac{\dfrac{\forall x P(x), Q(y_1) \vdash Q(y_1), \exists z Q(z)}{\forall x P(x), Q(y_1) \vdash \exists z Q(z)}\,(5)}{\forall x P(x), \exists y Q(y) \vdash \exists z Q(z)}\,(4)}{\forall x P(x), \exists y Q(y) \vdash P(f(v)) \wedge \exists z Q(z)}\,(2)}{\forall x P(x) \wedge \exists y Q(y) \vdash P(f(v)) \wedge \exists z Q(z)}\,(1)$$

The fraction (1) is obtained by applying the $\wedge$-L rule to $\wedge$ on the left-hand side of the conclusion. The fraction (2) is a result of applying the $\wedge$-R rule to $\wedge$ on the right-hand side of the conclusion. The fraction (3) is obtained by applying the $\forall$-L rule to $\forall x P(x)$ on the left-hand side of the conclusion with $f(v)$ being the term t in the rule, substituted for the free variable x of $P(x)$. The fraction (4) is obtained by applying the $\exists$-L rule to $\exists y Q(y)$ on the left-hand side of the conclusion with y_1 being an eigen-variable not occurring in P or Q. The fraction (5) is obtained by applying the $\exists$-R rule to $\exists z Q(z)$ on the right-hand side of the conclusion with the free variable z in $Q(z)$ replaced by y_1 being the term t in the rule. Thus each fraction is an instance of a **G** rule and the tree is an inference tree. The tree is also a proof tree because each leaf of the tree is an instance of the axiom. Hence the sequent in this example is provable.

Note that the order of the fractions (4) and (5) cannot be reversed. The reason is that if we first apply the $\exists$-R rule to the conclusion of the fraction (4), we shall have

$$\dfrac{\forall x P(x), \exists y Q(y) \vdash Q(t), \exists z Q(z)}{\forall x P(x), \exists y Q(y) \vdash \exists z Q(z)}\,(4')$$

with t being a term. And if we continue to apply the $\exists$-L rule to $\exists y Q(y)$ on the left-hand side in the numerator of the fraction $(4')$, we obtain:

$$\dfrac{\forall x P(x), Q(y_1) \vdash Q(t), \exists z Q(z)}{\forall x P(x), \exists y Q(y) \vdash Q(t), \exists z Q(z)}\,(5').$$

When we apply the $\exists$-*R* rule to $\exists z Q(z)$ on the right-hand side of $\vdash$ in the numerator of the fraction $(4')$, t in the substitution $Q[t/z]$ can be any term, e.g., t can be a constant symbol c, whereas y_1 on the left-hand side of $\vdash$ in the numerator of the fraction $(5')$ is not arbitrary and it has to be an eigen-variable not occurring in P, Q and $Q[t/y]$. Thus because of the difference between $Q(y_1)$ and $Q(t)$, the numerator of the fraction $(5')$ does not constitute an instance of the **G** axiom and the tree is an inference tree instead of a proof tree.

This example shows that we should invoke the $\exists$-*L* rule before applying the $\exists$-*R* rule when employing the **G** system in making formal proofs, whereas we should invoke the $\forall$-*R* rule before applying the $\forall$-*L* rule.

Through the above examples we can obtain some intuitive knowledge of proof trees as well as their constructions. We can see that it should be possible to devise a formal procedure by which an inference tree can be generated for any finite sequent. This will provide an automated procedure for generating formal proofs if the sequent is provable. In the following we will present an outline of such a procedure.

CP: The procedure for constructing formal proofs
Input: the procedure *CP* takes the sequent $\Gamma \vdash \Delta$ to be proved as its input;
Output: when the procedure halts, it outputs the inference tree of $\Gamma \vdash \Delta$ as well as its provability.

Suppose that the input sequent $\Gamma \vdash \Delta$ to be proved is:

$$A_1, \ldots, A_m \vdash B_1, \ldots, B_n$$

Body of *CP*:

(1) Construct an inference tree with $A_1, \ldots, A_m \vdash B_1, \ldots, B_n$ as its root.

(2) Check whether each of its leaves is an instance of the **G** axiom. If there is a leaf $A'_1, \ldots, A'_s \vdash B'_1, \ldots, B'_t$ that is not an instance of the axiom, then go to (3). Otherwise go to (7).

(3) Check whether each A_i $(1 \leqslant i \leqslant s)$ is an atomic formula. If there is an A_i that is not an atomic formula, then go to (4). Otherwise go to (5).

(4) If A_i is with syntactic structure $\exists x A''_i(x)$, $\neg A''_i$, $A''_i \wedge A'''_i$, $A''_i \vee A'''_i$, $A''_i \rightarrow A'''_i$, or $\forall x A''_i(x)$, then apply the corresponding left rules of the **G** system to expand the inference tree and let $i = i + 1$. After that, go to (3).

(5) Check whether each B_j $(1 \leqslant i \leqslant t)$ is an atomic formula. If there is an B_j that is not an atomic formula, then go to (6). Otherwise go to (2).

(6) If B_j is with syntactic structure $\forall x B''_j(x)$, $\neg B''_j$, $B''_j \wedge B'''_j$, $B''_j \vee B'''_j$, $B''_j \rightarrow B'''_j$, or $\exists x B''_j(x)$, then apply the corresponding right rules of the **G** system to expand the inference tree and let $j = j + 1$. After that, go to (5).

(7) Output the proof tree and the provability of $\Gamma \vdash \Delta$.

Readers familiar with programming should recognize that the above proof procedure is a breadth-first search. The reader may find more details in [Gallier, 1986]. It has

been proved that, when the sequent is provable, the above procedure terminates in a finite number of steps and outputs a proof tree. However, when the sequent is unprovable, especially when there are quantifier symbols, it is possible that the procedure does not terminate and generates an infinite inference tree [Gallier, 1986].

It is easy to see that the *CP* procedure presented in this section is not the most efficient search. Nonetheless, one has seen that the advent of formal proofs makes it possible to convert proofs of mathematical theorems into a symbolic calculus which can be accomplished by computers within a man-machine interactive software system.

3.3 Soundness of the G inference system

In the beginning of this chapter we asserted that every proved consequence is a logical consequence and this basic assumption can be expressed in first-order languages as:

$$\text{If } \Gamma \vdash A \text{ is provable, then } \Gamma \models A \text{ holds.}$$

This is called the *soundness* of the **G** system. In this section, we will prove that it holds. To do so, we first prove that for each rule of the **G** system, its denominator sequent has a counterexample if and only if at least one of its numerator sequents has a counterexample. We then prove that its denominator sequent is valid if and only if each of its numerator sequents is valid. Finally we prove that if a sequent is provable then it is valid. We begin by elucidating the semantics of sequents.

Definition 3.11 (Valid sequents). Let Γ be a formula sequence $A_1, \ldots, A_m$ and Δ be $B_1, \ldots, B_n$. We say that a sequent $\Gamma \vdash \Delta$ has a *counterexample* if there exist a structure $\mathbf{M}$ and an assignment σ such that

$$\mathbf{M} \models_\sigma A_i \quad \text{and} \quad \mathbf{M} \models_\sigma \neg B_j$$

hold for all i and j satisfying $1 \leqslant i \leqslant m$ and $1 \leqslant j \leqslant n$. We say that the sequent is *valid* if for any structure $\mathbf{M}$ and assignment σ,

$$\mathbf{M} \models_\sigma \neg A_i \quad \text{or} \quad \mathbf{M} \models_\sigma B_j$$

holds for some i satisfying $1 \leqslant i \leqslant m$ or some j satisfying $1 \leqslant j \leqslant n$.

According to Definition 3.11, the following lemma holds.

Lemma 3.2. *The sequent $\Gamma \vdash \Delta$ in Definition 3.11 is valid if and only if for any structure $\mathbf{M}$ and assignment σ,*

$$\mathbf{M} \models_\sigma (\neg A_1 \vee \cdots \vee \neg A_m \vee B_1 \vee \cdots \vee B_n)$$

holds.

Proof. The conclusion follows immediately from Definition 3.11. □

Definition 3.11 and Lemma 3.2 indicate that the validity of the sequent

$$A_1, \ldots, A_m \vdash B_1, \ldots, B_n$$

is equivalent to the validity of the formula

$$(A_1 \wedge \cdots \wedge A_m) \rightarrow (B_1 \vee \cdots \vee B_n).$$

Lemma 3.3. *For each rule of the **G** system, its denominator sequent has a counterexample if and only if at least one of its numerator sequents has a counterexample.*

Proof. Here we only prove that the lemma applies to the left and right rules of the two logical connective symbols $\neg$, $\vee$, and the quantifier symbol $\forall$ as well as to the cut rule.

The $\neg$-*L* rule:

$$\neg\text{-}L: \quad \frac{\Gamma, \Delta \vdash A, \Lambda}{\Gamma, \neg A, \Delta \vdash \Lambda}$$

As per Definition 3.11, the denominator sequent $\Gamma, \neg A, \Delta \vdash \Lambda$ has a counterexample if and only if there exist a structure $\mathbf{M}$ and an assignment σ such that $\neg A$ as well as each formula in Γ and Δ is true, and each formula in Λ is false. According to the semantics of $\neg$, this is the case if and only if $\mathbf{M}$ and σ make each formula in Γ and Δ true and A as well as each formula in Λ false. As per Definition 3.11, this is the case if and only if $\Gamma, \Delta \vdash A, \Lambda$ has a counterexample.

The $\neg$-*R* rule:

$$\neg\text{-}R: \quad \frac{A, \Gamma \vdash \Lambda, \Theta}{\Gamma \vdash \Lambda, \neg A, \Theta}$$

The denominator sequent $\Gamma \vdash \Lambda, \neg A, \Theta$ has a counterexample if and only if there exist a structure $\mathbf{M}$ and an assignment σ such that each formula in Γ is true, and $\neg A$ as well as each formula in Λ and Θ is false. According to the semantics of $\neg$, this is the case if and only if $\mathbf{M}$ and σ make A as well as each formula in Γ true and each formula in Λ and Θ false. As per Definition 3.11, this is the case if and only if $A, \Gamma \vdash \Lambda, \Theta$ has a counterexample.

The $\vee$-*L* rule:

$$\vee\text{-}L: \quad \frac{\Gamma, A, \Delta \vdash \Lambda \qquad \Gamma, B, \Delta \vdash \Lambda}{\Gamma, A \vee B, \Delta \vdash \Lambda}$$

As per Definition 3.11, the denominator sequent has a counterexample if and only if there exist a structure $\mathbf{M}$ and an assignment σ such that $A \vee B$ and each formula in Γ and Δ are true, and each formula in Λ is false. Under $\mathbf{M}$ and σ, according to the semantics of $\vee$, $A \vee B$ is true if and only if A or B is true. That is, at least one of the following two cases holds.

(1) $\mathbf{M}$ and σ make A as well as each formula in Γ, Δ true and each formula in Λ false, which amounts to $\Gamma, A, \Delta \vdash \Lambda$ having a counterexample.

(2) $\mathbf{M}$ and σ make B as well as each formula in Γ, Δ true and each formula in Λ false, which amounts to $\Gamma, B, \Delta \vdash \Lambda$ having a counterexample.

Hence the denominator sequent $\Gamma, A \vee B, \Delta \vdash \Lambda$ has a counterexample if and only if at least one of the two numerator sequents $\Gamma, A, \Delta \vdash \Lambda$ and $\Gamma, B, \Delta \vdash \Lambda$ has a counterexample.

The $\vee$-R rule:

$$\vee\text{-}R: \quad \frac{\Gamma \vdash \Lambda, A, B, \Theta}{\Gamma \vdash \Lambda, A \vee B, \Theta}$$

As per Definition 3.11, the denominator sequent of the $\vee$-R rule has a counterexample if and only if there exist a structure $\mathbf{M}$ and an assignment σ such that each formula in Γ is true, and $A \vee B$ as well as each formula in Λ and Θ is false. According to the semantics of $\vee$, under $\mathbf{M}$ and σ, this is the case if and only if each formula in Γ is true, and A, B as well as each formula in Λ and Θ are false, that is, if and only if $\Gamma \vdash \Lambda, A, B, \Theta$ has a counterexample.

The $\forall$-L rule:

$$\forall\text{-}L: \quad \frac{\Gamma, A[t/x], \forall x A(x), \Delta \vdash \Lambda}{\Gamma, \forall x A(x), \Delta \vdash \Lambda}$$

As per Definition 3.11, the denominator sequent of the $\forall$-L rule has a counterexample if and only if there exist a structure $\mathbf{M}$ and an assignment σ such that $\forall x A(x)$ as well as each formula in Γ and Δ is true, and each formula in Λ is false. According to the semantics of $\forall$, $\forall x A(x)$ is true if and only if $A_{\mathbf{M}[\sigma[x:=a]]}$ is true for any $a \in M$. For any term t, $t_{\mathbf{M}[\sigma]}$ is an element of M and hence $A_{\mathbf{M}[\sigma[x:=t_{\mathbf{M}[\sigma]}]]}$ is true. From the substitution lemma in Chapter 2, we know that

$$A_{\mathbf{M}[\sigma[x:=t_{\mathbf{M}[\sigma]}]]} = (A[t/x])_{\mathbf{M}[\sigma]}$$

holds for all t. Thus, if $\forall x A(x)$ is true under a structure $\mathbf{M}$ and an assignment σ, then $A[t/x]$ is true under them. That is, $\Gamma, \forall x A(x), \Delta \vdash \Lambda$ has a counterexample if and only if $\Gamma, A[t/x], \forall x A(x), \Delta \vdash \Lambda$ has a counterexample.

The $\forall$-R rule:

$$\forall\text{-}R: \quad \frac{\Gamma \vdash \Lambda, A[y/x], \Theta}{\Gamma \vdash \Lambda, \forall x A(x), \Theta}$$

where y is an eigen-variable different from the variables of Γ, Λ, $\forall x A(x)$ and Θ.

As per Definition 3.11, the denominator sequent of the $\forall$-R rule has a counterexample if and only if there exist a structure $\mathbf{M}$ and an assignment σ such that Γ is true, and $\forall x A$ as well as each formula in Λ and Θ is false. According to the semantics of $\forall$, $\forall x A$ is false under $\mathbf{M}$ and σ if and only if there exists an $m \in M$ such that $A_{\mathbf{M}[\sigma[x:=m]]} = F$. Since y is an eigen-variable different from the variables of Γ, Λ, A and Θ, let $\sigma(y) = m$. According to the substitution lemma we have

$$\begin{aligned}(A[y/x])_{\mathbf{M}[\sigma]} &= A_{\mathbf{M}[\sigma[x:=y_{\mathbf{M}[\sigma]}]]} \\ &= A_{\mathbf{M}[\sigma[x:=m]]} \\ &= F.\end{aligned}$$

This is equivalent to $A[y/x]$ being false under $\mathbf{M}$ and σ. Thus the denominator sequent has a counterexample if and only if $\mathbf{M}$ and σ make Γ true and $A[y/x]$ as well as each formula in Λ and Θ false. As per Definition 3.11, this amounts to the numerator sequent $\Gamma \vdash \Lambda, A[y/x], \Theta$ having a counterexample. □

Lemma 3.4 (Validity of rules). *For each **G** rule, its denominator sequent is valid if and only if its numerator sequents are all valid.*

Proof. Based on Lemma 3.3, we prove this lemma by contradiction. First, for each **G** rule, if its denominator sequent is valid, then each sequent in its numerator is also valid. Otherwise some sequent in its numerator having a counterexample would indicate that, as per Lemma 3.3, its denominator sequent has a counterexample, which contradicts the assumption. On the other hand, if each sequent in its numerator is valid, then its denominator sequent is also valid. Otherwise its denominator sequent having a counterexample would indicate that, as per Lemma 3.3, some sequent in its numerator has a counterexample, which would be a contradiction. □

Lemma 3.5. *The **G** axiom is valid.*

Proof. It is impossible for the **G** axiom $\Gamma, A, \Delta \vdash \Lambda, A, \Theta$ to have a counterexample because for any structure $\mathbf{M}$ and assignment σ, it is impossible that A on the left-hand side of $\vdash$ is true, whereas the same A on the right-hand side of $\vdash$ is false. As per Definition 3.11, the **G** axiom is valid. □

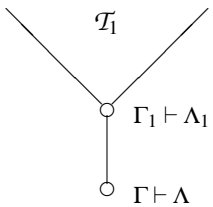
Having proved Lemmas 3.4 and 3.5, we can prove the soundness of the **G** system.

Theorem 3.1 (Soundness of the G system). *If the sequent $\Gamma \vdash \Lambda$ is provable, then $\Gamma \models \Lambda$ holds.*

Proof. According to the condition of the theorem, the sequent $\Gamma \vdash \Lambda$ is provable. Hence there exists a proof tree $\mathcal{T}$ of this sequent. We prove the theorem by structural induction on the tree $\mathcal{T}$.

If $\mathcal{T}$ is a single node tree, then it is an instance of the **G** axiom. As per Lemma 3.5, the sequent is valid. If $\mathcal{T}$ is not a single node tree, then there are two possibilities.

(1) $\mathcal{T}$ is a proof tree of the following form:

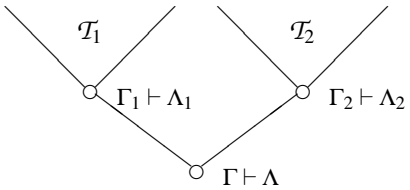


where $\mathcal{T}_1$ is a proof tree of $\Gamma_1 \vdash \Lambda_1$ and the fraction

$$\frac{\Gamma_1 \vdash \Lambda_1}{\Gamma \vdash \Lambda}$$

is an instance of some rule of the **G** system. According to the hypothesis of the induction $\Gamma_1 \models \Lambda_1$ holds, because $\mathcal{T}_1$ is a proof tree. Then by Lemma 3.4, we know that $\Gamma \models \Lambda$ holds.

(2) $\mathcal{T}$ is a proof tree of the following form:



where $\mathcal{T}_1, \mathcal{T}_2$ are proof trees of $\Gamma_1 \vdash \Lambda_1$ and $\Gamma_2 \vdash \Lambda_2$ respectively and the sequent

$$\frac{\Gamma_1 \vdash \Lambda_1 \quad \Gamma_2 \vdash \Lambda_2}{\Gamma \vdash \Lambda}$$

is an instance of some inference rule of the **G** system. According to the hypothesis of the induction, both $\Gamma_1 \models \Lambda_1$ and $\Gamma_2 \models \Lambda_2$ hold because $\mathcal{T}_1, \mathcal{T}_2$ are proof trees. Then by Lemma 3.4, we know that $\Gamma \models \Lambda$ holds. □

3.4 Compactness and consistency

The concept of *compactness* expresses the finiteness of formal proofs.

Theorem 3.2 (Compactness). *If Γ is a formula set and A is a formula with the sequent $\Gamma \vdash A$ provable, then there exists a finite formula set Δ such that $\Delta \subseteq \Gamma$ and $\Delta \vdash A$ is provable.*

Proof. If $\Gamma \vdash A$ is provable, then there exists a finite proof tree whose root is the sequent $\Gamma \vdash A$. The number of instances of the **G** rules applied by the proof tree is also finite. Denote the set of these instances by $\mathcal{R}$. $\mathcal{R}$ is pertinent only to a finite number of formulas that can be divided into two categories: one of them consists of formulas contained in Γ and is denoted as $\{A_{n_1}, A_{n_2}, \ldots, A_{n_k}\}$; the other consists of side formulas appearing in the instances of $\mathcal{R}$ and is denoted as $\{A_{m_1}, \ldots, A_{m_l}\}$. Let $\Delta = \{A_{n_1}, A_{n_2}, \ldots, A_{n_k}\}$. Then Δ is finite and $\Delta \subseteq \Gamma$. By the definition of proof trees, we will construct a proof tree of $\Delta \vdash A$ after deleting all the formulas, which are neither in Δ nor in $\{A_{m_1}, \ldots, A_{m_l}\}$, from the proof tree of $\Gamma \vdash A$. □

The compactness theorem indicates that if a sequent $\Gamma \vdash A$ is provable, then there exists a finite formula set Δ contained in Γ such that $\Delta \vdash A$ is provable. Hence the formal proof of $\Gamma \vdash A$ only uses a finite set of formulas contained in Γ even if Γ is a countably infinite set of formulas. Thus, henceforth, when Γ is a countably infinite set, the previous lemmas and theorems in this chapter still hold.

Lemma 3.6. (1) *If $\Gamma \vdash A$ is provable and $\Sigma \supseteq \Gamma$, then $\Sigma \vdash A$ is provable.*

(2) *If Λ is a formula set and $\Gamma \vdash A$ is provable, then $\Gamma \vdash A, \Lambda$ is provable.*

Proof. (1) From $\Gamma \vdash A$ being provable we know that it has a proof tree $\mathcal{T}$. Since $\Sigma \supseteq \Gamma$, we let $\Delta = \Sigma - \Gamma$ and add Δ to the left-hand side of $\vdash$ in each sequent appearing in the proof tree $\mathcal{T}$. The tree $\mathcal{T}'$ obtained is the proof tree of $\Sigma \vdash A$. Thus $\Sigma \vdash A$ is provable.

(2) If $\Gamma \vdash A$ is provable, then it has a proof tree $\mathcal{T}$. We add Λ to the right-hand side of $\vdash$ in each sequent appearing in the proof tree $\mathcal{T}$. The tree $\mathcal{T}'$ obtained is the proof tree of $\Gamma \vdash A, \Lambda$. Thus $\Gamma \vdash A, \Lambda$ is provable. □

Definition 3.12 (Consistency). Let Γ be a formula set. If there does not exist any formula A such that both sequents $\Gamma \vdash A$ and $\Gamma \vdash \neg A$ are provable, then we say that Γ is *consistent*.

Lemma 3.7. (1) *If a formula set Γ is consistent, then there exists a formula A such that the sequent $\Gamma \vdash A$ is unprovable.*

(2) *A formula set Γ is inconsistent if and only if for any formula A, both $\Gamma \vdash A$ and $\Gamma \vdash \neg A$ are provable.*

(3) *If a formula set Γ is consistent and the sequent $\Gamma \vdash A$ is provable, then the formula set $\Gamma \cup \{A\}$ is consistent. In this case we also say that Γ and A are* consistent.

(4) *If $\Gamma \vdash A$ is unprovable, then Γ is consistent with $\neg A$.*

Proof. (1) We prove the statement by contradiction. Suppose that there does not exist any formula A such that $\Gamma \vdash A$ is unprovable. Then for any formula B, both $\Gamma \vdash B$ and $\Gamma \vdash \neg B$ are provable and this contradicts the consistency of Γ.

(2) Sufficiency. If for every formula A, both $\Gamma \vdash A$ and $\Gamma \vdash \neg A$ are provable, then from the definition of consistency we know that Γ is inconsistent.

Necessity. If Γ is inconsistent, as per Definition 3.12, there exists a formula B such that both sequents $\Gamma \vdash B$ and $\Gamma \vdash \neg B$ are provable. From the $\neg$-*R* rule, $\Gamma, B \vdash$ is provable. Hence $\Gamma \vdash$ is provable according to the cut rule. According to Lemma 3.6, for any formula A, both $\Gamma, \neg A \vdash$ and $\Gamma, A \vdash$ are provable. This amounts to both $\Gamma \vdash A$ and $\Gamma \vdash \neg A$ being provable.

(3) We prove the statement by contradiction. Suppose that the set $\Gamma \cup \{A\}$ is inconsistent. Then for the formula A, both $\Gamma, A \vdash A$ and $\Gamma, A \vdash \neg A$ are provable. We apply the $\neg$-*R* rule to the latter sequent and hence we know that $\Gamma \vdash \neg A$ is provable. According to the condition, $\Gamma \vdash A$ is provable as well. This contradicts the consistency of Γ.

(4) We prove the statement by contradiction. If Γ is inconsistent with $\neg A$, as per (2) we know that $\Gamma, \neg A \vdash A$ is provable. An application of the $\neg$-*L* rule indicates $\Gamma \vdash A$ being provable, which contradicts the unprovability of $\Gamma \vdash A$. □

We define maximal consistent sets as follows:

Definition 3.13 (Maximal consistent sets). A formula set Γ is called a *maximal consistent set* if for any formula A, Γ being consistent with A implies that $A \in \Gamma$.

Lemma 3.8. *Let Γ be a maximal consistent set and A be a formula. Then $\Gamma \vdash A$ if and only if $A \in \Gamma$.*

Proof. Sufficiency. If $A \in \Gamma$, then the **G** axiom indicates that $\Gamma \vdash A$ is provable.

Necessity. If $\Gamma \vdash A$, then (3) of Lemma 3.7 indicates that Γ is consistent with A. Thus $A \in \Gamma$ as per Definition 3.13. □

3.5 Completeness of the G inference system

The complement of soundness is that every logical consequence is a formal consequence, and it can be expressed in first-order languages as

$$\text{If } \Gamma \models A \text{ holds, then } \Gamma \vdash A \text{ is provable.}$$

This is called the *completeness* of the **G** system. The purpose of this section is to prove the completeness of the **G** system. This amounts to proving that for any model $(\mathbf{M},\sigma)$, if Γ being true under $(\mathbf{M},\sigma)$ implies A being true under $\mathbf{M}$ and σ, then $\Gamma \vdash A$ is provable.

We will prove the completeness of the **G** inference system in four steps.

We first prove that for any consistent formula set Γ, there exists a method to extend Γ into a maximal consistent set. This method is called the *Lindenbaum procedure* and the maximal consistent set is called *Lindenbaum extension*.

Secondly, we prove that each Lindenbaum extension is a Hintikka set.

Thirdly, we prove that each consistent set is satisfiable according to the first and second steps since we have already proved the satisfiability of Hintikka sets in Chapter 2.

Finally, we prove the completeness of the **G** system by contradiction. In fact, if $\Gamma \models A$ holds but $\Gamma \vdash A$ is unprovable, then $\Gamma \cup \{\neg A\}$ is consistent according to (4) of Lemma 3.7. As per the third step, $\Gamma \cup \{\neg A\}$ is satisfiable, i.e., there exist a structure $\mathbf{M}$ and an assignment σ such that Γ and $\neg A$ are true. This contradicts the validity of $\Gamma \models A$.

It should be noted that because the set of formulas of $\mathscr{L}$ is a countable set, we can list the formulas of $\mathscr{L}$ as the following formula sequence:

$$A_1, A_2, \ldots, A_n, \ldots.$$

For instance, we can list the formulas in $\mathscr{L}$ as a sequence according to their rank and lexical ordering.

Definition 3.14 (Lindenbaum extension). Let Γ be a consistent formula set of $\mathscr{L}$. For any n, the formula set Γ_n is defined inductively as follows. $\Gamma_1 = \Gamma$ and

$$\Gamma_{n+1} = \begin{cases} \Gamma_n \cup \{A_n\}, & \text{if } \Gamma_n \text{ and } A_n \text{ are consistent,} \\ \Gamma_n, & \text{otherwise.} \end{cases}$$

Let

$$\Sigma = \bigcup_{n=1}^{\infty} \Gamma_n.$$

The above inductive definition is called the *Lindenbaum procedure* and Σ is called the *Lindenbaum extension* of Γ.

Lemma 3.9. *Let Γ be a consistent formula set. The Lindenbaum extension Σ of Γ is a maximal consistent formula set containing Γ.*

Proof. Evidently $\Gamma \subseteq \Sigma$. If the lemma does not hold, then either Σ is inconsistent or Σ is consistent, but not maximal.

(1) We prove the consistency of Σ by contradiction. If Σ is inconsistent, as per the definition of consistency there exists a formula A such that both $\Sigma \vdash A$ and $\Sigma \vdash \neg A$ are provable. From the compactness theorem we know that there exists a finite subset Δ of Σ such that both $\Delta \vdash A$ and $\Delta \vdash \neg A$ are provable. From the procedure of Lindenbaum extension we know further that there exists a $\Gamma_n \supseteq \Delta$. Lemma 3.6 indicates that both $\Gamma_n \vdash A$ and $\Gamma_n \vdash \neg A$ are provable. This contradicts the consistency of Γ_n.

(2) We then prove the maximality of Σ by contradiction. If Σ is consistent, but not maximal, then there exists an $A \notin \Sigma$ and Σ is consistent with A. Let A be denoted as A_n in the formula sequence of the Lindenbaum extension. If A_n is consistent with Γ_n, then we have $A_n \in \Gamma_{n+1} \subseteq \Sigma$, which is a contradiction. If A_n is inconsistent with Γ_n, then there exists a formula B such that both $\Gamma_n, A_n \vdash B$ and $\Gamma_n, A_n \vdash \neg B$ are provable. Since $\Gamma_n \subseteq \Sigma$, Lemma 3.6 indicates that $\Sigma, A_n \vdash B$ and $\Sigma, A_n \vdash \neg B$ are both provable, which contradicts the consistency of Σ and A. □

The Lindenbaum extension has the following completeness.

Lemma 3.10. *If Σ is the Lindenbaum extension of Γ, then for any formula A, either $A \in \Sigma$ or $\neg A \in \Sigma$.*

Proof. According to the construction of the Lindenbaum sequence, we might as well suppose that A is A_m.

If Γ_m is consistent with A, then according to the procedure of Lindenbaum extension, $A_m \in \Gamma_{m+1}$. And $\Gamma_{m+1} \subseteq \Sigma$ further indicates that $A_m \in \Sigma$, i.e., $A \in \Sigma$. In this case it is impossible that $\neg A \in \Sigma$. Otherwise, if $\neg A \in \Sigma$, then both $\Sigma \vdash \neg A$ and $\Sigma \vdash A$ are provable, which contradicts the consistency of Σ.

If Γ_m is inconsistent with A, then according to the procedure of Lindenbaum extension, $A_m \notin \Gamma_{m+1}$ and thus $A_m \notin \Sigma$, i.e., $A \notin \Sigma$. In this case we have $\neg A \in \Sigma$. Otherwise, if $\neg A \notin \Sigma$, then from the maximal consistency of Σ we know that $\neg A$ and Σ are inconsistent. Since Σ and A are inconsistent as well, according to (2) of Lemma 3.7, both $\Sigma, \neg A \vdash A$ and $\Sigma, A \vdash \neg A$ are provable. Hence both $\Sigma \vdash A$ and $\Sigma \vdash \neg A$ are provable. This contradicts the consistency of Σ. □

Lemma 3.11. *If Γ is a consistent set and Σ is its Lindenbaum extension, then Σ is a Hintikka set.*

Proof. It suffices to prove that Σ satisfies the conditions in the definition of a Hintikka set. The proof is as follows.

(1) From Lemma 3.10, we know that for any atomic formula A, either $A \in \Sigma$ or $\neg A \in \Sigma$.

(2) If $A \in \Sigma$, then from Lemma 3.8 we know that $\Sigma \vdash A$ is provable. According to the $\neg$-*L* rule, $\Sigma, \neg A \vdash$ is provable. Then by the $\neg$-*R* rule, $\Sigma \vdash \neg\neg A$ is provable. Hence Lemma 3.8 indicates that $\neg\neg A \in \Sigma$ holds.

(3) If $A \in \Sigma$ or $B \in \Sigma$, then Lemma 3.8 indicates that either $\Sigma \vdash A$ or $\Sigma \vdash B$ is provable. According to (2) of Lemma 3.6, $\Sigma \vdash A, B$ is provable. And as per the $\vee$-*R* rule, $\Sigma \vdash A \vee B$ is provable. According to Lemma 3.8, $A \vee B \in \Sigma$ holds.

(4) If $\neg A \in \Sigma$ and $\neg B \in \Sigma$, then according to Lemma 3.8, both $\Sigma \vdash \neg A$ and $\Sigma \vdash \neg B$ are provable. As per the $\neg$-*R* rule, both $\Sigma, A \vdash$ and $\Sigma, B \vdash$ are provable. As per the $\vee$-*L* rule, $\Sigma, A \vee B \vdash$ is provable. And as per the $\neg$-*R* rule, $\Sigma \vdash \neg(A \vee B)$ is provable as well. According to Lemma 3.8, $\neg(A \vee B) \in \Sigma$ holds.

(5) If $A \in \Sigma$ and $B \in \Sigma$, then according to Lemma 3.8, $\Sigma \vdash A$ and $\Sigma \vdash B$. As per the $\wedge$-*R* rule, $\Sigma \vdash A \wedge B$ is provable. Hence Lemma 3.8 indicates that $A \wedge B \in \Sigma$ holds.

(6) If $\neg A \in \Sigma$ or $\neg B \in \Sigma$, then according to Lemma 3.8, either $\Sigma \vdash \neg A$ or $\Sigma \vdash \neg B$ is provable. According to (2) of Lemma 3.6, $\Sigma \vdash \neg A, \neg B$ is provable. And as per the $\neg$-*R* rule, $\Sigma, A, B \vdash$ is provable. In addition, as per the $\wedge$-*L* rule, $\Sigma, A \wedge B \vdash$ is provable. Finally, as per the $\neg$-*R* rule, $\Sigma \vdash \neg(A \wedge B)$ is provable. Lemma 3.8 indicates that $\neg(A \wedge B) \in \Sigma$ holds.

(7) If $\neg A \in \Sigma$ or $B \in \Sigma$, then Lemma 3.8 indicates that $\Sigma \vdash \neg A$ or $\Sigma \vdash B$ is provable. According to (2) of Lemma 3.6, $\Sigma \vdash \neg A, B$ is provable. And as per the $\neg$-*R* rule, $\Sigma, A \vdash B$ is provable. In addition, as per the $\rightarrow$-*R* rule, $\Sigma \vdash A \rightarrow B$ is provable. Lemma 3.8 indicates that $A \rightarrow B \in \Sigma$ holds.

(8) If $A \in \Sigma$ and $\neg B \in \Sigma$, then Lemma 3.8 indicates that both $\Sigma \vdash A$ and $\Sigma \vdash \neg B$ are provable. As per the $\neg$-*R* rule, $\Sigma, B \vdash$ is provable. As per the $\rightarrow$-*L* rule, $\Sigma, A \rightarrow B \vdash$ is provable. In addition, according to the $\neg$-*R* rule, $\Sigma \vdash \neg(A \rightarrow B)$ is provable as well. Lemma 3.8 indicates that $\neg(A \rightarrow B) \in \Sigma$ holds.

(9) If for every $t \in H$, $\neg A[t/x] \in \Sigma$ holds and in particular we take y as t, then $\neg A[y/x] \in \Sigma$. Here H is a Herbrand domain containing variables, i.e., the set of terms, and y is the variable x or a variable that does not occur in A and Σ. Lemma 3.8 indicates that $\Sigma \vdash \neg A[y/x]$ is provable. As per the $\neg$-*R* rule, $\Sigma, A[y/x] \vdash$ is provable. As per the $\exists$-*L* rule, this amounts to $\Sigma, \exists x A(x) \vdash$ being provable. As per the $\neg$-*R* rule, $\Sigma \vdash \neg(\exists x A(x))$ is also provable. Lemma 3.8 indicates that $\neg(\exists x A(x)) \in \Sigma$ holds.

(10) If there exists a $t \in H$ such that $A[t/x] \in \Sigma$ holds, then $\Sigma \vdash A[t/x]$ is provable. As per the $\exists$-*R* rule, $\Sigma \vdash \exists x A(x)$ is provable. Then according to Lemma 3.8, $\exists x A(x) \in \Sigma$.

(11) If for every $t \in H$, $A[t/x] \in \Sigma$ holds and in particular we take y as t, then $A[y/x] \in \Sigma$ holds, where y is the variable x or a variable that does not occur in A or Σ. Lemma 3.8 indicates that $\Sigma \vdash A[y/x]$ is provable. As per the $\forall$-*R* rule, $\Sigma \vdash \forall x A(x)$ is provable. According to Lemma 3.8, $\forall x A(x) \in \Sigma$ holds.

(12) If there exists a $t \in H$ such that $\neg A[t/x] \in \Sigma$, then $\Sigma \vdash \neg A[t/x]$ is provable. As per the $\neg$-*R* rule, $\Sigma, A[t/x] \vdash$ is provable. As per the $\forall$-*L* rule, $\Sigma, \forall x A(x) \vdash$ is provable. And as per the $\neg$-*R* rule, $\Sigma \vdash \neg(\forall x A(x))$ is provable. Finally Lemma 3.8 indicates that $\neg(\forall x A(x)) \in \Sigma$.

We have now proved that Σ is a Hintikka set. □

Theorem 3.3 (Satisfiability). *If Γ is a consistent formula set, then Γ is satisfiable.*

Proof. We first extend Γ into a maximal consistent formula set Σ by invoking the Lindenbaum procedure. Lemma 3.11 indicates that Σ is a Hintikka set. According to Theorem 2.2, Σ is satisfiable. Thus Γ is satisfiable. □

Theorem 3.4 (Completeness). *Let Γ be a formula set and A be a formula. If $\Gamma \models A$ holds, then $\Gamma \vdash A$ is provable.*

Proof. We prove the theorem by contradiction. Suppose that $\Gamma \vdash A$ is unprovable. Then (4) of Lemma 3.7 indicates that $\Gamma \cup \{\neg A\}$ is consistent. Theorem 3.3 indicates that there exist a structure $\mathbf{M}$ and an assignment σ such that both $\mathbf{M} \models_\sigma \Gamma$ and $\mathbf{M} \models_\sigma \neg A$ hold. Nonetheless since $\Gamma \models A$ is valid, $\mathbf{M} \models_\sigma A$ must hold. This contradicts the principle of excluded middle. □

Summarizing the results of the previous sections of this chapter, we can obtain the following theorem.

Theorem 3.5. *Let Γ be a formula set and A be a formula.*

(1) *$\Gamma \models A$ is valid if and only if $\Gamma \vdash A$ is provable.*

(2) *Γ is satisfiable if and only if Γ is consistent.*

Proof. The conclusion (1) of the theorem can be directly deduced from Theorem 3.1 on soundness and Theorem 3.4 on completeness. As per Theorem 3.3, Γ being consistent indicates that it is satisfiable. Thus to prove the conclusion (2) of the theorem, it suffices to prove that if Γ is satisfiable, then it is also consistent. We prove this by contradiction. If Γ is inconsistent, then by definition there exists a formula A such that both $\Gamma \vdash A$ and $\Gamma \vdash \neg A$ are provable. Theorem 3.1 indicates that both $\Gamma \models A$ and $\Gamma \models \neg A$ are valid. Since Γ is satisfiable, there exist a structure $\mathbf{M}$ and an assignment σ such that Γ is true and hence A and $\neg A$ are true. This contradicts the principle of excluded middle. Thus Γ is consistent. □

This theorem shows that, for first-order languages, the principle that "*every proved conclusion is a logical consequence and vice versa*" holds. If the knowledge about a domain can be described by first-order languages, then this theorem furnishes a theoretical foundation for the conversion of mathematical proofs in the domain into symbolic calculus.

3.6 Some commonly used inference rules

In mathematical and scientific research, some methods of logical deduction are commonly used. Such methods are based on the rules of proof by contradiction, proof by cases, the rule of inconsistency, converse-negative deduction, the rule of modus ponens and the rule of substitution. In this section we present the formal inference rules of first-order languages in a form similar to those in "Grundlagen der Geometrie" of Hilbert [1899]. As in the completeness of the **G** system they can all be proved as deduced rules from the

G system. Since these deduced rules will be used in later chapters, this section provides semantic proofs for them, for the sake of simplicity.

Rule of proof by contradiction:

$$\frac{\neg A, \Gamma \vdash B \qquad \neg A, \Gamma \vdash \neg B}{\Gamma \vdash A}$$

Proof. We need to prove that if both $\neg A, \Gamma \vdash B$ and $\neg A, \Gamma \vdash \neg B$ are provable, then $\Gamma \vdash A$ is provable. Since both $\neg A, \Gamma \vdash B$ and $\neg A, \Gamma \vdash \neg B$ are provable, the soundness theorem indicates that $\neg A, \Gamma \models B$ and $\neg A, \Gamma \models \neg B$. For any structure $\mathbf{M}$ and assignment σ such that $\mathbf{M} \models_\sigma \Gamma$, if $\mathbf{M} \models_\sigma A$ does not hold, then $\mathbf{M} \models_\sigma \neg A$ and thus $\mathbf{M} \models_\sigma \neg A, \Gamma$. Further, $\neg A, \Gamma \models B$ and $\neg A, \Gamma \models \neg B$ indicate that $\mathbf{M} \models_\sigma B$ and $\mathbf{M} \models_\sigma \neg B$, which contradicts the principle of excluded middle for domains. Hence $\mathbf{M} \models_\sigma A$ holds and as a result, $\Gamma \models A$. Thus we have $\Gamma \vdash A$. □

Rule of proof by cases:

$$\frac{A, \Gamma \vdash B \qquad \neg A, \Gamma \vdash B}{\Gamma \vdash B}$$

Proof. We need to prove that if both $A, \Gamma \vdash B$ and $\neg A, \Gamma \vdash B$ are provable, then $\Gamma \vdash B$ is provable. Since both $A, \Gamma \vdash B$ and $\neg A, \Gamma \vdash B$ are provable, the soundness theorem indicates that $A, \Gamma \models B$ and $\neg A, \Gamma \models B$. For any structure $\mathbf{M}$ and assignment σ such that $\mathbf{M} \models_\sigma \Gamma$, in what follows we prove that $\mathbf{M} \models_\sigma B$ holds. In fact, if $\mathbf{M} \models_\sigma A$, then $\mathbf{M} \models_\sigma A, \Gamma$. Hence $A, \Gamma \models B$ implies $\mathbf{M} \models_\sigma B$. If $\mathbf{M} \models_\sigma \neg A$, then $\mathbf{M} \models_\sigma \neg A, \Gamma$, and thus $\neg A, \Gamma \models B$ implies $\mathbf{M} \models_\sigma B$. As a result we always have $\mathbf{M} \models_\sigma B$. Namely, for any structure $\mathbf{M}$ and assignment σ, $\mathbf{M} \models_\sigma \Gamma$ implies $\mathbf{M} \models_\sigma B$. As per the completeness theorem, we have $\Gamma \vdash B$. □

Rule of inconsistency:

$$\frac{\Gamma \vdash A \qquad \Gamma \vdash \neg A}{\Gamma \vdash B}$$

Proof. We need to prove that if both $\Gamma \vdash A$ and $\Gamma \vdash \neg A$ are provable, then $\Gamma \vdash B$ is provable for any formula B. If both $\Gamma \vdash A$ and $\Gamma \vdash \neg A$ are provable, then Γ is inconsistent. Lemma 3.7 (2) indicates that $\Gamma \vdash B$ is provable for any formula B. □

Converse-negative deduction:

$$\frac{A, \Gamma \vdash B}{\neg B, \Gamma \vdash \neg A} \text{ or } \frac{\Gamma \vdash A \to B}{\Gamma \vdash \neg B \to \neg A}$$

Proof. The first rule is proved as follows. If $A, \Gamma \vdash B$ is provable, then $\neg B, \Gamma \vdash \neg A$ is provable. As per the completeness theorem, it suffices to prove that for any structure $\mathbf{M}$ and assignment σ, if $\mathbf{M} \models_\sigma \neg B, \Gamma$ holds, then so does $\mathbf{M} \models_\sigma \neg A$. In fact, if $\mathbf{M} \models_\sigma \neg A$ does not hold, then $\mathbf{M} \models_\sigma A$ holds and thus $\mathbf{M} \models_\sigma A, \Gamma$ holds. By $A, \Gamma \vdash B$ and the soundness theorem we have $\mathbf{M} \models_\sigma B$, which contradicts $\mathbf{M} \models_\sigma \neg B, \Gamma$. The second rule can be proved similarly. □

Rule of modus ponens:

$$\frac{\Gamma \vdash A \qquad \Gamma \vdash A \to B}{\Gamma \vdash B}\ (1) \qquad\qquad \frac{\Gamma \vdash A[t/x] \qquad \Gamma \vdash \forall x(A(x) \to B(x))}{\Gamma \vdash B[t/x]}\ (2)$$

Proof. In order to prove rule (1) we need to prove that if both $\Gamma \vdash A$ and $\Gamma \vdash A \to B$ are provable, then $\Gamma \vdash B$ is provable. By the completeness theorem, it suffices to prove that for any structure $\mathbf{M}$ and assignment σ, if $\mathbf{M} \models_\sigma \Gamma$ holds, then so does $\mathbf{M} \models_\sigma B$. Since $\Gamma \vdash A$ is provable and $\mathbf{M} \models_\sigma \Gamma$ holds, the soundness theorem implies that $\mathbf{M} \models_\sigma A$ holds. And $\Gamma \vdash A \to B$ further implies that $\mathbf{M} \models_\sigma A \to B$. Thus according to Definitions 2.7 and 2.8, $\mathbf{M} \models_\sigma B$ holds. The proof for rule (2) is similar. □

The substitution rule is a formal inference rule for the equality symbol. The rule is given below:

Rule of substitution:

$$\frac{\Gamma \vdash A[t/x]}{\Gamma, t \doteq s \vdash A[s/x]}$$

Proof. We should prove that if $\Gamma \vdash A[t/x]$ is provable, then $\Gamma, t \doteq s \vdash A[s/x]$ is provable, where A stands for a formula, t and s stand for two terms. Suppose $\Gamma \vdash A[t/x]$ is provable. By soundness theorem, for any structure $\mathbf{M}$ and assignment σ, if $\mathbf{M} \models_\sigma \Gamma$ holds, then $\mathbf{M} \models_\sigma A[t/x]$ holds, i.e., $(A[t/x])_{\mathbf{M}[\sigma]} = \mathrm{T}$ holds. If we assume that $\mathbf{M} \models_\sigma t \doteq s$ holds, i.e., $(t \doteq s)_{\mathbf{M}[\sigma]} = \mathrm{T}$. According to Definition 2.8, $(t)_{\mathbf{M}[\sigma]} = (s)_{\mathbf{M}[\sigma]}$ holds. Thus, for any structure $\mathbf{M}$ and assignment σ, if $\mathbf{M} \models_\sigma \Gamma$ and $\mathbf{M} \models_\sigma t \doteq s$, i.e., $\mathbf{M} \models_\sigma \Gamma, t \doteq s$, then

$$\begin{aligned} \mathrm{T} &= (A[t/x])_{\mathbf{M}[\sigma]} \\ &= A_{\mathbf{M}[\sigma[x:=t_{\mathbf{M}[\sigma]}]]} \quad \text{(by substitution lemma)} \\ &= A_{\mathbf{M}[\sigma[x:=s_{\mathbf{M}[\sigma]}]]} \quad (\text{since} (t)_{\mathbf{M}[\sigma]} = (s)_{\mathbf{M}[\sigma]}) \\ &= (A[s/x])_{\mathbf{M}[\sigma]} \quad \text{(by substitution lemma)}. \end{aligned}$$

This means $\mathbf{M} \models_\sigma A[s/x]$ holds. According to the completeness theorem, $\Gamma, t \doteq s \vdash A[s/x]$ is provable. □

3.7 Proof theory and model theory

Up to this point we have systematically studied two groups of related concepts about the syntax and semantics of first-order languages:

Syntax	Semantics
first-order language	model
constant symbol	constant
function symbol	function
predicate	relation
equality symbol $\doteq$	equal sign $=$
atomic formula	atomic proposition
composite formula	composite proposition
substitution	assignment

In this chapter we have studied in depth two groups of concepts of sequents. One group is of provability and validity whereas the other is of consistency and satisfiability:

Syntax	Semantics
formal inference	logical reasoning
formal proof	mathematical proof
provability	validity
consistency	satisfiability

In the study of first-order languages, provability is usually considered the kernel of proof theory. Provability refers to a sequent $\Gamma \vdash A$ being provable, in which case A is called a *formal consequence* of Γ. A formal proof of $\Gamma \vdash A$ is a tree whose root is the sequent, whose leaves are instances of the **G** axiom, and whose remaining nodes are instances of **G** rules only. Each **G** inference rule is a symbolic operation on a logical connective symbol or quantifier symbol occurring in the sequent. From this point of view, the **G** system can be viewed as a symbolic calculus on logical connectives symbols and quantifier symbols. Since procedures can be built for constructing formal proofs by applying the **G** inference rules, the provability of a formal consequence A of Γ can be demonstrated by calling the procedures of the symbolic calculus.

Validity is usually considered to be the kernel of model theory. Validity refers to the fact that $\Gamma \models A$ holds, in which case A is called a *logical consequence* of Γ. When A is true under all the models which make Γ true then $\Gamma \models A$ is valid. In principle, validity has to be corroborated by checking all the models of Γ and A.

In this chapter we have also proved the soundness and completeness of the **G** inference system. Namely, if $\Gamma \vdash A$ is provable, then $\Gamma \models A$ holds; conversely if $\Gamma \models A$ holds, then $\Gamma \vdash A$ is provable. In short, we have demonstrated one of the most important results of first-order languages, *all the provable formulas are valid and all the valid formulas are provable*. In other words, for first-order languages and their models, provability and validity are equivalent concepts.

Consistency is another key concept in the proof theory of first-order languages, whereas satisfiability is a key concept in the model theory of first-order languages. Consistency asserts the non-existence of a formula A such that both $\Gamma \vdash A$ and $\Gamma \vdash \neg A$ are provable. Consistency and satisfiability are another pair of equivalent concepts.

This useful result was proved in this chapter by showing that if Γ is consistent then Γ is satisfiable and conversely. This shows that, in order to prove the consistency of Γ, we only need to find a model in which Γ is true.

The difference between satisfiability and validity is that Γ is satisfiable as long as **there exists** a model in which Γ is true, whereas $\Gamma \models A$ is valid if, for **any model M**, Γ being true indicates A being true.

Finally, since the introduction of first-order languages has converted mathematical proofs into symbolic calculus, it is natural to ask whether, for a given formula set Γ and a formula A inconsistent with Γ, we can construct another symbolic calculus system which can delete all the formulas in Γ that are inconsistent with A and derive the maximal subsets of Γ that are consistent with A. The answer is affirmative and we will discuss this inference system in Chapter 7.

Chapter 4

Computability & Representability

From the view point of functionality, there are two kinds of knowledge about a specific domain: one is *specificational knowledge* and the other is *implementational knowledge*. The latter is also called *constructive knowledge*. In computer science, the former is the specification for the software while the latter consists of the actual algorithms and programs used to implement the software. These two kinds of knowledge describe two different aspects of the same thing. Specificational knowledge describes the object by its properties. These might include principles, laws and theorems as well as describing functionality and other requirements. Implementational knowledge explains how to construct the object; it usually includes algorithms, rules of operation and methods of implementation as well as examples.

Specificational knowledge for a specific domain can often take the form of an axiom system. Such an axiom system may consist of several axioms: each axiom is a proposition and each proposition is composed of some basic concepts linked by logical connectives and quantifiers. If the basic concepts can be described by predicate symbols and the functions occurring in the propositions can be described by function symbols of a first-order language, then the axiom system can be described by a set of sentences of the language which is called a *formal theory*. In Section 4.1, we introduce the definition of a formal theory. Formal theories together with their formal consequences are the usual way of *specifying* knowledge using first-order languages.

On the other hand, the knowledge about construction or computation in a formal theory forms the implementational knowledge of the domain and with this knowledge we can construct models of the formal theory.

Specificational knowledge, in the form of an axiom system, is mainly used for deduction, induction and proof of properties about the system, while implementational knowledge is used for operation, computation and construction of a system that embodies those properties.

In this chapter, using the elementary arithmetic of $\mathbb{N}$ as an example, we will show how to specify and implement arithmetic operations to illustrate the difference between these two forms of knowledge. The formal theory Π is defined in Section 4.2. to specify arithmetic operations. It consists of ten axioms which are laws about the unary function symbol S, the binary predicate symbol $<$ and the binary function symbols $+$ and $\cdot$. These laws specify the properties of the successor function, addition and multiplication respectively. A computing system called P-kernel is defined in Section 4.3 to illustrate the implementation of arithmetical operations. P-kernel is a mathematical description of the arithmetical kernel of the language C. It consists of a series of P-procedures, each of which consists of a procedure declaration and a procedure body, where the procedure

body is composed of six statements. A computable function or decidable relation on $\mathbb{N}$ is defined as a halting P-procedure, which takes the variables of the function as input parameters and terminates with the value of the function.

Since specificational knowledge and implementational knowledge describe a domain from two different viewpoints, they are related one to the other by the same object and thus there is inevitably a relation between them. The question of whether implementational knowledge can be represented in terms of specificational knowledge, is called the *representation problem*. The complementary question of whether specificational knowledge can be constructed from implementational knowledge is called the *implementation problem*. The representation problem of P-kernel in Π is defined in Section 4.5. The representation of P-procedures and the statements of P-kernel is given in Section 4.8. A detailed outline of the proof of the theorem of representability is given in Section 4.9, and a full proof can be found in Appendix 3. The representability of elementary arithmetic is essential in the proofs of Gödel's incompleteness theorem and consistency theorem in Chapter 5.

4.1 Formal theory

A formal theory is a central concept of first-order languages. Many axiom systems in mathematics, principles of natural science, software specifications, functional descriptions of Large Scale Integrated circuits (LSI), and knowledge bases of artificial intelligence can all, to some extent, be described by formal theories of first-order languages.

Definition 4.1 (Formal theory). Suppose that Γ is a finite or countably infinite set of sentences of a first-order language $\mathscr{L}$. If Γ is consistent, then Γ is a *formal theory* of $\mathscr{L}$ or simply a *formal theory* and each sentence in Γ is an *axiom* of Γ. If Γ is a formal theory, then the sentence set

$$Th(\Gamma) = \{A \mid A \text{ is a sentence in } \mathscr{L} \text{ and } \Gamma \vdash A \text{ is provable}\}$$

is called the *closure* of Γ. If $\Gamma = \varnothing$, then

$$Th(\varnothing) = \{A \mid A \text{ is a sentence in } \mathscr{L} \text{ and } \vdash A \text{ is provable}\}$$

is the set of tautologies.

If $\mathbf{M}$ is a model of $\mathscr{L}$, and $\mathbf{M} \models \Gamma$, then we call $\mathbf{M}$ a *model* of the formal theory Γ.

A tautology is a formula of a first-order language $\mathscr{L}$, which is interpreted as true in every model of $\mathscr{L}$.

In this book we use the uppercase Greek letters such as Γ and Δ to denote formal theories and allow them to have subscripts and superscripts. A formal theory is a set of sentences that can also be expressed as a sequence of sentences. A formal theory is usually interpreted as an axiom system in a specific model. Generally an axiom system is a set of propositions without free variables. As Lemma 3.7 shows, if the sentence set Γ is inconsistent, then every sentence is a formal consequence of Γ. In this case the formal theory becomes meaningless. Hence a formal theory must be consistent.

Definition 4.1 states that the closure, $Th(\Gamma)$, is a formal theory consisting of all the formal consequence of Γ, and is a countably infinite set of sentences. Some textbooks define the closure as the formal theory, which can simplify the proofs of theorems. We purposely do not adopt such a definition in this book because $Th(\Gamma)$ is an infinite set whether Γ is finite or not. In reality, natural sciences, software systems, and knowledge bases are all finite, so the formal theories defined by Definition 4.1 are closer to reality than the closure.

Definition 4.2 ($Th(\mathbf{M})$). If $\mathbf{M}$ is a model of a first-order language $\mathscr{L}$, then the sentence set

$$Th(\mathbf{M}) = \{A \mid \ A \text{ is a sentence of } \mathscr{L}, \text{ and } \ \mathbf{M} \models A\}$$

is a *formal theory* of $\mathscr{L}$ with respect to the model $\mathbf{M}$.

$Th(\mathbf{M})$ is the set of all the sentences of $\mathscr{L}$ whose interpretations in the model $\mathbf{M}$ are true. Without causing confusion in the context, we also identify $Th(\mathbf{M})$ as the set of all true propositions of $\mathscr{L}$ with respect to $\mathbf{M}$, or simply as the set of all true propositions of $\mathbf{M}$. $Th(\mathbf{M})$ possesses the following completeness property.

Definition 4.3 (Completeness). We say that a formal theory Γ is *complete* if for every sentence A, either $\Gamma \vdash A$ or $\Gamma \vdash \neg A$ is provable.

Lemma 4.1. *For every model* $\mathbf{M}$ *of a language* $\mathscr{L}$, $Th(\mathbf{M})$ *is complete.*

Proof. Using the principle of excluded middle for domains, for every sentence A, either $\mathbf{M} \models A$ or $\mathbf{M} \models \neg A$ is true. If the former is true, by the definition of $Th(\mathbf{M})$, $A \in Th(\mathbf{M})$ holds and thus $Th(\mathbf{M}) \vdash A$ holds; otherwise we have $\neg A \in Th(\mathbf{M})$, i.e., $Th(\mathbf{M}) \vdash \neg A$ holds. □

Definition 4.4 (Independent theory). We call a formal theory Γ an *independent theory* if for every A, $A \in \Gamma$ implies

$$Th(\Gamma - \{A\}) \neq Th(\Gamma).$$

The following lemma directly follows from the definition of an independent theory.

Lemma 4.2. *Suppose that* Γ *is an independent theory and* $A \in \Gamma$. *Then neither* $\Gamma - \{A\} \vdash A$ *nor* $\Gamma - \{A\} \vdash \neg A$ *is provable.*

Proof. We prove the lemma by contradiction. If $\Gamma - \{A\} \vdash A$ is provable then $Th(\Gamma - \{A\}) = Th(\Gamma)$ holds, which contradicts Γ being an independent theory. Similarly, if $\Gamma - \{A\} \vdash \neg A$ is provable, then $\neg A \in Th(\Gamma)$ holds. Nonetheless $A \in \Gamma$ also holds. This contradicts the consistency of Γ. □

If neither $\Gamma - \{A\} \vdash A$ nor $\Gamma - \{A\} \vdash \neg A$ is provable, then $\Gamma - \{A\}$ and the formula A are *independent*. Hence an independent theory is a formal theory composed of mutually independent axioms.

The concept of independence of a formal theory and of axioms originated in mathematics and natural science. Most axiom systems in mathematics such as groups, rings,

fields and elementary arithmetic are independent. Most theories of natural science are also independent, i.e., their axioms, principles and postulates are mutually independent. In contrast, most software systems, knowledge bases and their specifications are not independent because, for software, efficiency and ease of use are more important.

4.2 Elementary arithmetic theory

We begin to learn and understand mathematics from arithmetic. First of all we abstract the concept of "natural numbers" from concrete objects and entities. Then we learn the operations of addition, subtraction, multiplication and division of natural numbers. Subsequently, we learn about fractions and rational numbers followed by irrational numbers. Afterwards our studies encompass functions, limits and calculus. The theory of natural numbers is the root of our knowledge of mathematics. In this section we introduce a formal theory in the language of elementary arithmetic $\mathscr{A}$, which is called the *theory of elementary arithmetic* [Enderton, 1972]. It is abbreviated to *elementary arithmetic* and denoted as Π. It is a formal theory about addition and multiplication of natural numbers. Elementary arithmetic is necessary to express several profound concepts of formal theories such as computability, provability, representability and incompleteness. We shall focus on computability and representability in this chapter and prove the incompleteness of elementary arithmetic Π in Chapter 5.

In the first chapter we introduced the language of elementary arithmetic $\mathscr{A}$ which contains a constant symbol 0, a unary function symbol S, two binary function symbols $+$ and $\cdot$, and a binary predicate symbol $<$.

Definition 4.5 (Elementary arithmetic theory Π). The elementary arithmetic theory Π is a formal theory composed of the following nine sentences in $\mathscr{A}$:

$$
\begin{aligned}
&\mathrm{A}_1 \quad \forall x_1 \neg (Sx_1 \doteq 0) \\
&\mathrm{A}_2 \quad \forall x_1 \forall x_2 (Sx_1 \doteq Sx_2 \rightarrow x_1 \doteq x_2) \\
&\mathrm{A}_3 \quad \forall x_1 \forall x_2 (x_1 < Sx_2 \leftrightarrow (x_1 < x_2 \vee x_1 \doteq x_2)) \\
&\mathrm{A}_4 \quad \forall x_1 \neg (x_1 < 0) \\
&\mathrm{A}_5 \quad \forall x_1 \forall x_2 (x_1 < x_2 \vee x_1 \doteq x_2 \vee x_2 < x_1) \\
&\mathrm{A}_6 \quad \forall x_1 (x_1 + 0 \doteq x_1) \\
&\mathrm{A}_7 \quad \forall x_1 \forall x_2 (x_1 + Sx_2 \doteq S(x_1 + x_2)) \\
&\mathrm{A}_8 \quad \forall x_1 (x_1 \cdot 0 \doteq 0) \\
&\mathrm{A}_9 \quad \forall x_1 \forall x_2 (x_1 \cdot Sx_2 \doteq x_1 \cdot x_2 + x_1)
\end{aligned}
$$

In Chapter 2 we introduced the model $\mathbf{N}$ of $\mathscr{A}$. Let $\mathbb{N}$ denote the set of all natural numbers with 0 being the zero of natural numbers. Let σ represent the "plus 1" function on $\mathbb{N}$ that is also called the *successor function* satisfying $\sigma(n) = n + 1$. The model

N interprets the unary function symbol S of $\mathscr{A}$ as the successor function σ, the binary function symbols $+$ and $\cdot$ of $\mathscr{A}$ as the addition and multiplication of $\mathbb{N}$ respectively, and $<$ as the "less than" relation of $\mathbb{N}$. We can verify that **N** is also a model of Π. According to Theorem 3.5, Π is consistent, so is a formal theory. $Th(\mathbf{N})$ is the set composed of all the sentences of $\mathscr{A}$ that are true in **N**.

The axioms A_1 and A_2 of the theory Π describe the properties of the unary function symbol S. In the model **N**, the axiom A_1 is interpreted as "0 is not a successor of any natural number." The axiom A_2 is interpreted as "the successor function is an injective function."

The axioms A_3 to A_5 describe the properties of the binary predicate symbol $<$. In the model **N**, the axiom A_3 is interpreted as "the natural number x_1 is smaller than the successor of the natural number x_2 if and only if x_1 is smaller than x_2 or x_1 is equal to x_2". The axiom A_4 is interpreted as "each natural number is not less than zero" and A_5 is interpreted as "for any two natural numbers, either they are equal or one is smaller than the other."

The axioms A_6 and A_7 describe the properties of the binary function symbol $+$. In the model **N**, the axiom A_6 is interpreted as "any natural number plus 0 equals itself." The axiom A_7 describes the relation between addition and the successor function: "the sum of the natural number x_1 and the successor of the natural number x_2 equals the successor of the sum of these two numbers."

The axioms A_8 and A_9 describe the properties of the binary function symbol $\cdot$. In the model **N**, the axiom A_8 is interpreted as "the product of any natural number and 0 equals 0." A_9 describes the relation between multiplication, addition and the successor function. It can be interpreted as "the product of the natural number x_1 and the successor of the natural number x_2 equals the product of these two numbers plus x_2."

Peano was the first person to study the axiomatization of elementary arithmetic. His theory of arithmetic only includes the axioms on the successor function and mathematical induction. The formal description of mathematical induction in first-order languages is the following axiom, which is actually a schema since A denotes any formula:

$$\mathrm{A}_{10} \quad (A[0/x_1] \wedge (A[S^n0/x_1] \rightarrow A[S^{n+1}0/x_1])) \rightarrow \forall x_1 A(x_1)$$

A_{10} is the last axiom (schema) of the theory Π and is interpreted as: in the model **N**, if $A[0]$ is true, and if $A[S^n0]$ being true implies $A[S^{n+1}0]$ being true, then $\forall x_1 A(x_1)$ is true.

In this book we treat 0 as a special natural number as well as the first element of $\mathbb{N}$ with respect to the relation $<$. In Chapter 1 we used

$$S^0 0 \text{ to denote } 0,\ S^{n+1}0 \text{ to denote } S(S^n 0),\ \text{and thus } S^n 0 \text{ for } \underbrace{SS\cdots S}_{n}0.$$

Here S^n0 is just an abbreviation instead of a term of $\mathscr{A}$. Its superscript n denotes making the successor operation for n times with $n \in \mathbb{N}$ being a natural number. This kind of notation will be frequently used in this chapter and Chapter 5.

For the convenience of discussion, we can introduce the non-negative subtraction symbol "$-$" on Π:

$$A_{11} \quad \forall x_1 \forall x_2 \forall x_3((x_2 < x_1) \rightarrow ((x_3 \doteq x_1 - x_2) \leftrightarrow (x_2 + x_3 \doteq x_1)))$$

$$A_{12} \quad \forall x_1 \forall x_2 \forall x_3(\neg(x_2 < x_1) \rightarrow ((x_3 \doteq x_1 - x_2) \leftrightarrow (x_3 \doteq 0)))$$

Hereafter all the $-$ in this chapter refer to the non-negative subtraction defined by the above two axioms. Using this method we can also introduce other function symbols on Π such as division and exponentiation.

The terms and formulas of $\mathscr{A}$ in this section are symbol strings. They are interpreted as natural numbers, functions and propositions of **N** with specific meanings in **N**. In particular, the propositions of **N** describe the relationships between addition, subtraction, multiplication and the successor function on the set of natural numbers. Furthermore, the natural number system $\mathbb{N}$ contains much richer mathematical theories such as those dealing with series and polynomials.

In Section 4.3 we will introduce a computing system defined on $\mathbb{N}$, called *P-kernel*. This system will be used to define decidable relations and computable functions on $\mathbb{N}$. The computing system of *P*-kernel can be viewed as a part of number theory. In Section 4.3 we shall expand the model **N** so that it contains the concepts necessary to define *P*-kernel. To fully understand this chapter, the reader needs to discriminate between terms and formulas of the first-order language $\mathscr{A}$, their corresponding functions and propositions in the model **N**, and the concepts, functions and propositions defined on $\mathbb{N}$ but not in the model **N**. The latter are called the *meta-language environment* of the first-order language $\mathscr{A}$.

4.3 *P*-kernel on $\mathbb{N}$

What are computable functions? Most people who have experience of programming would answer: a function $f(x)$ is computable if a programming language such as C can be used to design a function $F(x)$ with x being its formal parameter such that the execution of the **call** statement $F(a)$ on a computer halts after finitely many steps and returns the value $f(a)$.

Following this idea, we define a computing system on the set $\mathbb{N}$ of natural numbers, which is called *P-kernel*. We will introduce *P-procedures* in *P*-kernel to define computable functions and decidable relations. These concepts all are part of the implementational knowledge of number theory.

Definition 4.6 (*P*-procedure). *P*-kernel is a computing system defined on the set $\mathbb{N}$ of natural numbers. A *P*-procedure is an entity and is the 'first class citizen' of *P*-kernel. Each *P*-procedure is composed of a procedure declaration and a procedure body. The former consists of a procedure name, variable declarations and sub-procedure declarations, while the latter is composed of statements.

The procedure name has the form

$$\textbf{procedure } F(x_1, \ldots, x_k, x_{k+1})$$

with F being the procedure name and $x_1, \ldots, x_k, x_{k+1}$ being the formal parameters of the procedure. The formal parameters $x_1, \ldots, x_k$ are called *input parameters*, whose number

can be 0 or finite; x_{k+1} is called the *output parameter* and is used to store the results of computations. When the P-procedure is called, the formal parameters $x_1,\ldots,x_k$ are assigned by real parameters and x_{k+1} is assigned the value 0. The procedure body is executed after these assignments.

Each procedural declaration may contain finitely many local variable declarations:

$$x_{k+2},\ldots,x_{k+l};$$

Local variables should have the same form as formal parameters but they should not share the same name. They are used only in statements of the procedure body to store intermediate results of computations. When the variable x_i stores a natural number m, the *value* of the variable x_i is m. Formal parameters are used in the procedure body as variables. Except for formal parameters, all other variables take the value 0 by default before executions of the procedure body.

For convenience, we also use x,y,z with superscripts or subscripts to denote local variables and formal parameters. Each procedural declaration may contain finitely many (including 0) sub-procedure declarations that have the same form and structure as the procedure defined above.

Definition 4.7 (Statements of P-kernel). A P-procedure body allows for six different *statements*. They are *assignment statement*, *printing statement*, ***if*** *statement*, *sequential statement*, ***while*** *statement*, and ***call*** *statement*. The first two statements are called *atomic statements* whereas the other four are called *composite statements*. Each statement executes a definite computation. Hereafter we use the lowercase Greek letter α to denote statements and allow α to have subscripts or superscripts.

(1) Assignment statement:

$$x := e$$

where e is an arithmetic expression. Any natural number m and variable x, as well as the addition $+$, subtraction $-$ and product $\times$ of any two arithmetic expressions, are arithmetic expressions. The Backus normal form of arithmetic expressions is:

$$e ::= m \mid x \mid e_1+e_2 \mid e_1-e_2 \mid e_1\times e_2$$

The execution of the above assignment statement evaluates the arithmetic expression e first and then stores the value of e in x.

(2) Printing statement:

$$\textbf{print } x$$

This statement prints the content stored in the variable x.

(3) **If** statement:

$$\textbf{if } 0 < x \textbf{ then } \alpha_1 \textbf{ else } \alpha_2$$

This statement first checks the value stored in x. If the value is bigger than 0, then the statement α_1 is executed; otherwise the statement α_2 is executed.

(4) Sequential statement:

$$\alpha_1;\ \alpha_2$$

This statement indicates that a statement sequence with the adjacent statements separated by ";" is also a statement. The sequential statement first executes the statement α_1 and then executes the statement α_2 after the execution of α_1 is terminated.

(5) **While** statement:

$$\textbf{while}\ \ 0 < x\ \ \textbf{do}\ \ \alpha$$

The statement α is called the *loop body*. The **while** statement is executed as follows: it checks x and if x is bigger than 0, then the sequential statement α; **while** $0 < x$ **do** α is executed; otherwise the execution of the **while** statement is completed.

(6) **Call** statement:

$$F(m_1,\ldots,m_k,x_{k+1})$$

Here F is the name of a P-procedure with $k+1$ formal parameters. Its first k formal parameters are input parameters, whereas the last formal parameter x_{k+1} is the output parameter. The **call** statement first takes the natural numbers $m_1,\ldots,m_k$ and 0 as real parameters and assigns them to the formal parameters $x_1,\ldots,x_k$, and x_{k+1} of the procedure F respectively. Then it executes the procedure body of F. When the execution of the procedure body begins, the value of x_{k+1} is 0; after the execution terminates, the value stored in x_{k+1} is the output value of the procedure, which is the *computational result* of the procedure. The above mechanism of procedure call is named as *call by value*.

Definition 4.8 (P-procedure body). A procedure body is a finite sequence of statements:

$$\textbf{begin}\ \ \alpha\ \ \textbf{end}$$

The execution of procedure body starts from its first statement after the keyword **begin**, goes through the statements in order of their occurrence and terminates when it meets the keyword **end**.

Note that Davis [1958] and Ebbinghaus [1994] proved that to implement the statement $x := e$, it is sufficient to define two assignment statements $x := x+1$ and $x := x-1$ plus the other five statements in Definition 4.7. Our purpose in introducing the general form $x := e$ is to be consistent with the daily-used programming languages.

Strictly speaking, the P-procedure defined in this section is different from the functions defined by programming languages such as C. Every programming language is a formal language with strict syntactic rules, whereas every P-procedure is a mathematical mechanism defined on the set $\mathbb{N}$. The purpose of introducing the P-kernel system is to define what a computational procedure is, and the mechanism does not have those strict and detailed syntactic rules of programming languages.

For example, it does not prescribe the upper limits for natural numbers. Another example is that the variable x_k with the subscript is a local variable of a P-procedure, but it violates the syntactic rules of C and Pascal. Hence x_k is a variable taking values

from natural numbers instead of a local variable of the languages C and Pascal. In fact, a P-procedure can be viewed as a mathematical model of a "program segment." P-kernel is the core of the "computational mechanism" that every programming language should contain. The P-kernel can be used to define computable functions and decidable relations on the set $\mathbb{N}$.

Definition 4.9 (Halting P-procedures). Let $F(x_1,\ldots,x_k,x_{k+1})$ be a P-procedure. If for each group of real parameters $m_1,\ldots,m_k$, there exists a natural number n such that the **call** statement $F(m_1,\ldots,m_k,x_{k+1})$ terminates after finitely many steps of execution with the return value of x_{k+1} being n, then we say that F is a *halting procedure* and denote it as

$$F:\ m_1,\ldots,m_k \to n.$$

If we are only interested in whether the P-procedure halts or not, we can also denote it as

$$F:\ m_1,\ldots,m_k \to \square.$$

If there exists a group of real parameters $m_1,\ldots,m_k$ such that the execution of the **call** statement $F(m_1,\ldots,m_k,x_{k+1})$ never terminates, then we say that F is a *non-halting procedure* and denote it as

$$F:\ m_1,\ldots,m_k \to \perp.$$

Hereafter we shall use $f(x_1,\ldots,x_k)$ to denote f being a k-ary function with domain $\mathbb{N}\times\cdots\times\mathbb{N}$ and range $\mathbb{N}$. We use $f(m_1,\ldots,m_k)$ to denote the value of the function f at the point $(m_1,\ldots,m_k)$.

Definition 4.10 (Computable functions). Let $f(x_1,\ldots,x_k)$ be a k-ary function on $\mathbb{N}$. We call f a *computable function* on $\mathbb{N}$ if there exists a P-procedure $F(x_1,\ldots,x_k,x_{k+1})$ on $\mathbb{N}$ such that

$$F:\ m_1,\ldots,m_k \to f(m_1,\ldots,m_k)$$

holds for all real parameters $(m_1,\ldots,m_k)$.

Hereafter we shall use $r(x_1,\ldots,x_k)$ to denote r being a k-ary relation with domain $\mathbb{N}\times\cdots\times\mathbb{N}$ and range $\{1,\ 0\}$. We use $r(m_1,\ldots,m_k)$ to denote the value of the relation r at the point $(m_1,\ldots,m_k)$. The value being 1 means that the relation r holds at the point $(m_1,\ldots,m_k)$, whereas the value being 0 means that the relation r does not hold at the point $(m_1,\ldots,m_k)$.

Definition 4.11 (Decidable relations). Let $r(x_1,\ldots,x_k)$ be a k-ary relation on $\mathbb{N}$. We call r a *decidable relation* on $\mathbb{N}$ if there exists a P-procedure $R(x_1,\ldots,x_k,x_{k+1})$ on $\mathbb{N}$ such that for any $m_1,\ldots,m_k$, if the relation $r(m_1,\ldots,m_k)$ holds, then

$$R:\ m_1,\ldots,m_k \to 1$$

holds; if $r(m_1,\ldots,m_k)$ does not hold, then

$$R:\ m_1,\ldots,m_k \to 0$$

holds.

4.4 Church-Turing thesis

In Section 4.3 we defined computability by using the concept of halting *P*-procedure. In fact, this is only one of dozens of approaches to defining computability. We might call it *P-computability*. Historically, many scholars have propounded different definitions of computability. For instance, Gödel introduced *recursive functions* and defined a function on $\mathbb{N}$ as computable if it is a recursive function [Shoenfield, 1967]. Recursive functions are defined by structural induction as functions on $\mathbb{N}$:

R1. $+$, $\cdot$, $<$, I_i^n are recursive functions. Here $+$ and $\cdot$ denote the addition and multiplication respectively, $<$ denotes the "less than" relation, and I_i^n denotes taking the ith component m_i of an n-tuple array $(m_1,\ldots,m_n)$.

R2. If $G(m_1,\ldots,m_k)$ and $H_i(n)$, $i=1,\ldots,k$, are recursive functions and the function $F(n)$ is defined by

$$F(n) = G(H_1(n),\ldots,H_k(n)),$$

then $F(n)$ is a recursive function.

R3. If $G(m,n)$ is a recursive function and for any given natural number m there exists a natural number x such that $G(m,x)=0$, then

$$F(m) = \mu x(G(m,x)=0)$$

is a recursive function. In the above equality $\mu x\ (\cdots\ x\ \cdots)$ denotes the smallest value of x such that $(\cdots\ x\ \cdots)$ is true. Hence $F(m)$ is, for a given m, the smallest value of x such that $G(m,x)=0$ is true, i.e., $F(m)=\min\{x \mid G(m,x)=0\}$.

Turing introduced the Turing machine to define a computable function as one that can be computed by a Turing machine [Turing, 1936].

Church established the λ-calculus with which he defined his concept of computability [Church, 1941].

There are other definitions of computability such as the definition by register machines [Ebbinghaus et al, 1994]. The register machine can be regarded as a mathematical model of the assembly language.

The above definitions of computability reflect the experience and intuition of the authors about "computational mechanism" using different mathematical tools. One could ask whether these definitions are mathematically equivalent and whether they reveal the mathematical essence of computability. Researchers have proved theoretically that the above existent definitions of computability are mutually equivalent.

The idea of the proof can be briefly summarized as follows: proving that different computing systems are mutually equivalent amounts to proving that they can be mutually transformed into each other. In what follows we provide a method to prove this for recursive functions, *P*-kernel, register machines and Turing machines. Let us start with Gödel's recursive functions. Since, for each recursive function, we can design its corresponding halting *P*-procedure, the computable functions defined by recursive functions are also computable in terms of *P*-kernel. Further, since each halting *P*-procedure can be

implemented in assembly language, every computable function in terms of P-kernel is a computable function defined by register machines as well. Next, since register machines are Turing machines of a special kind, computable functions defined by register machines are also computable functions defined by Turing machines.

Finally, since we can prove that every Turing machine can be specified by a recursive function, every computable function defined by Turing machines is also computable in terms of recursive functions. Through this series of transformations, the equivalence of the above definitions is proved. By using a similar method we can prove that the λ-calculus and Turing machines are mutually equivalent.

Church and Turing both realised the importance of equivalence of different definitions of computability and they proposed a thesis, of which the following is a slight variation:

Principle 4.1 (Church-Turing thesis). ***Every computable function or decidable relation is recursive.***

Since the time when this thesis was propounded, people have realized that it is not a theorem because they cannot exhaust all the definitions of computability. Thus it seems that there is no mathematical proof for the Church-Turing thesis and it is just an assumed principle of computability.

The Church-Turing thesis allows us to use any computing system to define computability provided that the definition is equivalent to that using recursive functions. In fact, the different definitions of computability discussed in this section have already been adopted in various textbooks. Today, with the prevalence of computers, computability is no longer an abstract concept. Thus it is more intuitive and intelligible for us to use P-kernel in this book to define computability.[1]

4.5 Problem of representability

An important property of computable functions is that they can be represented in the formal theory Π, i.e., for every computable function $f(x_1,\ldots,x_k)$ defined on $\mathbb{N}$, there exists a formula $A_f(x_1,\ldots,x_k,x_{k+1})$ of $\mathscr{A}$ such that for any natural numbers $m_1,\ldots,m_k$ and m_{k+1}:

(1) if $f(m_1,\ldots,m_k)=m_{k+1}$ is true, then $\Pi \vdash A_f[m_1,\ldots,m_k,m_{k+1}]$ is provable;

(2) if $f(m_1,\ldots,m_k)\neq m_{k+1}$ is true, then $\Pi \vdash \neg A_f[m_1,\ldots,m_k,m_{k+1}]$ is provable.

The formula $A_f(x_1,\ldots,x_k,x_{k+1})$ is called the *representation* of the function $f(x_1,\ldots,x_k)$ in Π. The above property can be proved as a theorem. As in Definition 4.11, decidable relations are equivalent to computable functions taking either the value 0 or 1. Thus the

[1] The Church-Turing thesis motivated people to prove the mutual equivalence of different definitions of computability. The ideas and methods developed during these investigations have been widely used by computer scientists to study computational complexity, leading to such important results as the concept of NP-complete problems and the question of "whether P = NP." These are significant both theoretically and practically [Garey and Johnson, 1979, Hopcroft et al, 2006].

representability of computable functions in Π implies the representability of decidable relations in Π.

Definitions 4.15 and 4.16 in Section 4.9 will formally define the representability in Π of functions and relations defined on $\mathbb{N}$. Theorems 4.2 and 4.3 in Section 4.9 will further demonstrate the representability in Π of computable functions and decidable relations. Since *P*-kernel is defined by structural induction, Theorems 4.2 and 4.3 can both be proved by structural induction. The outline of the proof is as follows.

(1) Since each computable function is defined by a halting *P*-procedure, the key of the proof is to find a logical formula of Π representing the halting *P*-procedure.

(2) Since the computational behavior of each *P*-procedure is determined by the procedure body, the problem is further reduced to finding a logical formula of Π representing the procedure body.

(3) Since the procedure body is composed of statements, for each statement we have to find a logical formula of Π as its representation.

(4) Hence we need to define the computational behavior of each statement. It is well known that a configuration of a computer is determined by the current state of the memory as well as by the current statement. Thus the execution of a statement converts the current configuration to a new configuration. Therefore the computational behavior of a statement can be defined by the transition between the two configurations.

(5) If we can find logical formulas of Π representing the transition between states and configurations respectively, then we can define logical formulas of Π representing statements by structural induction, with which we can further prove the representability of procedure bodies.

In order to prove the representability theorem rigorously, we have to consider every possible structure of procedure bodies, since provability and computability are defined on first-order languages and the set of natural numbers respectively. As a result, the proof has to be meticulous and lengthy. Readers may refer to Appendix 3 for the detailed proof. Although we do not give the complete and rigorous proof of the representability theorem in this chapter, we shall provide a detailed road map for it in the following sections. We formally define those concepts that are necessary in the proof of the theorems and illustrate them through examples. We also state accurately every lemma needed in the proof.

4.6 States of *P*-kernel

We define states and their representations in Π in this section. The current state of the memory is determined by the current value of each variable. From the mathematical perspective, each state is a map from the set of variables to $\mathbb{N}$. Both the configurations and the states are different at different steps in the execution of the program.

Definition 4.12 (State). Let F be a P-procedure with variable set $\mathbf{V} = \{x_1, \ldots, x_k, x_{k+1}\}$, where $\{x_1, \ldots, x_k\}$ are the input parameters of F, and x_{k+1} is the output parameter. Each *state* σ is a mapping from the variable set $\mathbf{V}$ to $\mathbb{N}$, i.e.,

$$\sigma : \mathbf{V} \longrightarrow \mathbb{N}.$$

The form

$$[x_i]_\sigma = m_i \text{ or } x_i \mapsto m_i$$

is used to denote m_i being the value of the variable x_i in the state σ with $m_i \in \mathbb{N}$, $1 \leqslant i \leqslant k+1$.

Let e be a given expression of P-kernel and σ be a state. $[e]_\sigma$ denotes the value of e in the state σ, and is defined inductively as follows:

$$\begin{aligned}
[m]_\sigma &= m; \\
[x_i]_\sigma &= m_i; && \text{if } m_i \text{ is stored in } x_i \\
[e_1 + e_2]_\sigma &= [e_1]_\sigma + [e_2]_\sigma; \\
[e_1 - e_2]_\sigma &= [e_1]_\sigma - [e_2]_\sigma, && \text{if } [e_1]_\sigma \geqslant [e_2]_\sigma; \\
[e_1 - e_2]_\sigma &= 0, && \text{if } [e_1]_\sigma < [e_2]_\sigma; \\
[e_1 \cdot e_2]_\sigma &= [e_1]_\sigma \cdot [e_2]_\sigma.
\end{aligned}$$

For convenience, $\sigma[x_i \mapsto [e]_\sigma]$ is also a state defined by

$$[y]_{\sigma[x_i \mapsto [e]_\sigma]} = \begin{cases} [e]_\sigma, & \text{if } y = x_i, \\ [y]_\sigma, & \text{if } y \neq x_i. \end{cases}$$

A state of a P-procedure is denoted as

$$(x_1 \mapsto m_1, \ldots, x_{k+1} \mapsto m_{k+1}).$$

The following lemma can be proved by mathematical induction.

Lemma 4.3. *Let m, n and k be natural numbers.*

(1) *If $m = n$ holds, then $\Pi \vdash S^m 0 \doteq S^n 0$ is provable.*

(2) *If $m \neq n$ holds, then $\Pi \vdash \neg(S^m 0 \doteq S^n 0)$ is provable.*

(3) *If $m + n = k$ holds, then $\Pi \vdash S^m 0 + S^n 0 \doteq S^k 0$ is provable.*

(4) *If $m + n \neq k$ holds, then $\Pi \vdash \neg(S^m 0 + S^n 0 \doteq S^k 0)$ is provable.*

(5) *If $m - n = k$ holds, then $\Pi \vdash S^m 0 - S^n 0 \doteq S^k 0$ is provable.*

(6) *If $m - n \neq k$ holds, then $\Pi \vdash \neg(S^m 0 - S^n 0 \doteq S^k 0)$ is provable.*

(7) *If $m \cdot n = k$ holds, then $\Pi \vdash S^m 0 \cdot S^n 0 \doteq S^k 0$ is provable.*

(8) *If $m \cdot n \neq k$ holds, then $\Pi \vdash \neg(S^m 0 \cdot S^n 0 \doteq S^k 0)$ is provable.*

According to Definition 4.12, we can obtain the representation in Π of the value of the expression e in the state σ.

Definition 4.13 (Representation of the values of expressions in Π). Let σ be a state. Let $Tr([e]_\sigma)$ denote the representation of the value of the expression e in the state σ. $Tr([e]_\sigma)$ is defined inductively as follows:

(1) $Tr([m]_\sigma) = S^m 0$;

(2) $Tr([x_i]_\sigma) = S^{m_i} 0$ if $[x_i]_\sigma = m_i$ with $m_i \in \mathbb{N}$;

(3) $Tr([e_1 * e_2]_\sigma) = Tr([e_1]_\sigma) * Tr([e_2]_\sigma)$ with $*$ being $+$, $-$ or $\cdot$.

According to Definition 4.13 and Lemma 4.3, the following lemma holds.

Lemma 4.4. *Let e be an arithmetic expression of the P-procedure and σ be a state. Then the following sequent is provable:*

$$\Pi \vdash Tr([e]_\sigma) \doteq S^{[e]_\sigma} 0.$$

Proof. The lemma is proved by structural induction on the arithmetic expression e. □

4.7 Operational calculus of *P*-kernel

In this section we define the operational behavior of each statement of the *P*-kernel by a transition rule between configurations. All such rules constitute an operational calculus of the *P*-kernel.

Suppose that α is a statement in a *P*-procedure body and is currently under execution. Let σ be the current state during the execution of the statement. It is well known that the current configuration of each procedure under execution is completely determined by the current state and the current statement. Thus the pair

$$\langle \alpha, \sigma \rangle$$

is called the *current configuration* of the *P*-procedure or a *configuration* for short.

Generally speaking, the execution of a given statement α in a state σ can have two different kinds of status.

(1) The execution of the first statement of α terminates with a new state σ' generated and there is another statement α' that needs to be executed in the state σ'. Hence the new configuration after the execution of the statement is $\langle \alpha', \sigma' \rangle$.

For example α is $x_1 := e;\ \alpha'$. Then σ' is $\sigma[x_i \mapsto [e]_\sigma]$. Under such circumstances, the execution of the statement α in the state σ can be described by the following *transition* between the two configurations

$$\langle \alpha, \sigma \rangle \longrightarrow \langle \alpha', \sigma' \rangle.$$

We call the above transition a *first class transition*. Here $\longrightarrow$ stands for the transition.

(2) The execution of the statement terminates generating a new state σ' but with no other statement needing to be executed. Under such circumstances, the execution of the statement α in the state σ can be described by the transition

$$\langle \alpha, \sigma \rangle \longrightarrow \sigma'.$$

We call this transition a *second class transition*.

In the following, we introduce a transition rule to describe the execution of each statement of *P*-kernel.

(1) **The Assignment Statement.** The execution of the assignment statement $x_i := e$ in the state σ is described by the transition

$$\langle x_i := e,\ \sigma \rangle \longrightarrow \sigma[x_i \mapsto [e]_\sigma],$$

which is a second class transition. When the execution of the assignment statement in the state σ terminates, a new state $\sigma[x_i \mapsto [e]_\sigma]$ is generated with the value of the variable x_i changed to $[e]_\sigma$ and the values of the other variables unchanged.

(2) **The If Statement.** With the state σ, the execution of the **if** statement **if** $0 < x_i$ **then** α_1 **else** α_2 is described by two second class transitions as follows:

$$\text{if } 0 < [x_i]_\sigma, \text{ then } \langle \textbf{if } 0 < x_i \textbf{ then } \alpha_1 \textbf{ else } \alpha_2,\ \sigma \rangle \longrightarrow \langle \alpha_1,\ \sigma \rangle;$$
$$\text{if } 0 \geqslant [x_i]_\sigma, \text{ then } \langle \textbf{if } 0 < x_i \textbf{ then } \alpha_1 \textbf{ else } \alpha_2,\ \sigma \rangle \longrightarrow \langle \alpha_2,\ \sigma \rangle.$$

The first action of the execution of the **if** statement is to evaluate the variable x_i in the state σ. The first transition shows that if $0 < [x_i]_\sigma$ holds, then the new configuration $\langle \alpha_1, \sigma \rangle$ is generated, i.e., the next statement to be executed is α_1 with unaltered state σ. The second transition shows that if $0 < [x_i]_\sigma$ does not hold, then another new configuration $\langle \alpha_2, \sigma \rangle$ is generated, i.e., the next statement to be executed is α_2 with unaltered state σ.

(3) **The While Statement.** In the state σ, the execution of the **while** statement **while** $0 < x_i$ **do** α is also described by two second class transitions as follows:

$$\text{if } 0 < [x_i]_\sigma, \text{ then } \langle \textbf{while } 0 < x_i \textbf{ do } \alpha,\ \sigma \rangle \longrightarrow \langle \alpha;\ \textbf{while } 0 < x_i \textbf{ do } \alpha,\ \sigma \rangle;$$
$$\text{if } 0 \geqslant [x_i]_\sigma, \text{ then } \langle \textbf{while } 0 < x_i \textbf{ do } \alpha,\ \sigma \rangle \longrightarrow \sigma.$$

The first action of the execution of the **while** statement is to evaluate variable x_i in the state σ. The first transition indicates that if $0 < [x_i]_\sigma$ holds, then the **while** statement is executed and the new configuration $\langle \alpha; \textbf{while } 0 < x_i \textbf{ do } \alpha,\ \sigma \rangle$ is generated, i.e., the statement to be executed is a sequential statement that will execute the loop body α first before it executes the **while** statement **while** $0 < x_i$ **do** α again; if $0 < [x_i]_\sigma$ does not hold, then the execution of the **while** statement terminates.

(4) **The Call Statement.** Suppose that a procedure $F(x_1,\ldots,x_k,x_{k+1})$ has been declared and the procedure body is α. In the state σ, the execution of the statement $F(m_1,\ldots,m_k,x_{k+1})$ is described by the transition

$$\langle F(m_1,\ldots,m_k,x_{k+1}),\ \sigma\rangle \longrightarrow \langle \alpha,\ \sigma[x_1 \mapsto m_1,\ldots,x_k \mapsto m_k, x_{k+1} \mapsto 0]\rangle.$$

This transition shows that the execution of $F(m_1,\ldots,m_k,x_{k+1})$ in the state σ produces a new configuration,

$$\langle \alpha,\ \sigma[x_1 \mapsto m_1,\ldots,x_k \mapsto m_k, x_{k+1} \mapsto 0]\rangle,$$

where the next statement to be executed is the first statement of the procedure body, in the new state it will take real parameters $m_1,\cdots,m_{k+1}$ as the current values of the variables $x_1,\ldots,x_k$ and x_{k+1} respectively.

(5) **The Sequential Statement.** In the state σ, the execution of the sequential statement $\alpha_1;\alpha_2$ is described by the following two second class transitions:

$$\text{if } \langle \alpha_1,\ \sigma\rangle \longrightarrow \sigma', \text{ then } \langle \alpha_1;\alpha_2,\ \sigma\rangle \longrightarrow \langle \alpha_2,\ \sigma'\rangle;$$
$$\text{if } \langle \alpha_1,\ \sigma\rangle \longrightarrow \langle \alpha_1',\ \sigma'\rangle, \text{ then } \langle \alpha_1;\alpha_2,\ \sigma\rangle \longrightarrow \langle \alpha_1';\alpha_2,\ \sigma'\rangle.$$

The first transition indicates that if the state changes to σ' after the execution of the statement α_1 in the state σ terminates, then the new configuration generated by the execution of the sequential statement $\alpha_1;\alpha_2$ in the state σ is $\langle \alpha_2,\ \sigma'\rangle$.

The second transition shows that if a new configuration $\langle \alpha_1',\ \sigma'\rangle$ is generated after the statement α_1 in the state σ is executed, then the new configuration generated by the execution of the sequential statement $\alpha_1;\alpha_2$ in the state σ is $\langle \alpha_1';\alpha_2,\ \sigma'\rangle$.

In summary, the execution of each statement of *P*-kernel under a given state is described by one or two transition rules. The set of all these transition rules form an *operational calculus* of the *P*-kernel which is also called its structural operational semantics. It was propounded by Plotkin in the late 1970s and began to be used in the early 1980s in the investigations of programming theory, especially concurrent programming theory and typed programming [Milner, 1980; Plotkin, 1981; Li 1982]. Plotkin called the operational calculus *structural operational semantics*. Here the word "structural" means that the execution of an atomic statement is directly determined by a transition between configurations, whereas the execution of a composite statement is determined by the transitions of statements constituting the composite statement, i.e., the operational semantics of a composite statement is determined by the structure of the statement.

4.8 Representations of statements

In this section we shall provide a detailed road map for the proof of the representability theorem for halting *P*-procedures with two input parameters. This proof can be extended

to include halting P-procedures with more input parameters and local variables. In order to make the discussions and proofs in this section more intelligible, we make the following assumptions.

(1) The execution of a statement usually cannot be completed in one step. For instance, the execution of a **while** statement may generate several states such that the same variable of the while body takes different values in different states. Hence to describe the computational behavior of a statement, it is necessary to discriminate between each variable of a P-procedure, the same variable in different states of the execution, and its values in different states of the execution.

For instance, the variable x_1 in the ith configuration can be denoted as x_1^i. Let $\tau^i = \{x_1^i, x_2^i, x_3^i\}$ denote the variable set in the ith step of the execution of a statement. We call x_1^i, x_2^i, x_3^i *state variables*. For each i, the state variables x_1^i, x_2^i, x_3^i describe the state of the variables x_1, x_2, x_3 in the ith step of the execution of the statement.

In particular, we call the state $\sigma = (x_1 \mapsto m_1, x_2 \mapsto m_2, x_3 \mapsto 0)$, before the execution of the statement α, the *initial state*, whereas the state $\sigma' = (y_1 \mapsto n_1, y_2 \mapsto n_2, y_3 \mapsto n_3)$, after the termination of the execution of α, is called the *terminating state*. We call $\tau = \{x_1, x_2, x_3\}$ and $\tau' = \{y_1, y_2, y_3\}$ the *variable set of the initial state* and *variable set of the terminating state* respectively.

(2) The logical formula

$$\mathrm{cond}(A,\ B,\ C)\ =\ (A \to B) \wedge ((\neg A) \to C)$$

will be used later. Its meaning is *if A then B, otherwise C*.

(3) When discussing the representability of statements in Π, we prescribe that the variables and state variables in $\mathscr{A}$ share the same names and symbols with the corresponding variables and state variables in $\mathbb{N}$.

Hereafter we shall use in $\mathscr{A}$ the formula

$$T_\alpha(\tau, \tau')$$

to denote the representation of the statement α in Π. The set of its free variables is $\tau \cup \tau'$ with the variable set of the initial state being $\tau = \{x_1, x_2, x_3\}$ and the variable set of the terminating state being $\tau' = \{y_1, y_2, y_3\}$.

(1) Representation of the assignment statement

Let us first look at an example.

Example 4.1 (Representation of the assignment statement). Consider an assignment statement $\alpha:\ x_3 := x_1 + x_2$. Suppose that the initial state σ before the execution of the statement is $(x_1 \mapsto m_1,\ x_2 \mapsto m_2,\ x_3 \mapsto 0)$ with the initial state variable set $\tau = \{x_1, x_2, x_3\}$, and after the execution of the statement α, the terminating state becomes σ' with the terminating state variable set being $\tau' = \{y_1, y_2, y_3\}$. Since

$$[x_1 + x_2]_\sigma = m_1 + m_2,$$

according to the operational rule of the assignment statement in Section 4.7, we have

$$\sigma' = (y_1 \mapsto m_1,\ y_2 \mapsto m_2,\ y_3 \mapsto (m_1 + m_2)).$$

Generally speaking, if we use state variables to define the representation of the statement α, then $T_{x_3:=x_1+x_2}(\tau,\ \tau')$ should be

$$y_1 \doteq x_1 \wedge y_2 \doteq x_2 \wedge y_3 \doteq x_1 + x_2.$$

The set of free variables contained in $T_{x_3:=x_1+x_2}(\tau,\ \tau')$ is $\{x_1,\ x_2,\ x_3,\ y_1,\ y_2,\ y_3\}$. If we substitute the state variables by the representations of the values of the variables in the two states respectively, we shall obtain

$$T_{x_3:=x_1+x_2}(\tau,\ \tau')[S^{m_1}0, S^{m_2}0, 0, S^{m_1}0, S^{m_2}0, S^{m_1+m_2}0].$$

This is

$$(S^{m_1}0 \doteq S^{m_1}0)\ \wedge\ (S^{m_2}0 \doteq S^{m_2}0)\ \wedge\ (S^{m_1+m_2}0 \doteq S^{m_1}0 + S^{m_2}0).$$

We can prove that $T_{x_3:=x_1+x_2}(\tau,\ \tau')$ is a representation in Π of the statement $x_3 := x_1 + x_2$. In fact, if $n_3 = m_1 + m_2$, then

$$\Pi \vdash T_{x_3:=x_1+x_2}(\tau,\ \tau')[S^{m_1}0, S^{m_2}0, 0, S^{m_1}0, S^{m_2}0, S^{n_3}0] \text{ is provable;}$$

if $n_3 \neq m_1 + m_2$, then

$$\Pi \vdash \neg T_{x_3:=x_1+x_2}(\tau,\ \tau')[S^{m_1}0, S^{m_2}0, 0, S^{m_1}0, S^{m_2}0, S^{n_3}0] \text{ is provable.}$$

In the above sequent $[S^{m_1}0, S^{m_2}0, 0, S^{m_1}0, S^{m_2}0, S^{n_3}0]$, which follows the formula $T_{x_3:=x_1+x_2}(\tau, \tau')$, stands for (see Definition 1.7)

$$[S^{m_1}0/x_1, S^{m_2}0/x_2, 0/x_3, S^{m_1}0/y_1, S^{m_2}0/y_2, S^{n_3}0/y_3].$$

Obviously, if the formula $x_1 \doteq S^{m_1}0 \wedge x_2 \doteq S^{m_2}0 \wedge x_3 \doteq 0$ holds before the execution of the statement $x_3 := x_1 + x_2$, then the formula $y_1 \doteq x_1 \wedge y_2 \doteq x_2 \wedge y_3 \doteq x_1 + x_2$ holds after the execution of the statement terminates. The first formula is called the *pre-condition* of the statement α and the second formula is called the *post-condition* of the statement α. The concepts of pre-condition and post-conditions of a statement were introduced by Hoare and Dijkstra [Hoare, 1969, Dijkstra, 1976].

We see from this example that the representation of the assignment statement $x_3 := x_1 + x_2$ uses $y_3 \doteq x_1 + x_2$ which is a representation of the expression $x_1 + x_2$ with respect to state variables. Its definition is as follows.

Definition 4.14 (Representation of expressions). Let $\tau_z = \{z_1, z_2, z_3\}$ stand for the set of state variables and $[e]_{\tau_z}$ be the representation of the expression e with respect to τ_z. $[e]_{\tau_z}$ is inductively defined as follows:

(1) $[m]_{\tau_z} = S^m 0$;

(2) $[x_i]_{\tau_z} = z_i$, where $i = 1, 2, 3$;

(3) $[e_1 * e_2]_{\tau_z} = [e_1]_{\tau_z} * [e_2]_{\tau_z}$, where $*$ stands for $+, -, \cdot$.

Note that $[e]_{\sigma_z}$ given by Definition 4.13 is different from $[e]_{\tau_z}$ defined here. $[e]_{\sigma_z}$ is similar to the *call by value* mechanism in programming languages, i.e., the substitution is made after evaluating the variables, whereas $[e]_{\tau_z}$ is similar to the *call by name* mechanism in programming languages, i.e., substitution is made first and the variables are evaluated when necessary. These two representations obey the following relations.

Lemma 4.5. *Suppose that the state is* $\sigma_z = (z_1 \mapsto m_1, z_2 \mapsto m_2, z_3 \mapsto m_3)$. *Its corresponding state variable set is* $\tau_z = \{z_1, z_2, z_3\}$. *Then the following sequent is provable:*

$$\Pi \vdash [e]_{\tau_z}[S^{m_1}0/z_1, S^{m_2}0/z_2, S^{m_3}0/z_3] \doteq Tr([e]_{\sigma_z}).$$

After giving the representation of expressions in terms of state variables, we can now give the representation of the assignment statement $x_3 := e$ in a general form, i.e.,

$$T_{x_3:=e}(\tau, \tau') \quad \text{is} \quad ([x_1]_{\tau'} \doteq [x_1]_{\tau}) \wedge ([x_2]_{\tau'} \doteq [x_2]_{\tau}) \wedge ([x_3]_{\tau'} \doteq [e]_{\tau}),$$

or more directly

$$T_{x_3:=e}(\tau, \tau') \quad \text{is} \quad (y_1 \doteq x_1) \wedge (y_2 \doteq x_2) \wedge (y_3 \doteq [e]_{\tau}).$$

(2) Representation of the if statement

Suppose that the **if** statement is **if** $0 < x_1$ **then** α_1 **else** α_2. According to the operational semantics of the **if** statement, it executes the statement α_1 in the state σ if $[x_1]_\sigma > 0$ and it executes α_2 in the state σ if $[x_1]_\sigma = 0$.

If the representation of the statement α_1 is $T_{\alpha_1}(\tau, \tau')$ and the representation of the statement α_2 is $T_{\alpha_2}(\tau, \tau')$, then

$$T_{\textbf{if } 0<x_1 \textbf{ then } \alpha_1 \textbf{ else } \alpha_2}(\tau, \tau') \quad \text{is} \quad \text{cond}(0 < [x_1]_\tau, T_{\alpha_1}(\tau, \tau'), T_{\alpha_2}(\tau, \tau')).$$

(3) Representation of the sequential statement

Suppose that the sequential statement is $\alpha_1; \alpha_2$. According to the operational semantics of the sequential statement, the termination of α_1 started in the state σ will lead to an intermediate state σ_z; then α_2 executes in the state σ_z and terminates with the state σ'. Suppose $\sigma_z = (z_1 \mapsto k_1, z_2 \mapsto k_2, z_3 \mapsto k_3)$. Its corresponding state variable set is $\tau_z = \{z_1, z_2, z_3\}$. Let the representation of the statement α_1 be $T_{\alpha_1}(\tau, \tau_z)$. Further suppose that the representation of the statement α_2 is $T_{\alpha_2}(\tau_z, \tau')$. Then the representation of the statement $\alpha_1; \alpha_2$:

$$T_{\alpha_1;\, \alpha_2}(\tau, \tau') \quad \text{is} \quad \exists z_1 \exists z_2 \exists z_3 (T_{\alpha_1}(\tau, \tau_z) \wedge T_{\alpha_2}(\tau_z, \tau')).$$

We will illustrate the representation in Π of the sequential statement through the following example which will be useful in the discussion of the representation of the **while** statement.

Example 4.2 (Representation of the sequential statement). Consider a sequential statement

$$\alpha \text{ which is } x_1 := x_1 - 1;\ x_3 := x_3 + 1.$$

Let its initial state variable set be $\tau = \{x_1, x_2, x_3\}$.

Suppose that the state after the execution of the statement $x_1 := x_1 - 1$ is σ^1, which is both the terminating state of the statement $x_1 := x_1 - 1$ and the initial state of the statement $x_3 := x_3 + 1$.Let the state variable set of σ^1 be $\tau^1 = \{x_1^1, x_2^1, x_3^1\}$.

Suppose again that the terminating state of the sequential statement α is σ' whose state variable set is $\tau' = \{y_1, y_2, y_3\}$.

We know from the representation of the assignment statement that

$$\begin{aligned} T_{x_1:=x_1-1}(\tau,\ \tau^1) &= (x_1^1 \doteq x_1 - S^1 0) \wedge (x_2^1 \doteq x_2) \wedge (x_3^1 \doteq x_3), \\ T_{x_3:=x_3+1}(\tau^1,\ \tau') &= (y_1 \doteq x_1^1) \wedge (y_2 \doteq x_2^1) \wedge (y_3 \doteq x_3^1 + S^1 0). \end{aligned}$$

Hence the representation of α is

$$\exists x_1^1 \exists x_2^1 \exists x_3^1 ((x_1^1 \doteq x_1 - S^1 0 \wedge x_2^1 \doteq x_2 \wedge x_3^1 \doteq x_3) \wedge (y_1 \doteq x_1^1 \wedge y_2 \doteq x_2^1 \wedge y_3 \doteq x_3^1 + S^1 0)).$$

For this simple example, we can get more direct and simple representation. Firstly, using the $\exists$-*R* rule, we get

$$(x_1^1 \doteq x_1 - S^1 0 \wedge x_2^1 \doteq x_2 \wedge x_3^1 \doteq x_3) \wedge (y_1 \doteq x_1^1 \wedge y_2 \doteq x_2^1 \wedge y_3 \doteq x_3^1 + S^1 0).$$

Then applying the the substitution rule given in Chapter 3, we obtain

$$(y_1 \doteq x_1 - S^1 0 \wedge y_2 \doteq x_2 \wedge y_3 \doteq x_3 + S^1 0),$$

in which x_1^1, x_2^1, x_3^1 are substituted by $x_1 - S^1 0, x_2, x_3$ respectively.

(4) Representation of the while statement

Let us examine an example before we discuss the representation of the **while** statement.

Example 4.3 (Representation the while statement). Let a **while** statement α be

$$\textbf{while } 0 < x_1 \quad \textbf{do} \quad (x_1 := x_1 - 1;\ x_3 := x_3 + 1)$$

whose loop body α_1 is $x_1 := x_1 - 1;\ x_3 := x_3 + 1$. Let the initial state, the state in the beginning of the $(l+1)$th loop, and the terminating state of α be

$$\begin{aligned} \sigma: &\quad (x_1 \mapsto n_3,\ x_3 \mapsto 0), \\ \sigma^l: &\quad (x_1^l \mapsto n_3 - l,\ x_3^l \mapsto l), \\ \sigma': &\quad (y_1 \mapsto 0,\ y_3 \mapsto n_3) \end{aligned}$$

respectively. According to Example 4.2, we can prove by natural induction that the representation of the statement

$$\underbrace{\alpha_1;\ \cdots;\ \alpha_1}_{l\ times}$$

is

$$x_1^l \doteq x_1 - S^l 0 \ \wedge\ x_3^l \doteq S^l 0 + x_3.$$

It is the representation of the statement that is equivalent to executing l loops of α_1. If we substitute l in the above formula by the loop variable w, then

$$B(x_1,\ x_3,\ x_1^w,\ x_3^w,\ w) \quad \text{is} \quad (x_1^w \doteq x_1 - w \ \wedge\ x_3^w \doteq w + x_3)$$

holds. This is a general form to represent the execution of the wth loop. Thus the form

$$\forall\, w\ (w < x_1 \rightarrow\ (x_1^w \doteq x_1 - w \ \wedge\ x_3^w \doteq w + x_3))$$

represents the looping behavior of the **while** statement α.

It should be noted that this form is not a well-defined formula of first-order languages since variable x_1^w itself depends on the bound variable w of the quantifier symbol $\forall$. To make it into a well-defined formula, we replace x_1^w by y_1 and x_3^w by y_3 to obtain the formula

$$B(x_1, x_3, y_1, y_3, w) \quad \text{which is} \quad (y_1 \doteq x_1 - w \ \wedge\ y_3 \doteq w + x_3).$$

Let

$$I_\alpha \quad \text{be} \quad \forall w(w < x_1 \rightarrow B(x_1, x_3, y_1, y_3, w)).$$

It can be verified that the formula I_α is true under $\mathbf{N}$. I_α is called a *loop invariant* of the **while** statement α. The idea of the loop invariant was first introduced also by Hoare [1969].Note that

$$B[x_1/w] \quad \text{is} \quad (y_1 \doteq x_1 - x_1 \ \wedge\ y_3 \doteq x_1 + x_3)$$

which is the representation of the terminating state of the **while** statement. Thus

$$T_\alpha(\tau, \tau') \quad \text{is} \quad \text{cond}((0 < x_1),\ I_\alpha,\ B[x_1/w])$$

which represents the **while** statement α in Π.

Now let us examine the general representation of a **while** statement. Suppose that the **while** statement α is **while** $0 < w$ **do** α'. Let the P-procedure have k variables including input parameters, local variables, and output parameters. Since we only deal with halting P-procedures in computability, the execution of the **while** statement must terminate.

Let the number of loops be l and note that l changes along with the initial state. Also suppose that the initial and terminating states of the $(i+1)$th loop of the loop body α' are σ_i and σ_{i+1} respectively with $0 \leqslant i < l$. We denote these states as σ_0, σ_1, …, σ_l in order of their appearance and call it the *execution sequence* of the loop body with respect to the initial state σ_0, and denote it as $\{\sigma_i\}_0^l$. We also call σ_i the $(i+1)$th *execution state*

of the loop body. Our road map for the representation in Π of the **while** statement is as follows.

(I) The crux of the problem

In Example 4.3, we specified both the representation formula of the loop body and its loop invariant. The reason we could define both is that the structure of the program was simple so that we could guess them before we actually proved them. However, for an arbitrary loop body, it is difficult to guess the general representation form of its loop invariant.

Another possibility is to use structural induction to construct the representation of the **while** statement. But this is also not easy. In fact, if we let the formula $T_{\alpha'}(\tau_i, \tau_{i+1})$ be the representation of the $(i+1)$th loop of the **while** statement, then

$$T_{\alpha'}(\tau_0, \tau_1) \wedge \cdots \wedge T_{\alpha'}(\tau_i, \tau_{i+1}) \wedge \cdots \wedge T_{\alpha'}(\tau_{l-1}, \tau_l) \tag{4.1}$$

should be the "representation" of the loop body. The problem is that the number l of loops changes along with the initial states of the **while** statement and so does the length of the formula (4.1). Thus the above formula actually uses a set of formulas to represent the loop body instead of a single formula in Π as in previous statements.

(II) Gödel's solution

The major difficulty with the above is that we expect the formulas to be constructively defined. Another approach is to give a specification of the **while** statement, i.e., to use the formulas in Π to describe the properties of the execution sequence of loop body. These properties can be summarized as follows.

Lemma 4.6. *A state sequence $\{\sigma_i\}_0^l$ is the execution sequence of loop body of* **while** $0 < x_1$ **do** α' *if and only if it satisfies the following four conditions.*

(1) $\sigma_0 = \sigma$, *i.e.,* σ_0 *is the initial state of the* ***while*** *statement.*

(2) l *is the number of loops of the* ***while*** *statement with the initial state being* σ.

(3) $\sigma_l = \sigma'$ *and* $0 < [x_1]_{\sigma'}$ *does not hold. Here* σ_l *is the terminating state of the* ***while*** *statement.*

(4) *If the initial state is* σ_i *and* $0 < [x_1]_{\sigma_i}$, *then after executing the loop body* α', *the terminating state is* σ_{i+1}, *where* $0 \leqslant i < l$.

If this lemma is proved, then the representation of the **while** statement is reduced to representing the proposition "there exists a state sequence such that all the above four conditions are satisfied." The representation of condition (4) is the only difficulty.

If we assume that the representation of the loop body α' in the initial state σ_i and terminating state σ_{i+1} is $T_{\alpha'}(\tau_i, \tau_{i+1})$, then condition (4) can be represented in the form

$$\exists l \forall i (i < l \rightarrow (0 < [x_1]_{\tau_i} \wedge T_{\alpha'}(\tau_i, \tau_{i+1}))). \tag{4.2}$$

The problem with the above formula is the same as that with (4.1), i.e., the states σ_i and σ_{i+1} both change along with the change of the initial state σ of the **while** statement. Hence

it is not yet a legal formula in $\mathscr{A}$. The solution was provided by Gödel [Shoenfield, 1967]. His basic idea is that if we can use terms in $\mathscr{A}$ to represent each σ_i in the state sequence $\{\sigma_i\}_0^l$ and can define a function symbol in $\mathscr{A}$, then we can obtain terms as representations of every σ_i, after substituting the variables of this function symbol by the subscripts of the state sequence. In this way we can obtain a legal formula in $\mathscr{A}$ which is the representation of condition (4).

Specifically, since a P-procedure always uses a finite number of variables, for a P-procedure with k variables we assume $\sigma_i = (x_1^i \mapsto m_1^i, \ldots, x_k^i \mapsto m_k^i)$. In this way the value of the variable x_j under the state σ_i is $[x_j]_{\sigma_i} = m_j^i$, where $1 \leqslant j \leqslant k$. Thus the values of the variable x_j in the sequence of states $\{\sigma_i\}_0^l$ constitute a sequence $m_j^0, m_j^1, \ldots, m_j^l$ of natural numbers with $1 \leqslant j \leqslant k$. Hence the loop body execution state sequence can be represented by an $(l+1) \times k$ matrix $M[l+1][k]$ of natural numbers. The usual approach in programming languages is as follows:

$$M[l+1][k] := \begin{pmatrix} m_1^0 & m_2^0 & \ldots & m_k^0 \\ m_1^1 & m_2^1 & \ldots & m_k^1 \\ \vdots & \vdots & & \vdots \\ m_1^l & m_2^l & \ldots & m_k^l \end{pmatrix},$$

i.e., $M[i][j] = m_j^i$. It is obvious that, if we can represent the matrix $M[l+1][k]$, then we can also represent the state sequence.

Gödel solved the representation problem for state sequences by proving the following lemma.

Lemma 4.7 (Gödel). *There exists a function $\beta(x,y)$ defined on $\mathbb{N}$, which is representable in Π, such that for an arbitrary sequence a_0, a_1, …, a_{n-1} in $\mathbb{N}$, there exists a natural number a satisfying $\beta(a,i) = a_i$ and $\beta(a,i) \leqslant a-1$, where $i < n$.*

The key of the proof is to define a natural number a and a function β satisfying the lemma. We call a the *generator* of the sequence $a_0, a_1, \ldots, a_{n-1}$ and β its *generating function*. From the perspective of programming, the function β can be regarded as a storage allocation algorithm for the sequence $a_0, a_1, \ldots, a_{n-1}$.

For different subscripts i it allocates different storage addresses. The generator a is a natural number determined by the sequence $a_0, a_1, \ldots, a_{n-1}$ such that it is the starting address of the sequence and a_i is the computation result of β with a and the subscript i as its inputs.

In programming, the storage of the matrix $M[l+1][k]$ is treated as an array $\{a_i\}$ of length $(l+1)\cdot k$ such that $a_{i\cdot k+j-1} = M[i][j]$ holds. Hence, based on the above lemma, we design a ternary function $\gamma(x,y,z)$ and a natural number a to generate the elements of the matrix. Here $\gamma(x,y,z)$ is defined on $\mathbb{N}$ and constructed from the function $\beta(x,y)$ and it is proved to be a representable function. This shows how to represent the matrix in Π.

Suppose that the representation of $\gamma(x,y,z)$ is $C(x,y,z)$ in Π. Let t_1 denote term $C(S^a0, i, S^10)$ with a being the generator, S^a0 being the representation of a in Π, i representing the $(i+1)$th loop and S^10 representing the subscript of the first state variable x_1. Similarly, let t_2 denote $C(S^a0, i, S^20)$, t_3 denote $C(S^a0, i, S^30)$, s_1 denote $C(S^a0, Si, S^10)$,

v_2 denote $C(S^a0, Si, S^20)$, v_3 denote $C(S^a0, Si, S^30)$. The representation of condition (4) in Lemma 4.6:

$$I_\alpha \quad \text{is} \quad \exists l \forall i (i < l \rightarrow (0 < u_1 \wedge T_{\alpha'}(\tau_u, \tau_v)[t_1/u_1, t_2/u_2, t_3/u_3, s_1/v_1, s_2/v_2, s_3/v_3])).$$

Readers may refer to Appendix 3 for more details.

(5) Representation of the call statement

Suppose that the **call** statement is $F(m_1, m_2, x_3)$. Let σ, σ' be the initial and terminating states of the statement with τ, τ' being the corresponding variable sets of the two states respectively. Also let $\tau_u = \{u_1, u_2, u_3\}$ and $\tau_v = \{v_1, v_2, v_3\}$. According to its operational semantics, the **call** statement will execute the procedure body α' in the state $\sigma_u = (u_1 \mapsto m_1, u_2 \mapsto m_2, u_3 \mapsto [x_3]_\sigma)$ and the terminating state σ_v of the execution satisfies $[v_3]_{\sigma_v} = [y_3]_{\sigma'}$. Its corresponding formula $T_\alpha(\tau, \tau')$ is

$$(y_1 \doteq x_1) \wedge (y_2 \doteq x_2) \wedge (\exists v_1 \exists v_2 (T_{\alpha'}(\tau_u, \tau_v)[S^{m_1}0/u_1, S^{m_2}0/u_2, x_3/u_3, y_3/v_3])).$$

After giving the representation of the above five statements, we can prove the following lemma.

Lemma 4.8 (Representability of the procedure body). *Suppose that α is the procedure body of a halting P-procedure with its initial state $\sigma = (x_1 \mapsto m_1, x_2 \mapsto m_2, x_3 \mapsto m_3)$. Also suppose that the terminating state of α is $\sigma' = (y_1 \mapsto n_1, y_2 \mapsto n_2, y_3 \mapsto n_3)$. Let $\sigma_t = (y_1 \mapsto k_1, y_2 \mapsto k_2, y_3 \mapsto k_3)$ be an arbitrary state.*

(1) *If $\sigma_t = \sigma'$, i.e., $k_1 = n_1$, $k_2 = n_2$ and $k_3 = n_3$ hold, then*

$$\Pi \vdash T_\alpha(\tau, \tau')[S^{m_1}0, S^{m_2}0, S^{m_3}0, S^{k_1}0, S^{k_2}0, S^{k_3}0]$$

is provable.

(2) *If $\sigma_t \neq \sigma'$, i.e., $k_1 \neq n_1$ or $k_2 \neq n_2$ or $k_3 \neq n_3$ hold, then*

$$\Pi \vdash \neg T_\alpha(\tau, \tau')[S^{m_1}0, S^{m_2}0, S^{m_3}0, S^{k_1}0, S^{k_2}0, S^{k_3}0]$$

is provable.

Proof. See Appendix 3 for a detailed proof. □

Using Lemma 4.8, we can prove the following theorem.

Theorem 4.1 (Representability of the halting *P*-procedure). *Suppose that the P-procedure $F(x_1, x_2, x_3)$ is a halting procedure with its procedure body being α which defines a computable function $f(x_1, x_2)$. Then there exists a formula $B(x_1, x_2, x_3)$ in $\mathscr{A}$ such that for any natural number n:*

(1) *if $n = f(m_1, m_2)$, then*

$$\Pi \vdash B[S^{m_1}0, S^{m_2}0, S^n0] \text{ is provable;}$$

(2) *if* $n \neq f(m_1, m_2)$, *then*

$$\Pi \vdash \neg B[S^{m_1}0, S^{m_2}0, S^n 0] \text{ is provable.}$$

Proof. Following the operational semantics, the execution of the **call** statement in the configuration $\langle F(m_1, m_2, x_3), \sigma \rangle$ generates the following new configuration:

$$\langle \alpha, \sigma[x_1 \mapsto m_1,\ x_2 \mapsto m_2,\ x_3 \mapsto 0] \rangle.$$

Let

$$T_\alpha(\tau,\ \tau')[S^{m_1}0, S^{m_2}0, 0, 0, S^{n_1}0, S^{n_2}0, S^n 0]$$

be

$$A[S^{m_1}0, S^{m_2}0, 0, 0, S^{n_1}0, S^{n_2}0, S^n 0].$$

Using Lemma 4.8 on the representability of the procedure body, as well as the $\exists$-*R* rule of the **G** system, we can prove that, if $n = f(m_1, m_2)$, then

$$\Pi \vdash \exists x_3 \exists y_1 \exists y_2 A[S^{m_1}0, S^{m_2}0, x_3, y_1, y_2, S^n 0]$$

is provable. And if $n \neq f(m_1, m_2)$, then

$$\Pi \vdash \neg \exists x_3 \exists y_1 \exists y_2 A[S^{m_1}0, S^{m_2}0, x_3, y_1, y_2, S^n 0]$$

is provable. Let $B(x_1, x_2, x_3)$ be

$$\exists z \exists y_1 \exists y_2 A[x_1, x_2, z, y_1, y_2, x_3]$$

and the theorem is proved. □

In fact, using the same arguments, we can further prove that Theorem 4.1 holds for any computable functions with k variables.

4.9 Representability theorem

In this section we formally define the representability of functions and relations defined on $\mathbb{N}$. We also prove the representability of computable functions and decidable relations defined on $\mathbb{N}$.

Definition 4.15 (Representability of functions). Suppose that $f : \mathbb{N}^k \longrightarrow \mathbb{N}$ is a k-ary function defined on $\mathbb{N}$. If there exists a formula $A(x_1, \ldots, x_{k+1})$ in $\mathscr{A}$ such that for any $n_1, \ldots, n_{k+1} \in \mathbb{N}$,

$$\text{if } f(n_1, \ldots, n_k) = n_{k+1}, \text{ then } \Pi \vdash A[S^{n_1}0, \ldots, S^{n_{k+1}}0] \text{ is provable, and}$$
$$\text{if } f(n_1, \ldots, n_k) \neq n_{k+1}, \text{ then } \Pi \vdash \neg A[S^{n_1}0, \ldots, S^{n_{k+1}}0] \text{ is provable,}$$

we say that the function f is *representable* in Π and call the formula $A(x_1, \ldots, x_k, x_{k+1})$ the *representation* of the function f in Π.

The following theorem for computable functions is a direct consequence of Theorem 4.1.

Theorem 4.2. *If $f : \mathbb{N}^k \to \mathbb{N}$ is a k-ary computable function defined on $\mathbb{N}$, then the function f is representable in Π.*

Proof. By Definition 4.10, there exists a halting P-procedure $F(x_1,\ldots,x_k,x_{k+1})$ that computes the function f, since $f(x_1,\ldots,x_k)$ is a computable function defined on $\mathbb{N}$. Then according to Theorem 4.1, there exists a formula $A(x_1,\ldots,x_k,x_{k+1})$ in $\mathscr{A}$ that represents the procedure F in Π. By Definition 4.15, the formula $A(x_1,\ldots,x_k,x_{k+1})$ represents the function f in Π. □

Definition 4.16 (Representability of relations). Suppose that r is a k-ary relation defined on $\mathbb{N}$. If there exists a formula $A(x_1,\ldots,x_k)$ in $\mathscr{A}$ such that for any $n_1,\ldots,n_k \in \mathbb{N}$,

$$\text{if } r(n_1,\ldots,n_k) \text{ is true, then } \Pi \vdash A[S^{n_1}0,\ldots,S^{n_k}0] \text{ is provable, and}$$
$$\text{if } r(n_1,\ldots,n_k) \text{ is false, then } \Pi \vdash \neg A[S^{n_1}0,\ldots,S^{n_k}0] \text{ is provable,}$$

then we say that the relation r is *representable* in Π and the formula $A(x_1,\ldots,x_k)$ *represents* the relation r in Π.

Theorem 4.3. *If $r : \mathbb{N}^k \longrightarrow \mathbb{N}$ is a k-ary decidable relation on $\mathbb{N}$, then r is representable in Π.*

Proof. The conclusion follows immediately from Theorem 4.2. □

We have now proved the representability of halting P-procedures, computable functions and decidable relations in the theory of elementary arithmetic Π, which will be used to prove Gödel's theorems in the next chapter.

In addition, we would like to point out the following two issues:

Firstly, not every function on $\mathbb{N}$ is computable nor is every relation on $\mathbb{N}$ decidable. In the next chapter we will provide some instances of uncomputable functions and undecidable relations defined on $\mathbb{N}$.

Secondly, in software engineering, before writing the programs, the designers usually determine the requirement and specifications of a system. To the extent that specifications are written in first-order languages, they are formal theories. For a given formal theory, does there exist a halting P-procedure that implements it? Generally speaking, there is not a theorem, like the representability theorem, to guarantee that specifications can be implemented.

Even though there is no general procedure for implementing specifications in programs or hardware, due to the urgent demand of software development, enormous efforts have been made to develop systematic solutions to this problem. These efforts use formal calculus or computer-aided implementation systems.

Chapter 5

Gödel Theorems

From the viewpoint of logic there are two types of knowledge about a specific domain. One is given knowledge, the axiom system, and the other consists of the logical conclusions from the axioms. The logical conclusions are propositions deduced from the axioms by using inference rules, which are independent of the domain. Therefore, the question whether a given proposition is a logical conclusion only depends on the axioms.

This methodology for generating structured knowledge about a domain is called the axiomatic approach, whereby the axioms are defined first and then we hope to deduce all the other valid knowledge about a domain by logical deduction. It is natural to ask whether, for a specific domain, there is an axiom system that completely grasps all the essential characteristics of the domain. In other words, *whether every proposition or its negation is a logical conclusion.* This question is called the completeness problem of axiom systems.

In the early 1930s, Gödel proved, within the framework of first-order languages, that *every finite formal theory*[1]*, that contains the theory of elementary arithmetic* Π*, is incomplete*. This is the well-known incompleteness theorem of Gödel [Shoenfield, 1967]. Gödel's theorem is noted for its profundity. Our intuition tells us that useful theories of mathematics and natural science can be described by finite formal theories of first-order languages. These theories must also be consistent and most should contain the theory of arithmetic in order to describe numerical calculations. According to Gödel's incompleteness theorem, the formal theories that satisfy the above three conditions cannot be complete. Gödel further proved that *for any formal theory* Γ *satisfying the above three conditions, it is impossible to prove the consistency of* Γ *itself by using a formal inference system with* Γ *as premise*. This is Gödel's consistency theorem, or, as it is sometimes called, Gödel's second incompleteness theorem. These two theorems radically reveal the limitations of the axiomatic approach to structure the knowledge of mathematics and science.

The main task of this chapter is to prove Gödel's incompleteness theorem and consistency theorem. We shall focus on the proof of Gödel's incompleteness theorem and the outline of proof is as follows.

1. So long as it is proved that the theory of elementary arithmetic Π is incomplete, a similar method can be used to prove that any formal theory containing Π is incomplete.

2. The key to proving the incompleteness of Π consists in finding a formula A of $\mathscr{A}$

[1]The precise description should say that the axiom system is an axiomatizable set, which means it is an enumerable set.

such that both $\Pi \vdash A$ and $\Pi \vdash \neg A$ are unprovable.

3. Such a formula A is found by embodying, in elementary arithmetic, the well-known "liar's paradox", which is typically phrased as "*I am not telling the truth*" or "*this proposition* is unprovable." It can be shown that this type of proposition can be neither proved nor disproved.

4. The liar's paradox is a kind of self-referential proposition. If we can find a formula A of $\mathscr{A}$, which can be interpreted as "*this proposition is unprovable*," then using the method of proof by contradiction we can show that both A and $\neg A$ are unprovable in Π.

Following the above road map, we shall complete the proofs of Gödel's two theorems in five sections. Section 5.1 resolves the problem of describing self-referential sentences in $\mathscr{A}$. We use the method of Gödel coding to represent formulas by means of terms in $\mathscr{A}$ and then show that the formal description of every self-referential proposition is a solution of a fixed point equation of formulas. The decidable and enumerable sets of symbols are introduced in Section 5.2. It is proved there that any finite and complete formal theory is decidable. In Section 5.3 it is proved that the fixed point equation of formulas in the theory of elementary arithmetic Π is provable. Gödel's incompleteness theorem is proved in Section 5.4. The key to the proof is to construct the fixed point equation in $\mathscr{A}$, whose solution is a formula that expresses "this proposition is unprovable," and then to prove the undecidability of $Th(\Pi)$ by contradiction. Finally in Section 5.5 we prove that it is impossible to use the **G** system to deduce the consistency of Π. The key to the proof is again to use the method of Gödel coding to describe consistency.

5.1 Self-referential proposition

In this section we employ a more theoretical form of the liar's paradox which can be stated as follows:

"This proposition is unprovable." (5.1)

This sentence is *self-referential* because the words "this proposition" refer to both the whole proposition and the subject of the proposition (5.1).

Let us prove that (5.1) is a proposition that can be neither proved nor disproved. Let X denote "this proposition." Since "this proposition" refers to "this proposition is unprovable," X also denotes "this proposition is unprovable." Let us prove by contradiction that X is unprovable.

First, suppose "X is provable," i.e., "this proposition is unprovable" is proved. If we replace "this proposition" in quotes by X yields that "X is unprovable" is proved, which is a contradiction.

On the other hand, suppose "$\neg X$ is provable." Since $\neg X$ denotes the negation of "X is unprovable", that is "X is provable" is provable. This leads to a contradiction.

Thus both "this proposition is unprovable" and its negation are unprovable.

In what follows we will discuss how to express "this proposition is unprovable" in $\mathscr{A}$. In the above proof, X represents a proposition and is a "proposition variable" or "relation variable." Assume that Y is also a proposition variable and F is a "predicate" with Y being its free *proposition variable*, which is a unary predicate. Then $F(Y)$ can be interpreted as

$$\textit{"the proposition } Y \textit{ has the property } F\textit{."} \tag{5.2}$$

We further assume that the proposition variable X is equivalent to $F(Y)$, which can be represented as

$$X \leftrightarrow F(Y). \tag{5.3}$$

It can be interpreted as: X and $F(Y)$ have the same truth value, or the proposition X refers to "the proposition Y has the property F." If we replace the Y of $F(Y)$ in (5.3) by the proposition variable X, we shall obtain

$$X \leftrightarrow F(X). \tag{5.4}$$

(5.4) can be interpreted as the proposition X is that "the proposition X has the property F." (5.4) is the general mathematical form of self-referential propositions. It has the same form as the fixed point equation of a function f in mathematics:

$$x = f(x). \tag{5.5}$$

The difference is the fact that (5.4) is not a formula of first-order language because in the fixed point equation the variable is a "proposition variable" and the symbol $\leftrightarrow$ represents an equivalence relation between sentences. In contrast, the variable in (5.5) is a variable defined in first-order language.

Since the solutions of (5.5) are called *fixed points* of f, the self-referential propositions can be treated as the solutions of the fixed point equation (5.4) on propositions, i.e., they are fixed points of F.

We should emphasize again that the fixed point equation (5.4) is not a formula of first-order language because the variable X in $F(X)$ is not a variable of first-order language. By Definition 1.2, in first-order languages a free variable is a term that can be substituted only by terms such as a constant symbol, variable or function symbol. However, X in the fixed point equation (5.4) denotes a formula and is not a term. $F(A)$, obtained by replacing X by the formula A, is not a formula of first-order languages either.

To solve the above problem, Gödel invented a method of describing $F(A)$ as a first-order formula. This is the Gödel coding introduced in Section 1.5. His basic idea is to map each formula A bijectively to a natural number $\&A$ that is called the *Gödel number* of the formula A. Since each natural number n can be represented by the term S^n0 in $\mathscr{A}$, each formula A corresponds one-to-one to the term $S^{\&A}0$ that is called the *Gödel term* of the formula A. Because $S^{\&A}0$ is a term of $\mathscr{A}$, F can then be defined as an unary predicate symbol of $\mathscr{A}$ and $F[S^{\&A}0]$ is a legitimate formula of $\mathscr{A}$. Thus

$$A \leftrightarrow F[S^{\&A}0] \tag{5.6}$$

is also a legitimate formula of $\mathscr{A}$ describing the self-referential formula in $\mathscr{A}$ which is interpreted as "this proposition A is that this proposition A has property F."

Finally we discuss the description in $\mathscr{A}$ of "this proposition is unprovable." Since the "provability" is a property of the propositions, $\neg G(S^{\&A}0)$ describes the unprovability of the formula A if we can find an appropriate predicate $G(x)$ such that $G(S^{\&A}0)$ describes the provability of the formula A. Thus

$$A \leftrightarrow \neg G(S^{\&A}0) \tag{5.7}$$

is a description in $\mathscr{A}$ of "this proposition is unprovable." The above discussions tell us that neither (5.1) nor its negation is provable. Hence, if we prove that the fixed point of the equation (5.7) can be neither proved nor disproved in Π, then we prove the incompleteness of the theory of elementary arithmetic Π.

5.2 Decidable sets

We introduce the concept of decidability for sets of symbol strings, in order to prepare for the proof of Gödel's incompleteness theorem and consistency theorem in the following sections.

Definition 5.1 (Symbol sets, symbol strings and orders). Let $\mathbf{A}$ be a countable set of symbols:

$$\mathbf{A} = \{a_0, a_1, \ldots, a_k, \ldots,\},$$

where each a_i denotes a symbol.

An array composed of finitely many symbols in $\mathbf{A}$ is called a *symbol string* of $\mathbf{A}$. A symbol is allowed to appear in a symbol string repetitively. We usually use w, u and v to denote symbol strings and write them in the form

$$w = a_1 \cdots a_k, \qquad \text{where } a_i \in \mathbf{A},\ i = 1, \ldots, k.$$

Here k is called the *length of the symbol string* w denoted as $\text{length}(w)$. The empty symbol string is allowed and is denoted as $\square$.

The set $\mathbf{A}^*$ is composed of all the symbol strings of $\mathbf{A}$. We can define an ordering relation, or an *order* for short, among the symbol strings in $\mathbf{A}^*$. We prescribe that the order is reflexive and transitive and will be denoted as $\prec$.

Example 5.1 (Quasi-lexicographic order).

(1) For any two distinct symbols a_i and a_j in $\mathbf{A}$, we read $a_i \prec a_j$ as a_i precedes a_j.

(2) For any symbol strings w and w', we say that w precedes w' and denote it as $w \prec w'$ if

$$\text{length}(w) < \text{length}(w')$$

or

$$\text{length}(w) = \text{length}(w') \quad \text{and} \quad w = ua_iv,\ w' = ua_jv', \quad \text{and} \quad a_i \prec a_j.$$

Example 5.2 (A first-order language $\mathscr{L}$). Suppose that the symbol set **A** contains all the symbols used in the set V of variable symbols, the set C of logical connective symbols, the set Q of quantifier symbols, the set E of the equality symbol, the set P of parentheses, the set $\mathscr{L}_c$ of constant symbols, the set $\mathscr{L}_f$ of function symbols, the set $\mathscr{L}_P$ of predicate symbols, and all kinds of symbols of first-order languages employed in this book.

Each term of $\mathscr{L}$ is a symbol string and $\mathscr{L}_T$ is the set of symbol strings comprising all the terms. Each formula of $\mathscr{L}$ is a symbol string as well and $\mathscr{L}_F$ is the set of symbol strings comprising all the logical formulas. All these sets are subsets of $\mathbf{A}^*$.

In order to manipulate symbol strings, we add two more atomic statements to the P-kernel. They are the symbol addition statement and symbol subtraction statement, whose definitions are as follows.

Symbol addition statement

$$x_i := x_i + a_j$$

where i and j are natural numbers and a_j is a symbol. The symbol addition statement executes by appending the symbol a_j to the end of the symbol string stored in x_i.

Symbol subtraction statement

$$x_i := x_i - a_j$$

where i and j are natural numbers and a_j is a symbol. The symbol substraction statement executes as follows: if the last symbol of the symbol string stored in x_i is a_j, then a_j is deleted from x_i; otherwise, x_i is unchanged.

Here it should be pointed out that if we also treat the natural numbers $1, 2, \ldots, n, \ldots$ as symbols, then this extended P-kernel is a formal object language. The language, referred to as the *P-kernel language*, is defined on the symbol set **A** and the symbol set of natural numbers. It amounts to a subset of the language C except that the P-kernel contains countably infinite symbols.

Definition 5.2 (Decidable set). Let $W \subset \mathbf{A}^*$ be a set of symbol strings. F is a halting P-procedure whose input and output are both symbol strings. We say that the procedure F *decides* W if for any $w \in \mathbf{A}^*$, we have:

$$\text{if } w \in W \text{ then } F: w \to \text{Yes},$$
$$\text{if } w \notin W \text{ then } F: w \to \text{No}.$$

We say that the set W of symbol strings is *decidable* if there exists a P-procedure, F, that decides W. F is called a decision procedure of W.

In practical applications, it is sufficient that the procedure halts for any input and the outputs can be distinguished by whether its input is in W or not.

Example 5.3 (A first-order language $\mathscr{L}$ [continued]). Both the set of terms and the set of formulas of a first-order language $\mathscr{L}$ are decidable sets. We assume that both sets

are listed in lexicographic order. Take the set of terms as an example. If we introduce the finite alphabet

$$\mathbf{A}_0 = \{a,b,\ldots,z,A,B,\ldots,Z\} \cup \{0,1,\ldots,9\} \cup C \cup Q \cup E \cup P,$$

then every symbol in $\mathbf{A}$ can be represented by a string of $\mathbf{A}_0$.

For example, the constant symbol c_6 is viewed as $c6$, the variable symbol x_{12} is viewed as $x12$, and the formula $\forall x_{12}(y_1 \doteq f_1 x_{12})$ is viewed as $\forall x12(y1 \doteq f1x12)$. According to the definition of the terms of $\mathscr{L}$, we design a halting procedure as follows: For each input, a symbol string in $\mathbf{A}_0^*$,

(1) the procedure checks whether it is a constant symbol. If it is, then the procedure halts and outputs Yes;

(2) the procedure checks whether it is a variable. If it is, then the procedure halts and outputs Yes;

(3) the procedure checks whether it is a function term according to the definition. The procedure first checks whether the prefix symbol substring is a function symbol. If it is not, then the procedure outputs No;

(4) the procedure goes back to (1) to determine whether the following symbol strings are terms recursively.

This procedure halts because a symbol string contains only finite symbols of the alphabet $\mathbf{A}_0$. Similarly, we can design a halting procedure to distinguish the formulas of $\mathscr{L}$. The above procedure illustrates the idea of a parser (syntactic analysis program) in compiler technology.

Definition 5.3 (Recursively enumerable set). Let $W \subset \mathbf{A}^*$ be a set of symbol strings and F be a P-procedure. We say that F *enumerates* W if F has no input but it outputs every symbol string in W one by one. We say that W is a *recursively enumerable set* if there exists a P-procedure that enumerates W.

Example 5.4 (The set of natural numbers). The set of symbol strings $W = \{\mathrm{I}, \mathrm{II}, \mathrm{III}, \ldots\}$ is a recursively enumerable set. In fact, we can design a P-procedure F whose procedure body is composed of one statement:

$$\textbf{while } 0 < 1 \textbf{ do begin } x := x + \mathrm{I}; \textbf{ print } x \textbf{ end}.$$

The procedure F is not a halting procedure and it has no input. But it outputs all the symbol strings of W one by one. If the natural numbers are coded in the following way,

$$1 \text{ is coded as } \mathrm{I} \text{ and } n \text{ is coded as } \underbrace{\mathrm{II}\cdots\mathrm{I}}_{n \text{ times}},$$

then we have already proved that the set of natural numbers is a recursively enumerable set.

Lemma 5.1. *If the symbol set* $\mathbf{A}$ *is finite, then* $\mathbf{A}^*$ *is recursively enumerable.*

Proof. Let $\mathbf{A} = \{a_0, \ldots, a_n\}$. We design a P-procedure F such that it outputs all the symbol strings in $\mathbf{A}^*$ according to the quasi-lexicographic order as in Example 5.1. F is composed of two nested loops. In the inner loop, the symbol strings of the length m are generated and output according to the quasi-lexicographic order. In the outer loop, m increases by 1.

In the same way, we can design a P-procedure to prove that if the symbol set $\mathbf{A}$ is countable, then $\mathbf{A}^*$ is recursively enumerable. □

In order to prove that the fixed point equation (5.6) is provable in Π, we need the following two halt P-procedures.

Example 5.5 ($GN(X)$: Computing the Gödel number of the formula X). According to the definition of Gödel numbers of logical formulas in Section 1.5, we can design a halting P-procedure $GN(X)$ whose input is a formula A of $\mathscr{A}$ and whose output is the Gödel number $\&A$ of the formula A.

Example 5.6 ($GF(x)$: Decidability of the set of Gödel numbers of formulas with a single free variable). For a first-order language $\mathscr{L}$, let G_1 be the set consisting of Gödel numbers of the formulas whose free variable is x_1. $GF(x)$ takes a natural number n as input and then performs a prime decomposition on n according to the definition of Gödel numbers of formulas. If n is the Gödel number of some formula $R(x_1)$ with x_1 being its free variable, then it outputs the formula $R(x_1)$; otherwise, it outputs 0. $GF(x)$ is a halting P-procedure, and therefore G_1 is a decidable set.

The proof of Gödel's incompleteness theorem also involves the following lemma.

Lemma 5.2. *Let $\mathscr{L}$ be a first-order language and Γ be a formal theory of $\mathscr{L}$. If Γ is both recursively enumerable and complete, then the set $Th(\Gamma)$ is decidable.*

Proof. We design P-procedure Q to decide $Th(\Gamma)$ as follows. Since the formulas of $\mathscr{L}$ are decidable, Q calls the procedure that outputs the formulas one by one in the quasi-lexicographic order. For each output formula A, since Γ is recursively enumerable and complete, the procedure Q executes as follows: if $\Gamma \vdash A$ is provable, then it outputs Yes; if $\Gamma \vdash \neg A$ is provable, then it outputs No. The procedure Q halts and therefore $Th(\Gamma)$ is decidable. □

To prove Gödel's consistency theorem, we need to define the Gödel terms of sequents and inference trees. To do so, we need to define the Gödel numbers of the symbol $\vdash$ and the symbol tr denoting the tree structure. For this we need to make some minor revision of Definition 1.9 about Gödel numbers in Section 1.5. In that definition, the Gödel number of the variable x_1 follows right after that of the symbol $\exists$ and is defined as 27. In fact, as we pointed out at the time, every odd number after 23 can serve as the Gödel number of x_1. For instance, we can start from 101 and define $\&(x_n) = 101 + 2 \cdot n$.

In this way we can leave enough space for the Gödel coding of other symbols (such as $\vdash$ and other objects such as substitution operations, the **G** system and inference trees). In what follows we define the Gödel numbers of sequents and inference trees. Let

$$\Gamma = \{A_1, \ldots, A_m\} \quad \text{and} \quad \Delta = \{B_1, \ldots, B_n\}.$$

The Gödel numbers of sequents are:

$$\begin{array}{lll} \text{the symbol} & \&(\vdash) & = 25, \\ \text{the antecedent} & \&(\Gamma) & = \&(A_1 \wedge \cdots \wedge A_m), \\ \text{the succedent} & \&(\Delta) & = \&(B_1 \vee \cdots \vee B_n), \\ \text{the sequent} & \&(\Gamma \vdash \Delta) & = \langle \&(\vdash), \&(\Gamma), \&(\Delta) \rangle. \end{array}$$

Let $tr(\Gamma \vdash \Delta)$ be an inference tree whose root is the sequent $\Gamma \vdash \Delta$. Its Gödel number is defined inductively as follows:

$$\begin{array}{ll} \text{the symbol} & \&(tr) = 27, \\ \text{single node tree} & \&(tr(\Gamma \vdash \Delta)) = \langle \&(tr), \&(\Gamma \vdash \Delta) \rangle, \\ \text{a tree with one branch} & \&(\dfrac{tr(\Gamma' \vdash \Delta')}{\Gamma \vdash \Delta}) = \langle \&(tr), \&(\Gamma \vdash \Delta), \&(tr(\Gamma' \vdash \Delta')) \rangle, \\ \text{a tree with two branches} & \&(\dfrac{tr(\Gamma_1 \vdash \Delta_1) \quad tr(\Gamma_2 \vdash \Delta_2)}{\Gamma \vdash \Delta}) \\ & = \langle \&(tr), \&(\Gamma \vdash \Delta), \&(tr(\Gamma_1 \vdash \Delta_1)), \&(tr(\Gamma_2 \vdash \Delta_2)) \rangle. \end{array}$$

5.3 Fixed point equation in Π

In this section we answer the question posed at the end of Section 5.1 and prove the following fixed point theorem.

Theorem 5.1 (Fixed point theorem). *If $B(x)$ is a given formula of $\mathscr{A}$ with only one free variable, then there exists a sentence $\mathring{A}$ of $\mathscr{A}$ such that*

$$\Pi \vdash (\mathring{A} \leftrightarrow B[S^{\&\mathring{A}}0]) \tag{5.8}$$

is provable. The sentence $\mathring{A}$ is called the fixed point *of the equation $A \leftrightarrow B[S^{\&A}0]$.*

Proof. The proof is done in two steps: we first construct a sentence $\mathring{A}$ and then prove that $\mathring{A}$ is a solution of the equation (5.8).

1. Construction of the sentence $\mathring{A}$. Consider the following function:

$$f(n,m) = \begin{cases} \&R[S^m 0], & \text{if } n = \&R(x_1), \\ 0, & \text{otherwise}, \end{cases} \tag{5.9}$$

where $R(x_1)$ is a formula containing a single free variable. As in Section 5.2, we design the halting P-procedure F computing $f(n,m)$ with $GN(X)$ and $GF(x)$ as its subroutines. For any given input (n,m), F calls the procedure $GF(n)$ first.

If n is the Gödel number of a formula $R(x_1)$ with x_1 being its free variable, then $GF(n)$ outputs the formula $R(x_1)$ before it calls the procedure $GN(R[S^m 0])$ to output the Gödel number $\&R[S^m 0]$ of $R[S^m 0]$.

If n is not a Gödel number of any formula whose free variable is x_1, then it outputs 0. According to Definition 4.10, $f(n,m)$ is a computable function.

According to the representability theorem, i.e., Theorem 4.2, $f(n,m)$ is representable in Π. Let the formula $P(x_1,x_2,x_3)$ represent $f(x_1,x_2)$ in Π. If $f(\&R(x_1),m) = \&R[S^m 0]$, then

$$\Pi \vdash P[S^{\&R(x_1)}0, S^m 0, S^{\&R[S^m 0]}0] \tag{5.10}$$

is provable. If $f(\&R(x_1),m) \neq \&R[S^m 0]$, then

$$\Pi \vdash \neg P[S^{\&R(x_1)}0, S^m 0, S^{\&R[S^m 0]}0] \tag{5.11}$$

is provable. In particular, let $D(x_1)$ be

$$\forall x_2(P(x_1,x_1,x_2) \to B(x_2)). \tag{5.12}$$

The formula P in $D(x_1)$ represents $f(n,m)$ in Π and it is also the P occurring in (5.10) and (5.11). The formula B in (5.12) is the same B in (5.8). Next, let the formula $\mathring{A}$ be $D[S^{\&D(x_1)}0]$, i.e.,

$$\forall x_2(P(S^{\&D(x_1)}0, S^{\&D(x_1)}0, x_2) \to B(x_2)) \tag{5.13}$$

which is the sentence obtained by substituting x_1 in $D(x_1)$ with the term $S^{\&D(x_1)}0$.

2. Proving that $\mathring{A}$ is the solution of the fixed point equation (5.8)**.** It suffices to prove that

$$\text{both} \quad \Pi \vdash \mathring{A} \to B[S^{\&\mathring{A}}0] \quad \text{and} \quad \Pi \vdash B[S^{\&\mathring{A}}0] \to \mathring{A} \tag{5.14}$$

are provable.

Since $\mathring{A}$ is $D[S^{\&D(x_1)}0]$, we have

$$f(\&D(x_1),\&D(x_1)) = \&D[S^{\&D(x_1)}0] = \&\mathring{A}. \tag{5.15}$$

In addition, according to the representability theorem, f being a computable function indicates that

$$\Pi \vdash P[S^{\&D(x_1)}0, S^{\&D(x_1)}0, S^{\&\mathring{A}}0] \tag{5.16}$$

is provable. By definition, $\mathring{A}$ is the formula given in (5.13). Notice that the formula on the right-hand side of $\vdash$ in (5.17) is obtained by substituting the bound variable x_2 of the universal quantifier of $\mathring{A}$ by the term $S^{\&\mathring{A}}0$. Thus we can apply the $\forall$-*L* rule and the axiom of the **G** system and obtain that

$$\Pi, \mathring{A} \vdash P[S^{\&D(x_1)}0, S^{\&D(x_1)}0, S^{\&\mathring{A}}0] \to B[S^{\&\mathring{A}}0] \tag{5.17}$$

is provable. Then we apply the modus ponens rule to (5.16) and (5.17) and obtain that

$$\Pi, \mathring{A} \vdash B[S^{\&\mathring{A}}0] \tag{5.18}$$

is provable. Next, an application of the $\rightarrow$-R rule to (5.18) shows that

$$\Pi \vdash \mathring{A} \rightarrow B[S^{\&\mathring{A}}0] \tag{5.19}$$

is provable. Thus the first sequent in (5.14) is provable.

In what follows we prove that the second sequent in (5.14) is provable. Taking $n \neq \&\mathring{A}$, according to the representability theorem, i.e., Theorem 4.2, we know that

$$\Pi \vdash \neg P[S^{\&D(x_1)}0, S^{\&D(x_1)}0, S^n 0] \tag{5.20}$$

is provable. That is,

$$\Pi, \neg(x_2 \doteq S^{\&\mathring{A}}0) \vdash \neg P[S^{\&D(x_1)}0, S^{\&D(x_1)}0, x_2] \tag{5.21}$$

is provable. By the $\neg$-R rule,

$$\Pi, \neg(x_2 \doteq S^{\&\mathring{A}}0), P[S^{\&D(x_1)}0, S^{\&D(x_1)}0, x_2] \vdash \tag{5.22}$$

is provable. Then according to Lemma 3.6, we know that

$$\Pi, \neg(x_2 \doteq S^{\&\mathring{A}}0), B[S^{\&\mathring{A}}0], P[S^{\&D(x_1)}0, S^{\&D(x_1)}0, x_2] \vdash B(x_2) \tag{5.23}$$

is provable. The **G** axiom and the substitution rules indicate that

$$\Pi, x_2 \doteq S^{\&\mathring{A}}0, B[S^{\&\mathring{A}}0], P[S^{\&D(x_1)}0, S^{\&D(x_1)}0, x_2] \vdash B(x_2) \tag{5.24}$$

is provable. An application of the rule of proof by cases given in Section 3.6 to (5.23) and (5.24) proves that

$$\Pi, B[S^{\&\mathring{A}}0], P[S^{\&D(x_1)}0, S^{\&D(x_1)}0, x_2] \vdash B(x_2) \tag{5.25}$$

holds. As per the $\rightarrow$-R rule, this amounts to

$$\Pi, B[S^{\&\mathring{A}}0] \vdash P[S^{\&D(x_1)}0, S^{\&D(x_1)}0, x_2] \rightarrow B(x_2) \tag{5.26}$$

being provable. Using the $\forall$-R rule, this is actually

$$\Pi, B[S^{\&\mathring{A}}0] \vdash \forall x_2(P[S^{\&D(x_1)}0, S^{\&D(x_1)}0, x_2] \rightarrow B(x_2)) \tag{5.27}$$

being provable. And an application of the $\rightarrow$-R rule shows that

$$\Pi \vdash B[S^{\&\mathring{A}}0] \rightarrow \forall x_2(P[S^{\&D(x_1)}0, S^{\&D(x_1)}0, x_2] \rightarrow B(x_2)) \tag{5.28}$$

is provable. This amounts to

$$\Pi \vdash B[S^{\&\mathring{A}}0] \rightarrow \mathring{A} \tag{5.29}$$

being provable. By now we have proved that $\mathring{A}$ is a fixed point of the equation (5.8). □

The proof of Theorem 5.1 shows that, for the theory of elementary arithmetic, the solution of the fixed point equation (5.8) does exist and it is the formula $D[S^{\&D(x_1)}0]$. We need to make two comments about the proof of the fixed point theorem.

(1) The definition of the formula $\mathring{A}$ is not a vicious circle in the proof. In fact, we first defined the function $f(n,m)$ and proved that it is a computable function on $\mathbb{N}$. Since $f(n,m)$ is computable, then, according to Theorem 4.2, there is a formula $P(x_1,x_2,x_3)$ which represents the function $f(n,m)$ in Π. Because $P(x_1,x_2,x_3)$ is a formula of $\mathscr{A}$, $\forall x_2(P(x_1,x_1,x_2) \to B(x_2))$ is also a formula $D(x_1)$ whose corresponding Gödel number is $\&D(x_1)$. This number can be substituted for either the first variable n or the second variable m of the function $f(n,m)$. $\&\mathring{A}$ is the value $f(\&D(x_1),\&D(x_1))$ obtained by these substitutions whose corresponding sentence is $\mathring{A}$, i.e., $D[S^{\&D(x_1)}0]$. Thus none of the definitions of $D(x_1)$, $D[S^{\&D(x_1)}0]$ and $f(\&D(x_1),\&D(x_1))$ is a vicious circle.

(2) Suppose that the P-procedure computing $f(n,m)$ is $F(x_1,x_2,x_3)$ and the result is stored in x_3. In practical programming, before the procedure is executed it must be compiled into a segment of executable binary code, called the *code of the procedure*.

From the perspective of mathematics, this segment of binary code is a natural number as well. Since the formula $P(x_1,x_2,x_3)$ represents the function $f(n,m)$ in Π, the Gödel number of $P(x_1,x_2,x_3)$ can be regarded as the code of the procedure $F(x_1,x_2,x_3)$.

Because $D(x_1)$ is composed of $P(x_1,x_2,x_3)$, its Gödel number $\&D(x_1)$ can also be regarded as the code of the procedure F. In this way the function value $f(\&D(x_1),\&D(x_1))$ amounts to the procedure $F(x_1,x_2,x_3)$ executing $F(\&D(x_1),\&D(x_1),x_3)$ with *the code of the procedure itself* as its inputs of actual parameters and $f(\&D(x_1),\&D(x_1))$ as its output after it halts.

This is a kind of procedure with *the code of the procedure itself* being its actual parameters. This is different from structural induction which always starts from the atomic structures, which are already known and develops composite structures step by step without referring to itself.

5.4 Gödel's incompleteness theorem

In this section we prove Gödel's incompleteness theorem. We need to consider the following relation on $\mathbb{N}$.

Definition 5.4 (Relation $g(n)$). The subset of $\mathbb{N}$:

$$\mathbb{G} = \{\,\&A \mid A \in \mathscr{A} \text{ and } \Pi \vdash A \text{ is provable}\} \tag{5.30}$$

is called the *Gödel set* of $Th(\Pi)$. Suppose that $g(n)$ is a unary relation on $\mathbb{N}$ such that $g(n)$ holds if and only if $n \in \mathbb{G}$.

We can interpret Definition 5.4 in the following way: since each formal consequence A of Π is a formula of $\mathscr{A}$, there exists a Gödel number $\&A$ corresponding to A according to the Gödel coding introduced in the first chapter. These Gödel numbers form the set $\mathbb{G}$ which is defined by the unary relation $g(n)$. In what follows we shall prove that $g(n)$ is undecidable.

Lemma 5.3. *$g(n)$ is not representable in Π.*

Proof. Suppose that $g(n)$ is representable in Π. As in Definition 4.16, there exists a formula $G(x)$ of $\mathscr{A}$ with a single variable such that for any formula A, we have:

$$\text{if } g(\&A) \text{ holds, then } \Pi \vdash G[S^{\&A}0] \text{ is provable;}$$
$$\text{if } g(\&A) \text{ does not hold, then } \Pi \vdash \neg G[S^{\&A}0] \text{ is provable.}$$

As in the definition of $g(n)$, we also have:

$$g(\&A) \text{ does not hold, if and only if } \Pi \vdash A \text{ is unprovable;}$$
$$g(\&A) \text{ holds, if and only if } \Pi \vdash A \text{ is provable.}$$

Thus

$$\Pi \vdash \neg G[S^{\&A}0] \text{ is provable if and only if } \Pi \vdash A \text{ is unprovable.} \tag{5.31}$$

Since $G(x)$ is a formula of $\mathscr{A}$, $\neg G(x)$ is also a formula of $\mathscr{A}$. Consider the fixed point equation composed of the formula $\neg G(x)$. According to Theorem 5.1, there exists a sentence $\mathring{A}$ of $\mathscr{A}$ such that

$$\Pi \vdash (\mathring{A} \leftrightarrow \neg G[S^{\&\mathring{A}}0]) \tag{5.32}$$

is provable. Thus according to (5.32) and the modus ponens rule, $\Pi \vdash \mathring{A}$ is provable if and only if $\Pi \vdash \neg G[S^{\&\mathring{A}}0]$ is provable. Then according to (5.31), this is if and only if $\Pi \vdash \mathring{A}$ is unprovable, which is a contradiction. This shows that our assumption does not hold and thus $g(n)$ is not representable in Π. The lemma is proved. □

Lemma 5.4. *$g(n)$ is an undecidable relation on $\mathbb{N}$.*

Proof. If this lemma does not hold, then $g(n)$ will be a decidable relation on $\mathbb{N}$. According to Theorem 4.3, $g(n)$ is representable in Π, which contradicts Lemma 5.3. □

We should point out that $g(n)$ is the first undecidable relation or the first uncomputable function to be introduced in this book.

Corollary 5.1. *The set $\{A \mid A \in \mathscr{A}, \Pi \vdash A\}$ is undecidable.*

Proof. Suppose that $\{A \mid A \in \mathscr{A}, \Pi \vdash A\}$ is decidable. According to the Definition 5.4, $g(n)$ is decidable. This is a contradiction to Lemma 5.4. □

In the proof of Lemma 5.3, $G[S^{\&A}0]$ stands for A being provable in Π. According to our discussions on self-referential statements in Section 5.2, $\mathring{A}$ could be interpreted as

"This sentence is unprovable in Π."

Theorem 5.2 (Incompleteness of Π). *The theory of elementary arithmetic* Π *is an incomplete formal theory.*

Proof. We prove the theorem by contradiction. Suppose that Π is complete. Then according to Lemma 5.2, $Th(\Pi)$ is decidable and thus the relation $g(n)$ is decidable. This contradicts Lemma 5.4. □

Theorem 5.3 (Gödel's incompleteness theorem). *If* Γ *is a recursively enumerable formal theory that contains the theory of elementary arithmetic* Π*, then* Γ *is an incomplete formal theory.*

Proof. The proof is similar to but more complicated than that of the incompleteness of Π. □

Gödel's incompleteness theorem has at least the following implications: if one wants to establish a finite and consistent axiom system on a domain, which can be described by first-order languages and in which the additive and multiplicative operations of natural numbers are indispensable, then there exists a proposition about this domain such that neither the proposition itself nor its negation is a logical consequence of the axiom system. In other words, every such axiom system is incomplete. Since the arithmetic operations $+$ and $\times$ of natural numbers are a prerequisite for numerical calculation, and an axiom system has to be consistent, the incompleteness of an axiom system containing the theory of elementary arithmetic is inevitable. Thus Gödel's incompleteness theorem reveals the essential limitations of the axiomatic approach.

5.5 Gödel's consistency theorem

In this section we demonstrate Gödel's consistency theorem, i.e., for any formal theory containing Π, it is impossible to prove the consistency of the formal theory using formal inference systems with the formal theory itself as the premise. We shall show the method of the general proof by proving the result for the theory of elementary arithmetic Π using the **G** system.

In proving Gödel's incompleteness theorem, the key step was to find a suitable formula in $\mathscr{A}$ to describe "*this formula is unprovable in* Π." In the same way the key step in proving the consistency theorem is to find a formula Q in $\mathscr{A}$ to express the proposition "Π *is consistent*." If we can find such a Q, then to prove Gödel's consistency theorem, it suffices to prove that $\Pi \vdash Q$ is unprovable.

Recall, in Section 5.2 we introduced a method for generating the Gödel numbers of sequents and proof trees and used $\&tr(\Gamma \vdash A)$ to represent the Gödel number of the proof tree of the sequent $\Gamma \vdash A$.

Now consider the following binary relation on $\mathbb{N}$.

Definition 5.5 ($h(n,m)$). Let $h(n,m)$ be a binary relation on $\mathbb{N}$. For any $A \in Th(\Pi)$,

$$h(n,m) = \begin{cases} 1, & \text{if } n = \&A,\ \ m = \&tr(\Pi \vdash A), \\ 0, & \text{otherwise.} \end{cases} \tag{5.33}$$

Definition 5.5 indicates that the relation $h(\&A,m)$ holds if and only if m is the Gödel number of the proof tree of $\Pi \vdash A$.

Lemma 5.5. *$h(n,m)$ is a decidable binary relation on $\mathbb{N}$.*

Proof. By the definitions of the Gödel numbers of formulas, sequents and proof trees, we can design a P-procedure $H(x_1,x_2)$ with two formal parameters.

Suppose that when the procedure is called, the first real parameter is n whereas the second is m. The procedure first checks whether n is the Gödel number of some formula A.

If it is not, then the procedure outputs 0 and halts. Otherwise, the procedure checks whether m is the Gödel number of some proof tree $tr(\Pi \vdash A)$. If it is, then the procedure outputs 1 and halts; otherwise, it outputs 0. □

Since $h(n,m)$ is a decidable relation on $\mathbb{N}$, according to Theorem 4.3, it is representable in Π. Let the formula $B(x,y)$ be the representation of $h(n,m)$ in the first-order language $\mathscr{A}$. Then we have:

$$\begin{aligned} &\text{if } h(n,m) \text{ holds, then } \Pi \vdash B[S^n0, S^m0] \text{ is provable;} \\ &\text{if } h(n,m) \text{ does not hold, then } \Pi \vdash \neg B[S^n0, S^m0] \text{ is provable.} \end{aligned}$$

Next let us discuss how to describe the consistency of Π using the formulas in $\mathscr{A}$. According to (2) of Lemma 3.7 in Chapter 3,

$$\Pi \text{ is consistent if and only if } \Pi \vdash P \wedge \neg P \text{ is unprovable.}$$

Here P is a formula in $\mathscr{A}$.

Definition 5.6 (Sentence Q). Suppose that the formula $B(x,y)$ of $\mathscr{A}$ represents the binary relation $h(n,m)$ in Π and the formula $C(x)$ is $\exists yB(x,y)$. Let the sentence Q be

$$\neg C[S^{\&(P\wedge\neg P)}0], \quad \text{i.e.,} \quad \neg\exists yB(S^{\&(P\wedge\neg P)}0, y).$$

Here $\&(P \wedge \neg P)$ denotes the Gödel number of $P \wedge \neg P$.

The sentence $\neg C[S^{\&(P\wedge\neg P)}0]$ is interpreted in the model $\mathbf{N}$ as: for a formula P in $\mathscr{A}$, the proof tree of the sequent $\Pi \vdash P \wedge \neg P$ does not exist. This amounts to $\Pi \vdash P \wedge \neg P$ being unprovable. Hence the sentence Q describes the consistency of the formal theory Π. Thus to prove that "*the consistency of* Π *is unprovable in* Π," it suffices to prove that

$$\Pi \vdash Q \text{ is unprovable, i.e., } \Pi \vdash \neg C[S^{\&(P\wedge\neg P)}0] \text{ is unprovable.}$$

To prove this conclusion we need the following lemma.

Lemma 5.6. *Suppose that the sentence $\mathring{A}$ is a solution of the fixed point equation $\Pi \vdash A \leftrightarrow \neg C[S^{\&A}0]$. Then both $\Pi \vdash \mathring{A}$ and $\Pi \vdash \neg C[S^{\&\mathring{A}}0]$ are unprovable.*

Proof. We prove the lemma by contradiction. Suppose that $\Pi \vdash \mathring{A}$ is provable. By the definition of $h(n,m)$, there exists $m = \&(tr(\Pi \vdash \mathring{A}))$ such that $h(\&\mathring{A}, m)$ holds. According to Theorem 4.3 on representability and the $\exists$-*R* rule, $\Pi \vdash C[S^{\&\mathring{A}}0]$ is provable. Since $\Pi \vdash \mathring{A} \leftrightarrow \neg C[S^{\&\mathring{A}}0]$ holds and we also suppose that $\Pi \vdash \mathring{A}$ is provable, $\Pi \vdash \neg C[S^{\&\mathring{A}}0]$ is provable, which contradicts the consistency of Π. Thus $\Pi \vdash \mathring{A}$ is unprovable. □

By Definition 5.6, Q describes the consistency of Π. The unprovability of $\Pi \vdash \mathring{A}$ can be described by $\neg C[S^{\&\mathring{A}}0]$. Since Lemma 5.6 has been proved, we can use the method of Gödel coding to describe and prove the conclusion of Lemma 5.6, which amounts to the sequent

$$\Pi \vdash (Q \rightarrow \neg C[S^{\&\mathring{A}}0]) \tag{5.34}$$

being provable. After these preparations, we can now prove the following theorem.

Theorem 5.4 (Consistency of Π). *The consistency of the theory of elementary arithmetic Π cannot be proved using the **G** system with Π itself as the premise.*

Proof. Since the consistency of Π is described by Q, it suffices to prove that $\Pi \vdash Q$ is unprovable. We prove this by contradiction. Suppose that $\Pi \vdash Q$ is provable. Applying the modus ponens rule to this sequent and the sequent (5.34), we get the result that $\Pi \vdash \neg C[S^{\&\mathring{A}}0]$ is provable. This contradicts Lemma 5.6. The contradiction is caused by assuming that $\Pi \vdash Q$ is provable. Hence $\Pi \vdash Q$ is unprovable. □

Theorem 5.5 (Gödel's consistency theorem). *If a formal theory Γ contains the theory of elementary arithmetic Π, then the consistency of Γ cannot be proved formally by taking Γ as promise.*

Proof. The proof is similar to that of Theorem 5.4. □

We need to clarify the following three points about Gödel's consistency theorem.

Firstly, Gödel's consistency theorem coincides with our experience. Reviewing the previous chapters, we find that whenever we proved the consistency of some formal theory, we never proved its consistency by inference rules, starting from the axioms contained in the formal theory. In general, we used the method of model checking to prove its satisfiability, from which its consistency can be deduced.

Secondly, the significance of Gödel's consistency theorem is as follows. Suppose that some domain can be described by a first-order language and it uses arithmetic. Also suppose that the axiom system describing the domain knowledge is recursively enumerable. Then the consistency of the axiom system cannot be proved only by the process of formal inference from the axiom system. This theorem further reveals the limitations of the axiomatic approach to structuring domain knowledge. It also shows that it is a profound and difficult problem to determine whether an axiom system is consistent.

Finally, a software system can be viewed as a formal system, or at least its specification can, to some extent, be described by a formal theory containing the theory of

elementary arithmetic Π. One of the main tasks in software development is to determine whether the software system is consistent, reliable and satisfies its design requirements. Gödel's consistency theorem tells us that, to answer this question, one has to use methods and theories different to logical inference from the specification. This is the reason that model checking is widely used in practical software engineering, and the development of new methods for model checking has become a hot and sustained research topic in computer science.

5.6 Halting problem

The halting problem is a famous undecidable problem. Turing defined computability and invented the Turing machine shortly after Gödel proved his incompleteness theorem. Furthermore, Turing proved that the halting problem is an undecidable problem. In fact, the method of his proof was inspired by Gödel. The halting problem asks whether there exists a P-procedure G such that for every P-procedure F, G can determine whether F halts. In this section, we shall prove the undecidability of the halting problem.

In order to determine whether an arbitrary P-procedure F is a halting procedure, the P-procedure G has to take F as input and to halt. In the process, it has to determine whether F is a halting procedure. Hence it is necessary to define formally: "the P-procedure G takes the P-procedure F as input." Using the idea of Gödel coding, this can be done only if we have a proper coding for P-procedures. As we remarked in Section 5.2, the inputs of the P-procedure G can only be symbol strings. Therefore, if we want to take every P-procedure as input, we have to find a coding method that transforms every P-procedure into a symbol string, which we shall call the code of the P-procedure. Then we will take the code of each P-procedure as the input of the P-procedure G. Once the coding method is defined, all the codes of the halting P-procedures will constitute a set $\Upsilon^\sharp$ of symbol strings. Proving that the halting problem is an undecidable problem amounts to proving that $\Upsilon^\sharp$ is an undecidable set.

Definition 5.7 (Character set A). The character set $\mathbf{A}_1$ of P-procedures is composed of the following boldfaced character strings:

$$\mathbf{A}_1 = \{\textbf{procedure}, \textbf{begin}, \textbf{end}, \textbf{if}, \textbf{then}, \textbf{else}, \textbf{while}, \textbf{do}, \ldots\}.$$

Let

$$\begin{aligned}\mathbf{A}_2 = {} & \{A,B,C,\ldots,X,Y,Z\} \cup \{a,b,c,\ldots,x,y,z\} \cup \{\Gamma,\Delta,\ldots,\Omega\} \\ & \cup \{0,1,\ldots,9\} \cup \{:=,+,-,\cdot,<,=,:,;,|,(,),\{,\}\} \cup \{,\}\end{aligned}$$

which is a set of characters and define

$$\mathbf{A} = \mathbf{A}_1 \cup \mathbf{A}_2.$$

According to the above definition, each P-procedure F is a character string of $\mathbf{A}^*$. We call this character string the *character string of* F.

Example 5.7 (*P*-procedure).

```
procedure ABC(w: string)
begin
    while 0 < w  do w := w+ ı;
end
```

If the above *P*-procedure is written as one line, then it has the following form:

$$\textbf{procedure}\ \mathrm{ABC}(w : \mathrm{string})\ \textbf{begin while}\ 0 < w\ \textbf{do}\ w := w+ \imath;\textbf{end}$$

Hence the *P*-procedure ABC is a character string of $\mathbf{A}^*$. It is also called a *string of* $\mathbf{A}^*$.

Definition 5.8 ($\mathbf{P}^*$). $\mathbf{P}^*$ is the set consisting of the strings of all the *P*-procedures.

Obviously, $\mathbf{P}^*$ is a subset of $\mathbf{A}^*$.

Lemma 5.7. *The set* $\mathbf{P}^*$ *is decidable with respect to* $\mathbf{A}^*$.

Proof. The syntactic analysis program of *P*-procedures is a program to decide $\mathbf{P}^*$. □

According to the previous discussion in this chapter, both $\mathbf{A}^*$ and $\mathbf{P}^*$ are recursively enumerable. Hereafter, we shall use the lexicographic order to enumerate the strings of *P*-procedures.

Definition 5.9 (Coding of *P*-procedures). If u_n is the nth string of the sequence $\mathbf{A}^*$, then let

$$w_{u_n} = \underbrace{\imath\imath\cdots\imath}_{n\ times}$$

be the coding of u_n.

If the string of a *P*-procedure F is u_n, then the coding of F is defined as w_{u_n} and denoted as w_F.

Hereafter we shall use $\imath^n$ to denote $\underbrace{\imath\imath\cdots\imath}_{n\ times}$ and let $0 < \imath$. And $\imath^m < \imath^n$ if and only if $m < n$ holds.

Definition 5.10 (Coding set Υ). The coding set Υ is composed of all the codings of *P*-procedures, i.e.,

$$\Upsilon = \{w_F \mid F \in \mathbf{P}^*\}.$$

Lemma 5.8. *The set* Υ *is decidable with respect to* $\{\imath\}^*$.

Proof. Let $w \in \{\imath\}^*$ have length n. Since $\mathbf{A}^*$ is recursively enumerable, we can enumerate the nth string u_n of $\mathbf{A}^*$ in the lexicographic order. Since $\mathbf{P}^*$ is decidable, we can decide whether $u_n \in \mathbf{P}^*$ holds. Thus with w_{u_n} being the coding of u_n, we can decide whether $w_{u_n} \in \Upsilon$ holds. □

Since the coding of each P-procedure F is an element of the string set $\{|\}^*$, i.e., a word, we can design a P-procedure G whose input is the coding of F. Evidently the P-procedure G can take its own coding w_G as input. When G takes its own coding w_G as input, it either halts or does not halt. We use Υ^+ to denote the set of codings of P-procedures that halt after taking their own codings as inputs.

Definition 5.11 (Υ^+)**.**

$$\Upsilon^+ = \{w_F \mid F \in \mathbf{P}^* \text{ and } F : w_F \longmapsto \square\}.$$

Here Υ^+ is a set consisting of the codings of P-procedures. Each element of the set is a coding of some P-procedure, which halts after taking its own coding as input. In what follows we shall prove that Υ^+ is undecidable.

Lemma 5.9. *The set Υ^+ is undecidable.*

Proof. We prove the lemma by contradiction. For simplicity, in this proof we only consider whether a P-procedure halts. Suppose that there exists a P-procedure F_0 which can decide Υ^+, and for any P-procedure F, we have:

$$\begin{aligned} &\text{if } F : w_F \longmapsto \square \text{ then } F_0 : w_F \longmapsto \square; \\ &\text{if } F : w_F \longmapsto \bot \text{ then } F_0 : w_F \longmapsto |\,. \end{aligned}$$

Using F_0 as basis, we employ the following method to construct a P-procedure F_1. If F halts after taking w_F as input, and F_0 stores its output in the variable x with the output statement being **print** x, then F_1 uses the **while** statement

$$\textbf{while } 0 < | \textbf{ do } x := x+|$$

instead of the statement **print** x in F_0. In this way, the F_1 obtained by revising F_0 does not halt. If F does not halt after taking w_F as input, according to the assumption, F_0 halts and outputs $|$ after it takes w_F as input. In this case let F_1 output $\square$.

The P-procedure F_1 defined in this way has the following property: for every P-procedure F,

$$\begin{aligned} &\text{if } F : w_F \longmapsto \square \text{ then } F_1 : w_F \longmapsto \bot; \\ &\text{if } F : w_F \longmapsto \bot \text{ then } F_1 : w_F \longmapsto \square. \end{aligned}$$

Thus for every P-procedure F,

$$F : w_F \longmapsto \square \quad \text{if and only if} \quad F_1 : w_F \longmapsto \bot.$$

In particular, let $F = F_1$. Then we have:

$$F_1 : w_{F_1} \longmapsto \square \quad \text{if and only if} \quad F_1 : w_{F_1} \longmapsto \bot,$$

which is a contradiction. This contradiction is caused by assuming that there exists a P-procedure F_0 that can decide the set Υ^+. Thus Υ^+ is undecidable. $\square$

The above lemma shows the undecidability of Υ^+. Υ^+ is the set of P-procedures that halt after taking their own codings as inputs. By proving the undecidability of the set of halting P-procedures without input, we shall prove that the set of codings of all the halting P-procedures is undecidable.

Definition 5.12 ($\Upsilon^\sharp$)**.**

$$\Upsilon^\sharp = \{w_F \mid F \in \mathbf{P}^* \text{ and } F: \ \longmapsto \square\}.$$

Obviously $\Upsilon^\sharp$ is a countable set. If we can prove that $\Upsilon^\sharp$ is undecidable, then the undecidability of the set of halting P-procedures is proved.

Lemma 5.10. *For every P-procedure F, we can design a P-procedure F' such that*

$$F: \ w_F \longmapsto \square \quad \textit{if and only if} \quad F': \ \longmapsto \square.$$

Proof. Without loss of generality, suppose that the formal parameter of the P-procedure F is x. Since Υ is decidable, we can design a P-procedure F'' such that, each time it executes, it generates a coding of F, i.e., $w_F = |^n$.

The P-procedure F' can be designed as follows. F' first calls the procedure F'' before it executes the following assignment statement:

$$x := w_F.$$

Then it starts to execute the original statements of F. Hence we can prove that $F: \ w_F \longmapsto \square$ if and only if $F': \ \longmapsto \square$. □

Finally, let us prove that $\Upsilon^\sharp$ is undecidable.

Theorem 5.6. *The set $\Upsilon^\sharp$ is undecidable.*

Proof. We prove the theorem by contradiction. Suppose that $\Upsilon^\sharp$ is decidable and let the P-procedure that determines the decidability of $\Upsilon^\sharp$ be F_1. Since Υ is decidable, for any $w \in \{|\}^*$, we first use the deciding program of Υ to decide whether w belongs to Υ. If $w \notin \Upsilon$, then the program outputs $\square$. If $w \in \Upsilon$, then w is the coding of some P-procedure F, i.e., $w = w_F$ holds. We apply the constructive method in the proof of Lemma 5.10 to F to generate a P-procedure F'. Since we assume that $\Upsilon^\sharp$ is decidable, $w_{F'} \in \Upsilon^\sharp$ holds if and only if $w_F \in \Upsilon^+$ holds after F_1 takes $w_{F'}$ as input and executes a finite number of steps, as in Lemma 5.10. Hence Υ^+ is decidable, which contradicts Lemma 5.9. Thus $\Upsilon^\sharp$ is undecidable. □

Our first four chapters have shown a possible approach to theoretical research in mathematics and natural science.

Before first-order languages are introduced, each mathematical or scientific theory is a domain that is usually composed of constants, variables, functions and propositions, which describe the meaningful knowledge in the domain. All the true propositions can be further divided into axioms and corollaries. Each axiom is a proposition that is assumed to be true and, together, the axioms form an axiom system. The corollaries are logical

consequences of the axiom system. The goal of research is to obtain all logical consequences of the axiom system and to clarify the logical structure of the domain. In this sense, all scientific theories can be viewed, in some way, as mathematical systems which, in Chapter 2, we called domains.

The logical connectives and quantifiers in propositions determine the logical structure of the theory. The same logical connective or quantifier has the same meaning in all domains. The logical inference rules with respect to these are also valid in all theories. The logical analysis of the propositions contained in a theory is realized through invoking the inference rules. Hence, according to the basic soundness assumption, the logical consequences of a theory are proved consequences, which can be obtained by mathematical proof. This is called the axiomatic approach, which was first introduced into mathematics at the beginning of the 20th Century and propagated into various areas of natural science.

In general, we still need to discover whether the soundness assumption is reliable and how to guarantee that the set of logical inference rules is complete. Furthermore, the construction of a mathematical proof is usually a hard job.

If we can specify a theory in a first-order language, then the process of logical inference in the theory can be replaced by a formal inference system, for example, the **G** system, which is a symbolic calculus. In this case, the basic soundness and completeness of the theory is guaranteed by the theorems of Chapter 3. This converts mathematical proof in the theory into a procedure manipulating symbols, which can be accomplished by computers within an interactive software system. This is called the formal approach. The formal approach provides a mathematical foundation for the development of digital systems and for building the information society. However, Gödel's theorems show the limitation of this approach. It can never capture the complete knowledge of any domain that needs arithmetic.

Chapter 6

Sequences of Formal Theories

The process of scientific research follows a general pattern. Firstly, observations are made and data is gathered. Secondly, patterns are extracted from these observations and generalized by induction into propositions. Thirdly, these propositions are analyzed to find their logical consequences and relationships. The basic propositions can be seen as axioms and the axioms and their consequences together form a theory.

The fourth and vital phase of research is making predictions from the logical consequences of a theory. A theory is only accepted if, as well as explaining known facts, it can predict new phenomena which can be tested by experiment. If these experiments contradict the predictions, then the theory is refuted and it needs to be revised by getting rid of some axioms and devising new ones that both fit with the data and do not result in refutable predictions. This produces a new version of the theory.

In summary, the evolution of a theory is formed by making propositions via induction, establishing axiom systems, carrying out logical analysis on propositions, checking the consistency of logical consequences with observed data, and making revisions according to refutation by facts. A scientific theory is not a fixed object but a dynamic process, using prediction and refutation to produce more reliable versions of the theory. This process is iterative and each version produces new predictions to be tested and refuted, resulting in a sequence of theories, which gradually approach closer to the truth about the domain.

Chapters 1 to 5 have analyzed in depth the process of logical analysis by introducing first-order languages and formal inference systems. We saw how the process of deduction can be distilled into a symbolic calculus, which can, to some extent, be mechanized. The soundness and completeness of first-order languages guarantees this methodology produces logical consequences, but Gödel's theorems also warn us that it has limitations.

However, the process of analyzing experiments, inducing axioms from them and revising a theory when it is refuted, has needed complex intellectual work, which involves experience, intuition and the ability to synthesize observations into generalizations.

Nevertheless, we might hope that we can formalize some of this process into a symbolic calculus to complement the deductive system presented in previous chapters. Chapters 6 to 9 do just this and encapsulate the inductive process of generating new axioms and the process of revising a theory as formal procedures. Furthermore, we will also formalize the description of research methodology and analyze the properties of sequences of theory versions.

In this chapter we introduce the concepts of a sequence of formal theories, the limit of such a sequence and a proscheme. The latter is a word coined by the author, to combine the meanings of procedure and schema. In Chapter 7, we shall define refutation by facts

using the concept of model, develop a revision calculus for formal theories, and prove the reachability, soundness, and completeness of this system. The three fundamental properties that a reliable proscheme should possess will be presented in Chapter 8. In Chapter 9, a system of formal calculus for inductive inference will be given and its reliability will be proved.

6.1 Two examples

In this section, we will illustrate, through two examples, the concepts we need in order to formalize the description of the axiomatic process.

Example 6.1 (Software development). Every software product exists in the form of versions. Although each version of a software product might be extremely large and complicated, it is a formal system consisting of programs in some programming language. Each version of a software system is the end result of a stage of its development. Hence software development is also a process generating a sequence of versions of the software. Take the Microsoft Windows operating system as an example. By releasing the versions of Windows one after another, Microsoft gradually improves and refines the personal computing environment that it produces. The versions of Windows form the following sequence:

$$\text{Windows}\,1.0, \ldots, \text{Windows}\,3.1, \ldots, \text{Windows}\,95, \ldots, \text{Windows}\,98, \ldots, \text{Windows}\,2000, \ldots$$

This sequence is a production record of the development of Windows. In fact, there were many more versions than these. The above versions are those which formed final released products. In the production of each version, many internal versions may be generated which are labeled in a format like: [major version].[minor version].[build number]. For example, a release of Windows Vista had a version number of "6.0.6000", and one evaluation release of Windows 7 had version number "6.1.7100". In the active development stage of commercial software, new versions are usually generated on a daily basis, each being stamped with a unique version number. The ideal Windows, in the minds of its designers, is gradually realized by constantly improving the functionality version by version. In other words, the ideal Windows is the limit of the version sequence.

In software development, a new version is usually generated for the following reasons.

1. The developers may want to provide new services to the customers. For instance, Windows 95 saw the introduction of email and internet services, and programs to exploit them, such as Internet Explorer.
2. Bugs have been found in the current version that need to be corrected. They say that tens of thousands of bugs were fixed between the beta version of Windows 2000 and its final release version.

Each version of Windows is a formal system in the form of a set of software programs, whereas its functional specifications can be seen as a first-order formal theory.

Although this might be extremely large and complicated, these evolving specifications form a sequence of formal theories. The specifications for the ideal Windows could be described as the limit of this sequence.

Adding new functionality to a version amounts to adding new program modules. In this book we refer to the new functions as new axioms for the version, and we sometimes call these new rules. A bug can be regarded as a refutation of the functionality of the version and is a counterexample for it. In this book, we call such a counterexample a refutation by facts of the version.

There are two kinds of refutation by facts:

1. customers find that the functionality of a software system does not match the specifications;
2. the functionality of the system is different from the requirements of its customers although it matches its specifications.

The major reasons for generating a new version of the software system are to add new rules to a version and to revise a version according to the refutations by facts. In order to make a formal description of the software development process or the axiomatization process of software specifications, it is necessary to clearly define the concepts of 'new axioms' and 'refutations by facts' on a formal theory.

From the above discussion we can see that the new rules and refutations by facts are only concerned with the specific usage of a software. The previous chapters indicate that their theoretical context is the model theory of the specifications. Therefore, the formal description of these two concepts requires a more detailed study of model-theory.

Software Engineers have found that the generation of a new version of software, such as Windows, is best performed in an overall development framework such as the *Microsoft Solution Framework* (MSF). Such a framework is actually a kind of development schema for software. Under the guidelines of MSF or other frameworks, members of a software development team find bugs by testing data rationally and then correcting these bugs efficiently and cooperatively within the team. MSF has unambiguous definitions of, not only the design and implementation of the new functionality of every version, but also the configuration and generation of the new versions. There are also numerous auxiliary software tools for design, testing and correction of errors. MSF is only one of many such frameworks. These development methods, strategies and schemas are called *proschemes*. In what follows, we will investigate the properties of proschemes at the theoretical level of first-order languages.

For the second example, let us take the process of evolution for physics, which Einstein described in *Relativity: The Special & The General Theory* [Einstein, 1921]. Following the above discussions, we can call this process the *axiomatization process for physics*.

Example 6.2 (Evolution of physics). The course of development of physics from Galileo to the theory of relativity of Einstein can be divided into the following four phases.

Phase 1. This is the phase of physics before Galileo. Let us use Γ_1 to denote all the physical principles and laws that had been understood by human beings up to this time.

Phase 2. Galileo was perhaps one of the first scientists to formulate laws on the basis of observations and experiments. He discovered physical laws such as the Galilean transformation **V**, which specifies the velocity of a moving object in different coordinate systems. And he also proposed the principle of relativity **R** in his famous book: *Dialogo Sopra i due massimi Sistemi del mondo, tolemaico e copernico*.

The principle of relativity states that, if different coordinate systems are relatively in uniform rectilinear motion, then a physical law expressed in one system has the same mathematical form when transformed into the other.

The Galilean transformation states that the velocities of an object measured in different coordinate systems are related to the relative velocities between the systems. In what follows let us use first-order language to describe the Galilean transformation. Let the predicate $B(x)$ denote "x is an object" and the predicate $A(x)$ denote "if the velocity of x relative to the coordinate system K is $\overrightarrow{V}$, and the velocity of the coordinate system K relative to K' is $\overrightarrow{W}$, then the velocity of x relative to K' is $\overrightarrow{V}+\overrightarrow{W}$." The Galilean transformation can be described by the formula

$$\mathbf{V}:\ \forall x(B(x)\rightarrow A(x)).$$

Galileo expanded Γ_1 by introducing **R** and **V** into physics as new principles. This led to the formation of a new version Γ_2 of physics that was later called *Galilean physics*, i.e.,

$$\Gamma_2=\Gamma_1\cup\{\mathbf{R},\mathbf{V}\}.$$

R and **V** are examples of *new axioms* added to Γ_1, whereas the new version, Γ_2, is called an *N-type version* of Γ_1.

Phase 3. Following from the work of Galileo, Kepler and others, Newton propounded three laws $\mathbf{N}_1$, $\mathbf{N}_2$, $\mathbf{N}_3$ of motion and the law **E** of universal gravitation. Since these laws are not contradictory to Galilean physics Γ_2, Newton introduced them as new axioms to form a new version Γ_3 of physics. This became known as *classical physics* or *Newtonian physics*, i.e.,

$$\Gamma_3=\Gamma_2\cup\{\mathbf{N}_1,\mathbf{N}_2,\mathbf{N}_3,\mathbf{E}\}.$$

The version Γ_3 is an N-type version of Γ_2.

Phase 4. Classical physics was widely accepted and used for almost two hundred years. It satisfactorily explained existing observations and successfully predicted new phenomena such as the existence of Neptune. It was not until the end of the 19th Century, when people tried to measure the velocity of light, that the discrepancy between the computations of classical physics and the observed results of experiments was found. More specifically, if we regard light as a photon which is a particle and denote it as c, then $B(c)$ holds. According to the Galilean transformation, we can deduce $A(c)$ using the modus ponens rule, that is:

$$\Gamma_3, B(c), \forall x(B(x)\rightarrow A(x))\vdash A(c).$$

Here $A(c)$ can be interpreted as: if the velocity of a photon in a coordinate system K is $\overrightarrow{C}$, and the velocity of a coordinate system K relative to K' is $\overrightarrow{W}$, then the velocity of

the photon in K' is $\overrightarrow{C} + \overrightarrow{W}$. This is the prediction of classical physics: "the velocity of light observed by an observer in the coordinate system K' is subject to the changes of the velocity of K relative to K'."

However this prediction contradicted the experiments performed to measure the velocity of light. Those experiments supported its opposite $\neg A(c)$, i.e., the velocity of light does not depend on the velocity of the body emitting the light. Under such circumstances, we say that the sentence $A(c)$ was refuted by experiment, or $\neg A(c)$ forms a refutation by facts.

So classical physics was challenged by refutations by facts and hence the Newtonian version Γ_3 of physics had to be revised so that it became consistent with the experiments.

Using brilliant logical intuition, Einstein concluded that the Galilean transformation had to be deleted. In fact, by the **G** system,

$$B(c), \neg A(c) \vdash \neg \forall x(B(x) \to A(x))$$

is provable, that is, $B(c), \neg A(c)$ is inconsistent with $\forall x(B(x) \to A(x))$. Thus, if $\neg A(c)$ is accepted, then the Galilean transformation has to be deleted from classical physics Γ_3. Furthermore, $\neg A(c)$, the constancy of the velocity of light should be accepted as a new principle, **O**.

The revision of classical physics Γ_3 can be accomplished in two steps. The first step is to delete the Galilean transformation from Γ_3 so as to obtain a new version Γ_4 of physics:

$$\Gamma_4 = \Gamma_3 - \{\mathbf{V}\}.$$

We call the version Γ_4 an *R-type version* of Γ_3, where R is used because it is the first letter of *revision*.

The second step is to expand Γ_4 by adding the principle **O**. From this principle, Einstein showed that the Lorentz transformation **L** was a suitable replacement of Galilean transformation in order to retain the principle of relativity. This led to

$$\Gamma_5 = \Gamma_4 \cup \{\mathbf{O}, \mathbf{L}\}.$$

Γ_5 retains the principle of relativity, Newton's three laws of motion, and the law of universal gravitation but the Galilean transformation is deleted. The constancy of the velocity of light was added to physics as a new law and the Galilean transformation was replaced by the Lorentz transformation. Γ_5 is a new version of physics that is called the *special theory of relativity*.

After the special theory of relativity, Einstein further proposed the idea that gravitational mass and inertial mass should be equal. He introduced this equivalence principle and produced a new version of physics, the general theory of relativity Γ_6.

Physics is still developing and in the process of axiomatization. These versions of physics in different phases form the following sequence in order of their appearance:

$$\Gamma_1, \Gamma_2, \Gamma_3, \Gamma_4, \Gamma_5, \Gamma_6, \ldots.$$

The truth of physics is the limit of this sequence. From the above brief description of the evolution of physics, we can see the common approach shared by Galileo, Newton,

and Einstein to the axiomatization process. That is, they propounded new laws according to experimental results, revised the existing versions according to whether these new laws are new axioms or refutations by facts and formed sequence of versions in different phases, which describes the evolution of physics.

This version sequence gradually approaches the truth of physics. Such an approach is a kind of proscheme of the axiomatization process of physics. Scientific research follows many other different approaches and schemas. From the abstract level of first-order languages, most of them can be described, as in the above example, by the proschemes of the axiomatization processes.

Scientific research is an art using insight and experience. However, the deletion and addition of principles in response to experiments can be formalized into a symbolic calculus and viewed as an inference system. This is the aim of the following chapters.

The working paradigms of scientific research can, to some extent, also be expressed as one of the mathematical mechanisms we have called proschemes. We will treat the proschemes as an indispensable part of the axiomatization process and study their properties.

The above two examples show us that the concepts and methods introduced in the previous chapters are insufficient to describe this process. So new axioms, refutations by facts, revisions of formal theories, sequences of formal theories, limits of sequences and proschemes will be new features added into the framework of first-order languages.

6.2 Sequences of formal theories

The concept of a sequence of formal theories is indispensable in describing the evolution of domain knowledge. As we have seen in Section 6.1, the versions of a software system form a sequence of formal systems, whereas the versions of the specifications form a sequence of formal theories. The versions of physics in different historical phases form the theory sequence of physics. Generally speaking, for every theory of mathematics and natural science, the versions in different phases of its development form a sequence of scientific theories. Sequences of formal theories are abstract descriptions of these examples, while, on the other hand, these examples are models of formal sequences.

Definition 6.1 (Sequence of formal theories). If for every natural number n, Γ_n is a formal theory, then we call

$$\Gamma_1, \Gamma_2, \ldots, \Gamma_n, \ldots$$

a *sequence of formal theories*, or a *sequence* for short. The sequence is denoted as $\{\Gamma_n\}$.

If for every natural number n, $\Gamma_n \subseteq \Gamma_{n+1}$ (or $\Gamma_n \supseteq \Gamma_{n+1}$) holds, then we call the sequence an *increasing sequence* (or *decreasing sequence*). A sequence that is neither increasing nor decreasing is a *non-monotonic sequence*.

Before continuing, we would like to note that, hereafter, whenever mentioning a sentence P, we refer to the equivalence class consisting of the sentences that are equivalent to P, i.e., $P \leftrightarrow Q$. The representative element of the equivalence class is P.

In the following definitions Γ_n will be regarded as a set of sentences with $n \in \mathbb{N}$.

Definition 6.2 (Limits of sequences). Let $\{\Gamma_n\}$ be a sequence of formal theories. We call the set

$$\{\Gamma_n\}^* = \bigcap_{n=1}^{\infty} \bigcup_{m=n}^{\infty} \Gamma_m$$

the *upper limit* of the sequence $\{\Gamma_n\}$ and the set

$$\{\Gamma_n\}_* = \bigcup_{n=1}^{\infty} \bigcap_{m=n}^{\infty} \Gamma_m$$

the *lower limit* of the sequence $\{\Gamma_n\}$.

If the set $\{\Gamma_n\}_*$ of sentences is consistent and

$$\{\Gamma_n\}_* = \{\Gamma_n\}^*,$$

then we say that the sequence $\{\Gamma_n\}$ is convergent and its *limit* is its upper (or lower) limit, which is denoted as

$$\lim_{n\to\infty} \Gamma_n.$$

The following lemma explains the meanings of the upper and lower limits of a sequence.

Lemma 6.1.

(1) $A \in \{\Gamma_n\}^*$ *if and only if there exist infinitely many natural numbers* k_n *such that* $A \in \Gamma_{k_n}$.

(2) $A \in \{\Gamma_n\}_*$ *if and only if there exists a natural number* N *such that for every natural number* m *satisfying* $m > N$, $A \in \Gamma_m$.

Proof. (1) $A \in \{\Gamma_n\}^*$ if and only if for any $n \geqslant 1$, $A \in \bigcup_{m=n}^{\infty} \Gamma_m$. This holds if and only if there exists a $k_n \geqslant n$ such that $A_n \in \Gamma_{k_n}$.

(2) The proof is similar to (1). Readers may prove it by themselves. □

Lemma 6.2.

(1) If a sequence $\{\Gamma_n\}$ *is increasing, then it is convergent with its limit being* $\bigcup_{n=1}^{\infty} \Gamma_n$.

(2) If the sequence is decreasing, then it is also convergent with its limit being $\bigcap_{n=1}^{\infty} \Gamma_n$.

Proof. (1) It follows easily from $\{\Gamma_n\}$ being increasing that for any $m \geqslant 1$,

$$\bigcup_{n=m}^{\infty} \Gamma_n = \bigcup_{n=1}^{\infty} \Gamma_n. \text{ Hence } \{\Gamma_n\}^* = \bigcup_{n=1}^{\infty} \Gamma_n.$$

Also it follows from $\{\Gamma_n\}$ being increasing that

$$\bigcap_{m=n}^{\infty} \Gamma_m = \Gamma_n. \text{ Hence } \{\Gamma_n\}_* = \bigcup_{n=1}^{\infty} \Gamma_n.$$

(2) The proof is similar to (1) and readers may prove it by themselves. □

Hereafter the closure of a formal theory Γ will often be used. The following lemma holds on the limits of closures.

Lemma 6.3.

$$\{Th(\Gamma_n)\}_* = Th(\{Th(\Gamma_n)\}_*).$$

Proof. Since $\{Th(\Gamma_n)\}_* \subseteq Th(\{Th(\Gamma_n)\}_*)$ holds, we only need to prove that

$$Th(\{Th(\Gamma_n)\}_*) \subseteq \{Th(\Gamma_n)\}_*.$$

For every $A \in Th(\{Th(\Gamma_n)\}_*)$, $\{Th(\Gamma_n)\}_* \vdash A$ is provable. According to the compactness theorem, there exist $A_{n_1}, \ldots, A_{n_k}$ such that $A_{n_i} \in \{Th(\Gamma_n)\}_*$ for every i and

$$\{A_{n_1}, \ldots, A_{n_k}\} \vdash A$$

is provable. By the definition of the lower limit, there must exist an $N > 0$ such that for $n > N$,

$$\{A_{n_1}, \ldots, A_{n_k}\} \subseteq Th(\Gamma_n).$$

Hence $Th(\Gamma_n) \vdash A$ is provable. According to the definition of the closure of the theory, $Th(Th(\Gamma_n)) = Th(\Gamma_n)$. Thus $A \in Th(\Gamma_n)$ for $n > N$. This amounts to $A \in \{Th(\Gamma_n)\}_*$. Therefore $Th(\{Th(\Gamma_n)\}_*) \subseteq \{Th(\Gamma_n)\}_*$. □

In what follows we give four examples of sequences of formal theories.

Example 6.3 (Constant sequence). Let A be a sentence. We call the sequence

$$\{A\}, \{A\}, \ldots, \{A\}, \ldots$$

a *constant sequence*. It is not difficult to verify that

$$\{\Gamma_n\}_* = \{A\} = \{\Gamma_n\}^*.$$

By Definition 6.2, this constant sequence is convergent with its limit being $\{A\}$.

Example 6.4 (Sequence of closures). Consider the following sequence:

$$\Gamma_1, \Gamma_2, \ldots, \Gamma_n, \ldots$$

$$\{P_1,\ P_1 \to Q\}, \{P_2,\ P_2 \to Q\}, \ldots, \{P_n,\ P_n \to Q\}, \ldots.$$

We can verify that the upper and lower limits of this sequence are

$$\{\Gamma_n\}^* = \varnothing, \quad \{\Gamma_n\}_* = \varnothing$$

respectively. By Definition 6.2, this sequence converges to the empty set. Since

$$P_n,\ P_n \to Q \vdash Q$$

is provable, and for its sequence of closures

$$Th(\Gamma_1), Th(\Gamma_2), \ldots, Th(\Gamma_n), \ldots.$$

it is not difficult to verify that

$$\{Th(\Gamma_n)\}^* = Th(\{Q\}) = \{Th(\Gamma_n)\}_*.$$

Hence the sequence of closures converges to $Th(\{Q\})$. For this example, the limit of the sequence $\{\Gamma_n\}$ is different from that of the sequence $\{Th(\Gamma_n)\}$.

Example 6.5 (Positive/negative sequence). This example provides a nonconvergent sequence. Let

$$\Gamma_n = \begin{cases} \{A\}, & \text{if } n = 2k-1, \\ \{\neg A\}, & \text{if } n = 2k, \end{cases}$$

where k is a nonzero natural number. It is not difficult to verify that $\{\Gamma_n\}^* = \{A, \neg A\}$ whereas $\{\Gamma_n\}_* = \varnothing$. The upper and lower limits of the sequence are different and the sequence $\{\Gamma_n\}$ is not convergent.

Example 6.6 (Random sequence). Let the sentence A denote "a coin is tossed and lands head up." Let Γ_n denote the result of tossing the coin the nth time. In this way the sequence $\{\Gamma_n\}$ is a random sequence with respect to A and $\neg A$. The upper and lower limits of the sequence are

$$\{\Gamma_n\}^* = \{A, \neg A\}, \quad \{\Gamma_n\}_* = \varnothing$$

respectively. Hence the sequence is not convergent. The non-convergence of the sequence shows that the rules contained in the formal theory cannot accurately describe the essential characteristics of this process. If P denotes "a coin is tossed and the probability of its head being up is 50%," and $\Gamma_n = \{P\}$, then the sequence becomes a constant sequence whose limit is $\{P\}$.

6.3 Proschemes

In the proofs of a number of important results of mathematical logic, sequences of formal theories play important roles. We will take the Lindenbaum Lemma in Chapter 3 as an example to analyze the roles of sequences of formal theories.

Example 6.7 (Lindenbaum sequence). The Lindenbaum Lemma says that every given formal theory Γ can be expanded to a maximal consistent set, that is, a maximal formal theory. This lemma plays a key role in proving the completeness of the **G** system. The idea of proving the lemma is to construct the maximal formal theory directly. Specifically, the construction proceeds as follows.

(1) Since the sentences in $\mathscr{L}$ are countable, we can organize them into a sequence:

$$A_1, A_2, \ldots, A_n, \ldots.$$

(2) We define every element in the sequence $\{\Gamma_n\}$ inductively. Let $\Gamma_1 = \Gamma$ and Γ_{n+1} be defined by Γ_n and A_n in the following way:

$$\Gamma_{n+1} = \begin{cases} \Gamma_n \bigcup \{A_n\}, & \text{if } \Gamma_n \text{ and } A_n \text{ are consistent,} \\ \Gamma_n, & \text{otherwise.} \end{cases}$$

(3) By the above definition, $\{\Gamma_n\}$ is an increasing sequence. According to Lemma 6.2 the sequence converges to its limit $\bigcup_{n=1}^{\infty}\Gamma_n$, which is the maximal formal theory containing Γ.

From this proof we can see that the maximal theory containing Γ is the limit of the sequence $\{\Gamma_n\}$ of formal theories. Every element of the sequence is defined recursively. The following is the method Lindenbaum used in defining the sequence $\{\Gamma_n\}$ of formal theories. We write it out in a form similar to a *P*-procedure.

Example 6.8.

proscheme `Lindenbaum`*(Γ_n: `theory`; A: `formula`; **var** Γ_{n+1}: `theory`)
begin
  if not $(\Gamma_n \vdash \neg A)$ **then**
    $\Gamma_{n+1} := \Gamma_n \cup \{A\}$
  else
    $\Gamma_{n+1} := \Gamma_n$
end
proscheme `Lindenbaum`(Γ: `theory`; $\{A_n\}$: `formula sequence`)
begin
  Γ': `theory`;
  print Γ;
  $n := 1$;
  while $0 < n$ **do**
    `Lindenbaum`$^*(\Gamma, A_n, \Gamma')$;
    print Γ';
    $\Gamma := \Gamma'$;
    $n := n + 1$
end

We can see that the functionality of the sub-proscheme `Lindenbaum`* is to generate a new theory Γ_{n+1}. Each call of `Lindenbaum`* has an initial theory Γ_n and a sentence A_n as inputs and it outputs a new theory Γ_{n+1}. The functionality of the main body is to input $\{A_n\}$ one by one and output the Lindenbaum sequence $\{\Gamma_n\}$.

The similarities between the proscheme and the *P*-procedure defined in Chapters 4 and 5 are as follows.

(1) They share the same structure and declarations. For instance, the above proscheme is constructed from its body and the declaration of its sub-proscheme.

(2) Their statements also share the same form. The proscheme allows the usage of the assignment statement, printing statement, **if** statement, sequential statement, **while** statement and **call** statement.

The proscheme differs from the *P*-procedure in the following aspects:

(1) More data types are allowed in the proscheme. For instance, `theory`, `formula`, and `formula sequence` in the declarations of the above proscheme denote data

types not allowed in *P*-procedures. The variable after **var** denotes an output formal parameter of the proscheme and is used to store the result. Here **var** is used to discriminate between the input formal parameter and the output formal parameter.

(2) The input of the proscheme `Lindenbaum` is an infinite sequence:

$$A_1, A_2, \ldots, A_n, \ldots.$$

For each input A_n of the proscheme `Lindenbaum`, the sub-proscheme `Lindenbaum`* must execute once and output Γ_{n+1}. As the elements of the sequence $\{A_n\}$ are input consecutively, the proscheme `Lindenbaum` outputs a sequence of formal theories:

$$\Gamma, \Gamma_2, \ldots, \Gamma_n, \ldots.$$

This output sequence is an increasing and convergent sequence.

(3) On the basis of $=$ and $<$ prescribed by the *P*-procedure in Chapter 4, the operators **and**, **or**, and **not** are added in the conditions of the **if** statement and **while** statement. In addition, the following undecidable conditions are allowed: e.g., $\Gamma \vdash A$ and **consistent**(Γ, A), i.e., Γ is consistent with A.

This is the essential difference between the proscheme and the *P*-procedure. Hence in this book we call the execution of a proscheme an *operation*. It is essentially different from *P*-procedures in that a *P*-procedure is computable while, in general, a proscheme is not.

Definition 6.3 (Proscheme). A proscheme expands the definition of a *P*-procedure in the following way:

(1) `formula` is a data type denoting legal first-order sentences. `theory` is data type denoting first-order formal theories. `formula sequence` is a data type, which denotes a sequence of sentences.

(2) The Boolean expressions allow not only $=$, $<$, $\leqslant$ and the logical operators **and**, **or**, and **not**, but also undecidable conditions such as $\Gamma \vdash A$ and **consistent**(Γ, A).

(3) The input of a proscheme can be a sequence of sentences. Its output can be a sequence of formal theories and is called the *output sequence* of versions of the formal theory.

To be consistent with our previous notation, in a proscheme, we shall use the letters A, B, C, ... to denote first-order sentences and the uppercase Greek letters Γ, Δ, Θ, Λ, ... to denote formal theories. These letters can have subscripts and superscripts. We use $\{\Gamma_n\}$ to denote a sequence of theories. The concept of proscheme is an expansion of the concept of *P*-procedure.

We would like to emphasize again, the most significant difference between a proscheme and a *P*-procedure is that the **if** statement and **while** statement of a proscheme allow undecidable conditions. Hence the proscheme is not always computable.

One of the major objectives of this book is to study the properties of proschemes. In the following sections, we shall introduce a few sequences of formal theories that can be generated by proschemes. Like the Lindenbaum sequence, these sequences are monotone sequences. They play key roles in the proofs of important results of mathematical logic.

6.4 Resolvent sequences

Robinson [1965] devised the resolution method when he was studying automated proof of mathematical theorems by computers. In this section, we discuss the resolution method, which takes conjunctive normal forms as its objects. A conjunctive normal form is a conjunction of finitely many sub-sentences, each of which is a disjunction of finitely many literals. Each literal is either a predicate symbol or the negation of a predicate symbol. In this section we only consider those predicates with no variables. For instance,

$$(P_1 \vee P_2) \wedge (\neg P_2 \vee P_3) \wedge (P_1 \vee P_2 \vee \neg P_3)$$

is a conjunctive normal form. $P_1 \vee P_2$, $\neg P_2 \vee P_3$ and $P_1 \vee P_2 \vee \neg P_3$ are its sub-sentences. If we use

$$Q_1, Q_2, \ldots, Q_n$$

to denote sub-sentences, then a conjunctive normal form can also be written as

$$C = \{Q_1, Q_2, \ldots, Q_n\}.$$

The comma "," in the above expression denotes the logical connective "$\wedge$". Hence a conjunctive normal form can be treated as a set of sub-sentences. The resolution method determines whether a conjunctive normal form is satisfiable and it is defined by resolvent relations.

Definition 6.4 (Resolvent relation). Suppose that $C = \{Q_1, Q_2, \ldots, Q_n\}$ is a conjunctive normal form. We say that the sub-sentences Q_i and Q_j have a *resolvent relation* if there exists a predicate L and formulas Q_i^1 and Q_j^1 such that

$$Q_i = L \vee Q_i^1, \quad Q_j = \neg L \vee Q_j^1.$$

We define $Q := Q_i^1 \vee Q_j^1$ as the *resolvent* of Q_i and Q_j and denote it as

$$Q_i,\ Q_j \rightarrowtail Q,$$

which reads as Q_i and Q_j resolve into Q.

In particular, let $\square$ denote the empty sentence. Then the empty sentence rule

$$Q,\ \neg Q \rightarrowtail \square$$

holds. This rule shows that the conjunctive normal form $Q \wedge \neg Q$ is always false.

The semantics of resolvent relations is given in the following lemma.

Lemma 6.4 (Resolvent relation). *Suppose that $C = \{Q_1, Q_2, \ldots, Q_n\}$ is a conjunctive normal form and there exist i and j such that $Q_i, Q_j \rightarrowtail Q$ holds. Then C is satisfiable if and only if $C \wedge Q$ is satisfiable.*

Proof. Since we only discuss predicates without variables, we only need to consider the truth values of the predicate symbols. Let I denote the interpretation of the predicate symbols.

Sufficiency: If the interpretation I makes $C \wedge Q$ satisfiable, then it makes C satisfiable.

Necessity: Let Q_i and Q_j be as in Definition 6.4. It suffices to prove that if I makes $Q_i \wedge Q_j$ true, then I makes Q true. That is to say, if I makes $(L \vee Q_i^1) \wedge (\neg L \vee Q_j^1)$ true, then I makes $Q_i^1 \vee Q_j^1$ true. Hence we only need to consider the following two cases:

(1) I makes L true. In this case in order for I to make $(L \vee Q_i^1) \wedge (\neg L \vee Q_j^1)$ true, I has to make Q_j^1 true. Thus I makes $Q_i^1 \vee Q_j^1$ true, i.e., I makes Q true.

(2) I makes L false. In this case in order for I to make $(L \vee Q_i^1) \wedge (\neg L \vee Q_j^1)$ true, I has to make Q_i^1 true. Thus I makes $Q_i^1 \vee Q_j^1$ true, i.e., I makes Q true. □

For a given conjunctive normal form $C = \{Q_1, Q_2, \ldots, Q_n\}$, we refer to the procedure of defining the resolvents in a recursive way as the *resolution procedure* of C. More specifically,

$$\begin{aligned} Res^0(C) &:= C, \\ Res^1(C) &:= \{Q \mid Q_i, Q_j \in C \text{ and } Q_i, Q_j \rightarrowtail Q\} \cup C, \\ &\vdots \\ Res^{n+1}(C) &:= Res(Res^n(C)), \\ &\vdots \end{aligned}$$

By definition, $Res^n(C) \subseteq Res^{n+1}(C)$ holds for every n. Since C only contains a finite number of predicate symbols, this procedure terminates after a finite number of operations. That is, there exists an m such that $Res^m(C) = Res^{m+1}(C)$. The resolution procedure generates a **finite** sequence of formal theories as follows:

$$Res^0(C), Res^1(C), \ldots, Res^m(C).$$

The resolvent closure of C is the limit of this sequence:

$$\{Res^n(C)\}^* := \bigcup_{n=0}^{\infty} Res^n(C) = Res^m(C).$$

We obtain a new form $C \wedge Q$ when we apply the resolution inference rule to a given conjunctive normal form C. According to Lemma 6.4, if $C \wedge Q$ is not satisfiable, then neither is C. Hence, if after a finite number of resolution steps, an empty sentence appears in $C \wedge Q$, then C is not satisfiable. In this way we have proved the following theorem.

Theorem 6.1. *Suppose that* $C = \{Q_1, Q_2, \ldots, Q_n\}$ *is a conjunctive normal form. If*

$$\text{there exists an } m > 0 \text{ such that } \square \in Res^m(C),$$

then C is not satisfiable.

The resolution procedure can be described by a proscheme. For convenience we define the following:

1. Let `CNF` be a data type representing conjunctive normal forms.
2. Let `Resolvent`$(\Gamma : \texttt{CNF}; Res : \texttt{CNF})$ be a P-procedure. It returns $\{Q \mid Q_i,\ Q_j \rightarrowtail Q$ and $Q_i, Q_j \in \Gamma\}$.

The input of the proscheme `Resolution` is a conjunctive normal form and the output is a resolvent closure that is also a conjunctive normal form.

proscheme `Resolution` (C : `CNF`; **var** Res : `CNF`)
begin
  Res' : `CNF`;
  $Res' := C$;
  $Res := \varnothing$;
  while not $Res \doteq Res'$ **do**
    $Res := Res'$;
    `Resolvent`(Res, Res');
    $Res' := Res \cup Res'$;
    print Res
end

The above resolution proscheme has two characteristics. First, the procedure is decidable. In fact, although a conjunctive normal form might contain tens of thousands of sub-sentences, the procedure to compute the resolvent halts since C is finite. Secondly, the output sequence is a finite monotone sequence whose limit is computable and is just the last element of the sequence. In the case of this example, the proscheme `Resolution` is a P-procedure.

6.5 Default expansion sequences

Default is a routine mechanism in programming languages. For instance, in the declarations of procedures and functions, if no initial value is assigned to a variable of integer type, then we prescribe that the initial value of this variable is 0. In this case we say that the *default value* of this variable is 0. Another example is that logical programming languages usually prescribe that if a knowledge base does not define a predicate P, then by default, we prescribe that $\neg P$ is true.

In this way we can ensure that the truth value of every predicate can be found in the knowledge base. This is the *closed world assumption* in logical programming. In first-order languages, the role of **default** can be described as follows: for a formal theory Γ, if $\Gamma \vdash \neg B$ is unprovable, then by default we assume that $\Gamma \vdash B$ is provable.

In 1980, Reiter generalized the concept of **default** and gave it a formal definition which he applied to non-monotonic reasoning [Reiter, 1980]. His default rules have the following form:

$$\frac{A : MB(x)}{B(x)}.$$

Here the formula A is called the *prerequisite* of the default rule, M the *default operator*, $B(x)$ in the numerator the *default premise*, and $B(x)$ in the denominator the *default conclusion*. The meaning of this default rule is: if A holds and $A \vdash \neg B(x)$ is unprovable, then $B(x)$ holds by default.

In this section we consider a simple case of default reasoning, that is, *normal default reasoning* introduced by Reiter. Its definition is as follows.

Definition 6.5 (Set of normal default rules). (1) Let A and B be sentences of $\mathscr{L}$. We call

$$\frac{A : MB}{B}$$

a *normal default rule*.

(2) Let Γ be a formal theory. If $\Gamma \vdash A$ is provable and $\Gamma \vdash \neg B$ is unprovable, then we call B a *default conclusion* of Γ with respect to $\dfrac{A : MB}{B}$.

(3) Δ is a countable (including finite) set of default rules:

$$\Delta = \left\{ \frac{A_1 : MB_1}{B_1}, \frac{A_2 : MB_2}{B_2}, \cdots, \frac{A_i : MB_i}{B_i}, \cdots \right\}.$$

$D(\Delta)$ is the set consisting of all the default conclusions of Δ.

Item (2) in the above definition shows that each default rule in Δ is meaningful only in the context of a given formal theory Γ. If Γ deduces B by default, then it can be understood as B being regarded as a formal consequence of Γ by default. Reiter also introduced the following general concept of default expansion, which was defined in two steps.

Definition 6.6 (Default operator). Let Γ be a formal theory, Δ be a set of default rules, and $\mathbf{F}$ be a map from a formula set to another formula set. If there exists a consistent formula set Λ such that $\mathbf{F}$ satisfies the following three properties:

(1) $\Gamma \subseteq \mathbf{F}(\Lambda)$,

(2) $\mathbf{F}(\Lambda) = Th(\mathbf{F}(\Lambda))$,

(3) if $\dfrac{A_n : MB_n}{B_n} \in \Delta$ with $A_n \in \mathbf{F}(\Lambda)$ and $\neg B_n \notin \Lambda$, then $B_n \in \mathbf{F}(\Lambda)$,

then we call $\mathbf{F}$ the *default operator* of Δ with respect to Λ.

Definition 6.7 (Default expansion). Let Γ be a formal theory, Δ be a set of default rules, and Λ be a consistent formula set. Suppose that $\mathbf{F}$ is the default operator of Δ with respect to Λ and $\mathbf{F}(\Lambda)$ is a consistent formula set. If Λ is the minimal fixed point of the equation

$$\mathbf{F}(\Lambda) = \Lambda,$$

then we call Λ a *default expansion* of Γ with respect to Δ.

In the above definition:

Property (1) of the default operator $\mathbf{F}$ shows that the default expansion contains the formal theory Γ.

Property (2) shows that the default expansion is a closure of the theory with respect to a formal inference such as $\mathbf{G}$.

Property (3) shows that the default expansion is a closure with respect to default inferences.

The difficulty in finding a solution for Λ has been greatly increased by the conditions $A_n \in \mathbf{F}(\Lambda)$ and $\neg B_n \notin \Lambda$ in (3) of Definition 6.6 plus the requirement of Λ being a minimal fixed point of the default operator $\mathbf{F}$ in Definition 6.7. In fact, the construction of the default expansion Λ is rather difficult for the set composed of generic default rules given in the beginning of this section.

Nevertheless, for a normal default set, we can generate a monotonic sequence of formal theories by defining a proscheme similar to `Lindenbaum` and prove that its limit is a default expansion.

Specifically, we have the following.

Definition 6.8 (Normal default expansion sequence). For any given formal theory Γ and set Δ of normal default rules, the *default expansion sequence* is defined recursively as follows.

(1) $\Xi_1 := \Gamma$.

(2) The default rules in Δ are examined one by one in sequence. For the rule $\dfrac{A_n : MB_n}{B_n}$,

$$\Xi_{n+1} := \begin{cases} \Xi_n \cup \{B_n\}, & \text{if } \Xi_n \vdash A_n, \Xi_n \nvdash \neg B_n, \\ \Xi_n, & \text{otherwise.} \end{cases}$$

It is not difficult to see that the above definition can be given by a proscheme:

proscheme `Default`(Γ: `theory`; Δ: `normal default rule set`)
begin
  Ξ, Ξ': `theory`;
  $\Xi := \Gamma$;
  $n := 1$;

print Ξ;
while $\dfrac{A_n : MB_n}{B_n} \in \Delta$ **do**
if $\Xi \vdash A_n$ **and** $\Xi \nvdash \neg B_n$ **then**
$\Xi' := \Xi \cup \{B_n\}$
else
$\Xi' := \Xi$;
$\Xi := \Xi'$;
$n := n+1$;
print Ξ
end

The inputs of the proscheme are Γ and the set Δ of normal default rules, whereas the output is the sequence of formal theories

$$\Xi_1, \Xi_2, \ldots, \Xi_n, \ldots.$$

The above sequence is an increasing sequence of formal theories. The following lemma shows that the limit of the sequence

$$Th(\Xi_1), Th(\Xi_2), \ldots, Th(\Xi_n) \ldots$$

is exactly the default expansion we are seeking.

Lemma 6.5. *For any given formal theory* Γ *and set* Δ *of normal default rules, if* $\{\Xi_n\}$ *is a normal default expansion sequence of* Γ *with respect to* Δ*, then the formula set*

$$\Lambda = \lim_{n \to \infty} Th(\Xi_n)$$

is a default expansion of Γ *with respect to* Δ*.*

Proof. It is not difficult to prove that the formal theory-closure operator Th is the default operator $\mathbf{F}$ in Definition 6.6, and Λ is the fixed point of $\mathbf{F}$ in Definition 6.7. Since we have $A_n \in Th(\Xi_n)$ in Definition 6.8 and noticing that $\{\Xi_n\}$ is an increasing sequence, $A_n \in \mathbf{F}(\Lambda)$ as required by Definition 6.6 holds. Further, both $\neg B_n \notin \Lambda$ and $B_n \in \mathbf{F}(\Lambda)$ hold. □

6.6 Forcing sequences

In this section we introduce the concepts of forcing relation, generic set and forcing sequence. We also prove that a forcing sequence is an increasing sequence of formal theories.

Let $\mathscr{L}$ be a first-order language with Γ being a formal theory of $\mathscr{L}$. Also suppose that $\mathscr{C}$ is a countable set of new constant symbols that are not contained in $\mathscr{L}$. Let $\mathscr{L}_{\mathscr{C}}$ be the first-order language generated from $\mathscr{L}$ by adding $\mathscr{C}$ to its set of constant symbols.

Definition 6.9 (T-condition). Let Γ be a formal theory of $\mathscr{L}$ with Δ being a set consisting of finitely many literals in $\mathscr{L}_{\mathscr{C}}$, i.e., a set consisting of finitely many atomic formulas and negations of atomic formulas. If $\Gamma \cup \Delta$ is consistent, then we say that Δ is a *T-condition* of Γ.

For a sentence A in $\mathscr{L}_{\mathscr{C}}$, we can define the forcing relation between A and the T-condition Δ of Γ using the following rules on logical connective symbols and quantifier symbols. The forcing relation is read as Δ forces A and denoted as $\Delta \Vdash_{\Gamma} A$. Without causing confusion, we can omit the subscript Γ and denote it as $\Delta \Vdash A$.

Definition 6.10 (Forcing relation). Let Δ be a T-condition of Γ and A be a sentence in $\mathscr{L}_{\mathscr{C}}$. $\Delta \Vdash A$ is defined by the following rules:

(1) $\Delta_1, A, \Delta_2 \Vdash A$;

(2) $$\frac{\text{there does not exist any T-condition } \Sigma \supseteq \Delta \text{ of } \Gamma \text{ such that } \Sigma \Vdash B \text{ holds}}{\Delta \Vdash \neg B};$$

(3) $$\frac{\Delta \Vdash B}{\Delta \Vdash B \vee C}, \qquad \frac{\Delta \Vdash C}{\Delta \Vdash B \vee C};$$

(4) $$\frac{\Delta \Vdash B \quad \Delta \Vdash C}{\Delta \Vdash B \wedge C};$$

(5) $\dfrac{\Delta \Vdash B(c)}{\Delta \Vdash \exists x B(x)}$, where c is a constant in $\mathscr{C}$;

(6) $\dfrac{\Delta \Vdash B(y)}{\Delta \Vdash \forall x B(x)}$, where y is a new variable that can only be substituted by constants in $\mathscr{C}$.

The definition of forcing requires that, if the forcing relations in the numerators hold, then those in the denominators hold. An equivalent definition to rule (2) in Definition 6.10 is: if $\Delta \Vdash \neg B$ does not hold then there exists a T-condition $\Sigma \supseteq \Delta$ of Γ such that $\Sigma \Vdash B$ holds.

We can see from the definition that a forcing relation is also a logical inference relation. It is different from formal inference systems such as **G** only in the inference rule (2) for the symbol $\neg$.

For the other logical symbols $\wedge$, $\vee$ and the quantifier symbols $\forall$ and $\exists$, the forcing inference rules bear the same form as the inference rules of first-order languages.

Also, compared with the **G** system, there are no left rules in the forcing inference system. The reason is that the T-condition Δ only contains atomic sentences and negations of atomic sentences.

Lemma 6.6. *If $\Delta \Vdash_{\Gamma} A$ holds, then A and Γ are consistent.*

Proof. The conclusion can be proved by structural induction. □

Lemma 6.7. *Suppose that Δ and Σ are T-conditions of Γ. If both $\Delta \subseteq \Sigma$ and $\Delta \Vdash A$ hold, then $\Sigma \Vdash A$ holds.*

Proof. We prove the lemma by structural induction.

(1) If A is an atomic sentence, then $\Delta \Vdash A$ indicates that $A \in \Delta \subseteq \Sigma$, and thus $\Sigma \Vdash A$.

(2) If A is $\neg B$, then $\Delta \Vdash A$ indicates that there does not exist any $\Delta' \supset \Delta$ such that $\Delta' \Vdash B$. Thus there is no $\Sigma' \supset \Sigma \supseteq \Delta$ such that $\Sigma' \Vdash B$. Hence $\Sigma \Vdash A$.

(3) If A is $B \vee C$, then from $\Delta \Vdash A$ we know that either $\Delta \Vdash B$ or $\Delta \Vdash C$. From the induction hypothesis we know that either $\Sigma \Vdash B$ or $\Sigma \Vdash C$. Thus $\Sigma \Vdash A$.

(4) If A is $\exists x B(x)$, then from $\Delta \Vdash A$ we know that there exists a c such that $\Delta \Vdash B(c)$. The induction hypothesis implies that $\Sigma \Vdash B(c)$ and thus $\Sigma \Vdash A$. □

Definition 6.11 (T-generic set). Suppose that Σ is a set consisting of atomic sentences or negations of atomic sentences in $\mathscr{L}_{\mathscr{C}}$. We say that Σ is the *T-generic set* of Γ if it satisfies the following two conditions.

(1) Every finite subset of Σ is a T-condition of Γ.

(2) For every sentence A in $\mathscr{L}_{\mathscr{C}}$, there exists a T-condition $\Delta \subseteq \Sigma$ of Γ such that either $\Delta \Vdash A$ or $\Delta \Vdash \neg A$.

Lemma 6.8. *If Σ is the T-generic set of Γ, then for every sentence A in $\mathscr{L}_{\mathscr{C}}$, either $\Sigma \Vdash A$ or $\Sigma \Vdash \neg A$ holds.*

The above lemma shows that a T-generic set possesses some sort of completeness with respect to forcing inference. Similar to the Lindenbaum lemma, we have the following results about T-generic sets.

Lemma 6.9. *For every T-condition Δ of Γ, there exists a T-generic set Σ of Γ such that $\Sigma \supseteq \Delta$.*

Proof. Since $\mathscr{C}$ is countable, we can list all the sentences in $\mathscr{L}_{\mathscr{C}}$ as follows:

$$A_1, A_2, A_3, \ldots, A_n, \ldots.$$

The T-condition of Γ that contains Δ is defined as follows.

(1) If $\Delta \Vdash \neg A_1$, then let $\Delta_1 = \Delta$. If $\Delta \Vdash \neg A_1$ does not hold, then by rule (2) in Definition 6.10, there must exist a T-condition Λ of Γ such that $\Lambda \supseteq \Delta$ and $\Lambda \Vdash A_1$. Let Λ be Δ_1. Hence

$$\text{either } \Delta_1 \Vdash A_1 \text{ or } \Delta_1 \Vdash \neg A_1 \text{ holds.}$$

(2) Similarly, if we already have $\Delta \subseteq \Delta_1 \subseteq \Delta_2 \subseteq \cdots \subseteq \Delta_n$ such that for $1 \leqslant k \leqslant n$,

$$\text{either } \Delta_k \Vdash A_k \text{ or } \Delta_k \Vdash \neg A_k \text{ holds,}$$

then we still invoke the method in (1) to define Δ_{n+1}. That is, if $\Delta_n \Vdash \neg A_{n+1}$, then let $\Delta_{n+1} = \Delta_n$; if $\Delta_n \Vdash \neg A_{n+1}$ does not hold, then by rule (2) in Definition 6.10, there must exist a T-condition Λ_n of Γ such that $\Lambda_n \supseteq \Delta_n$ and $\Lambda_n \Vdash A_{n+1}$. Let Δ_{n+1} be this Λ_n. Then

$$\text{either } \Delta_{n+1} \Vdash A_{n+1} \text{ or } \Delta_{n+1} \Vdash \neg A_{n+1} \text{ holds.}$$

From this we obtain an increasing sequence of T-conditions as follows:

$$\Delta \subseteq \Delta_1 \subseteq \Delta_2 \subseteq \Delta_3 \subseteq \cdots \subseteq \Delta_n \cdots .$$

We call this sequence a *forcing sequence*. We further let

$$\Sigma = \bigcup_{i=1}^{\infty} \Delta_i.$$

It is not difficult to prove that Σ is the T-generic set of Γ containing the T-condition Δ. □

We can see from the above that the forcing sequence is an important mechanism to expand the T-condition Δ of Γ into its T-generic set. The T-generic set Σ is the limit of the forcing sequence $\{\Delta_n\}$. As in this lemma and its proof, we can construct a model $\mathbf{M}_\Sigma$ using a similar method as we used in the definition of the Hintikka set. In this model, for every sentence A in $\mathscr{L}_\mathscr{C}$, $\mathbf{M}_\Sigma \vDash A$ if and only if $\Sigma \Vdash A$. This is the theorem of models for generic sets [Wang, 1987].

Let Γ denote the Zermelo-Fraenkel axiom system of set theory. Let Δ denote the T-conditions composed of atomic propositions or negations of atomic propositions that do not satisfy Cantor's continuum hypothesis. Using the above forcing method, we can define the generic set Σ that contains Δ and is consistent with Γ, from which the generic set model $\mathbf{M}_\Sigma$ is generated. In this model the Zermelo-Fraenkel axioms of set theory are true, whereas Cantor's continuum hypothesis is false. Thus one can prove the independence of the Zermelo-Fraenkel axiom system of set theory and Cantor's continuum hypothesis [Cohen, 1966].

The problem of forcing sequence is that the definition of Λ_n is an existential definition. In this case, we find it difficult to design a proper proscheme that generates Λ_n.

6.7 Discussions on proschemes

We have introduced three basic concepts in this chapter. They are the proscheme, a sequence of formal theories and its limit. We believe that most readers will easily accept the sequence of formal theories and its limit, but in the case of a proscheme, readers might question its usefulness. This is because it is meant to be some kind of procedure and yet, when Γ contains the theory of arithmetic, neither the condition $\Gamma \vdash A$ nor **consistent**(Γ, A) is decidable. As a result, these two conditions cannot be generically implemented on computers and we may well ask what is the use of a procedure that cannot be implemented.

We have three answers that justify the introduction of proschemes:

1. In mathematical logic, the proscheme formulates an important and often used technique for proving theorems. For instance, the idea of proscheme and limit of a version sequence was used implicitly in the first part of this book to prove the completeness of the formal inference system **G**. The key step in that proof was, in effect, to invoke the `Lindenbaum` proscheme to construct the maximal consistent set. In this chapter we saw that, for the development of the concept of default inference in non-monotonic reasoning

and the analysis of resolution, the proscheme technique is indispensable. Also, sequences of theories and the limit of those sequences are central concepts in theory of forcing.

In Chapters 8 and 9, we shall discuss convergent non-monotonic version sequences. Such sequences are directly related to software development methods and inductive problems, whose generation would be impossible without the idea of proscheme. These examples indicate that the proscheme and the limit of a version sequence are useful methods for proving some worthwhile conclusions in mathematical logic.

2. If the conditions $\Gamma \vdash A$ and **consistent**(Γ, A) are decidable, then the proscheme becomes a halting Turing machine except that the input may be a infinite sequence. In this circumstance:

(i) All problems that are solvable by a Turing machine can be solved by a proscheme with a finite input sequence.

(ii) The problems that are implemented by a *real computational mechanism* in *Complexity and Real Computation* [Blum et al, 1997] could also be solved by a proscheme [Li, 2000, Li et al, 2001, Li and Ma, 2004].

(iii) However, even though the above conditions are decidable, there are problems that can not be resolved by a proscheme, i.e., the solution of this problem is not the limit of the output sequence of the proscheme with a recursively enumerable input sequence, such as the forcing problem.

3. Finally, when the conditions $\Gamma \vdash A$ and **consistent**(Γ, A) in the proscheme are undecidable, then the proscheme is not a halting procedure in the Turing sense. But this does not mean that we can never solve the problem in other ways.

On the contrary, this problem will motivate us to seek new methods and to invent new techniques; for example, model checking. Our analysis of the proscheme separates problems into two classes. One is solvable by the proscheme. The other can not be solved with this approach. This separation itself is meaningful. In any case, the concept of proscheme can enhance our understanding of the difficulty of problems.

Chapter 7

Revision Calculus

In scientific research, one tries, from a large body of knowledge, to extract the most fundamental propositions to use as an axiom system. All axiom systems developed in mathematics and science have evolved in many stages rather than being created all at once.

In the process of axiomatizing domain knowledge, each version of the axiom system is imperfect and new axioms, principles or laws may be proposed at any time. For instance, in the evolution of physics discussed in Chapter 6, Newton's three laws were proposed to augment Galileo's version. As another example, consider the process of software development. Each version of the software needs new functions to be added to meet the demands of the designers and the users.

We call these new axioms, laws, and functions *new conjectures*. When new conjectures are proposed, the current version of the axiom system must be extended and a new version is born.

On the other hand, an axiom system may contain axioms and laws that are inconsistent with the results of experiments. For instance, the Galilean transformation is inconsistent with the experimental results about the velocity of light in a moving coordinate system. In this case, we say that Galilean transformation meets a *refutation by facts*, or Galileo's version of physics is refuted by facts.

Similarly, no software designer can design and implement, at one go, functioning software that does not have any bugs. Bugs can always be found in software systems and this is why software systems always appear version by version.

If some logical consequence of an axiom system is inconsistent with the results of experiments, i.e., is refuted by facts, then the axiom system must be changed and one must abandon the axioms in the current version that contradict the results and retain the remaining axioms that are consistent with the facts.

The main purpose of this chapter is to demonstrate how to make, extend and revise axiom systems by adding new concepts to first-order languages and their models. This will establish a revision calculus for formal theories. This system is called **R**-*calculus*. It consists of four sets of rules, which are **R**-axioms, **R**-rules for logical connective symbols, **R**-rules for quantifier symbols, and **R**-cut rules. We will illustrate the usage of **R**-calculus with some typical examples and will prove the soundness, completeness, and reachability of **R**-calculus.

The concepts of necessary antecedents of logical consequences, new conjectures and new axioms, and refutation by facts and formal refutation are introduced in Sections 7.1, 7.2, and 7.3. **R**-calculus is introduced in Section 7.4 and several examples are discussed in Sections 7.5, 7.6 and 7.7. In Section 7.8 the concept of reachability is introduced

and it is proved that **R**-calculus is reachable. In Section 7.9 the concepts of soundness and completeness are introduced and it is proved that **R**-calculus is both sound and complete. In Section 7.10 **R**-calculus is extended for sets of inconsistent formulas and the basic theorem of testing is proved.

7.1 Necessary antecedents of formal consequences

In Chapter 3 we proved the compactness theorem, i.e., for any given formula set Γ and formula A, if $\Gamma \vdash A$ is provable, then there exists a finite formula set $\Delta \subseteq \Gamma$ such that $\Delta \vdash A$ is provable. In this section we introduce the concept of necessary antecedent of a formal consequence A with respect to Γ, which is essential for defining **R**-calculus.

Definition 7.1 (Necessary antecedents of the formal consequence). Suppose that Γ is a formula set, A is a formula and $\Gamma \vdash A$ is provable. We call a formula set Δ the *necessary antecedent set* of the formula A with respect to Γ if $\Delta \subseteq \Gamma$ is the minimal formula set for which $\Delta \vdash A$ holds. In other words, if $\Delta' \subset \Delta$, then $\Delta' \vdash A$ is unprovable. We say that B is a *necessary antecedent* of the formal consequence A, which is denoted as $B \mapsto_\Delta A$, if $B \in \Delta$.

If $\Gamma \vdash A$ is provable and its proof tree is $\mathcal{T}$, then the necessary antecedent set of A with respect to Γ is constructible. First of all, we need to introduce the concept of antecedent of A with respect to Γ.

Definition 7.2 (Antecedent set of a proof tree). Suppose that Γ is a formula set, A is a formula and $\Gamma \vdash A$ is provable. Also suppose that $\mathcal{T}$ is a proof tree of $\Gamma \vdash A$ with P, Q and R being formulas appearing in $\mathcal{T}$.

(1) If Γ' is a formula set and $\Gamma', P \vdash P$ is a leaf of the proof tree $\mathcal{T}$, then P on the left-hand side is the antecedent of P on the right-hand side of $\vdash$ with respect to $\mathcal{T}$.

(2) P is an antecedent of Q with respect to $\mathcal{T}$, if a node of the proof tree $\mathcal{T}$ is an instance of a right rule of the **G** system with Q appearing as $B \wedge C$, $B \vee C$, $B \to C$, $\neg B$, $\forall x B(x)$ or $\exists x B(x)$ in the denominator of the rule, i.e., as the principal formula of the rule (see Definition 3.8), and P appearing as B, C, $B[t/x]$ or $B[y/x]$ in the numerator of the rule, i.e., as the side formula of the rule.

(3) P is an antecedent of Q with respect to $\mathcal{T}$, if a node of the proof tree $\mathcal{T}$ is an instance of a left rule with P appearing as $B \wedge C$, $B \vee C$, $B \to C$, $\neg B$, $\forall x B(x)$ or $\exists x B(x)$ in the denominator of the rule, i.e., as the principal formula of the rule, and Q appearing as B, C, $B[t/x]$ or $B[y/x]$ in the numerator of the rule, i.e., as the side formula of the rule. The side formula B, C, $B[t/x]$ or $B[y/x]$ is the antecedent of the side formula on the right-hand side of $\vdash$ in the denominator of the rule with respect to $\mathcal{T}$.

(4) If P is an antecedent of Q with respect to the proof tree $\mathcal{T}$ and Q is an antecedent of R with respect to $\mathcal{T}$, then P is an antecedent of R with respect to $\mathcal{T}$.

Let $\mathcal{P}(\Gamma, A, \mathcal{T})$ denote the set consisting of all the antecedents of A with respect to the proof tree $\mathcal{T}$.

Example 7.1 ($\wedge$-R rule).

$$\frac{A,B \vdash A \qquad A,B \vdash B}{A,B \vdash A \wedge B}$$

By Definition 7.2 (2), the formulas A and B are antecedents of the formula $A \wedge B$ with respect to the above proof tree.

Example 7.2. Consider the sequent $C,A,\forall x(A \to B(x)) \vdash \exists x B(x)$ and let

$$\begin{aligned} S_1 :&\quad C,A^{*4},(\forall x(A \to B(x)))^{*2} \vdash A^{*3},B[t/x]^{*1},\exists x B(x),\\ S_2 :&\quad C,A,(\forall x(A \to B(x)))^{*2},B[t/x]^{*3} \vdash B[t/x]^{*1},\exists x B(x). \end{aligned}$$

The following is a proof tree $\mathcal{T}$ of the sequent:

$$\dfrac{\dfrac{\dfrac{S_1\ (4) \qquad S_2\ (5)}{C,A,(\forall x(A \to B(x)))^{*2},(A \to B[t/x])^{*2} \vdash B[t/x]^{*1},\exists x B(x)}\,(3)}{C,A,(\forall x(A \to B(x)))^{*2} \vdash B[t/x]^{*1},\exists x B(x)}\,(2)}{C,A,\forall x(A \to B(x)) \vdash \exists x B(x)}\,(1)$$

Node (1) of the proof tree is obtained by applying the $\exists$-R rule. According to Definition 7.2 (2), $B[t/x]^{*1}$ is an antecedent of $\exists x B(x)$. We use $*$ on the upper right corner of a formula to represent an antecedent. The following numeral 1 represents node (1) of the proof tree. Thus for node (1), $B[t/x]^{*1}$ is an antecedent of $\exists x B(x)$.

Node (2) is obtained by applying the $\forall$-L rule to $\forall x(A \to B(x))$ in the denominator. According to Definition 7.2 (3), $(\forall x(A \to B(x)))^{*2}$ on the left-hand side of $\vdash$ in the denominator is an antecedent of $(A \to B[t/x])^{*2}$ on the left-hand side of $\vdash$ in the numerator. $(A \to B[t/x])^{*2}$ is an antecedent of $B[t/x]^{*1}$.

Node (3) of the proof tree is obtained by applying the $\to$-L rule to $A \to B[t/x]$. According to Definition 7.2 (3), $(A \to B[t/x])^{*2}$ is an antecedent of A^{*3} on the right-hand side of $\vdash$ in the first sequent of the numerator. It is also an antecedent of $B[t/x]^{*3}$ on the left-hand side of $\vdash$ in the second sequent. A^{*3} and $B[t/x]^{*3}$ are antecedents of $B[t/x]^{*1}$ on the right-hand side of $\vdash$ in the denominator.

Node (4) of the proof tree is an instance of the axiom. A^{*4} on the left-hand side is an antecedent of A^{*3} on the right-hand side of $\vdash$.

Node (5) is an instance of the axiom as well. $B[t/x]^{*3}$ on the left-hand side is an antecedent of $B[t/x]^{*1}$ on the right-hand side of $\vdash$.

Thus the antecedent set of the formal consequence $\exists x B(x)$ of the sequent $C,A,\forall x(A \to B(x)) \vdash \exists x B(x)$ with respect to the proof tree $\mathcal{T}$ is

$$\{B[t/x],\forall x(A \to B(x)),A \to B[t/x],A\}.$$

Example 7.3 (Necessary antecedent). In Example 7.2, the necessary antecedent set of the formal consequence $\exists x B(x)$ of the sequent $C,A,\forall x(A \to B(x)) \vdash \exists x B(x)$ with respect to the proof tree $\mathcal{T}$ is

$$\{C,\ A,\ \forall x(A \to B(x))\} \cap \{B[t/x],\forall x(A \to B(x)),A \to B[t/x],A\},$$

that is,

$$\{A, \forall x(A \to B(x))\}.$$

We can see from the above two examples that if the sequent $\Gamma \vdash A$ is provable and $\mathcal{T}$ is its proof tree, then according to the antecedent set given by Definition 7.2, we can construct the necessary antecedent set of A with respect to $\mathcal{T}$.

Lemma 7.1. *Suppose that the sequent $\Gamma \vdash A$ is provable and $\mathcal{T}$ is its proof tree.*

(1) *The set $\mathcal{P}(\Gamma, A, \mathcal{T})$ is decidable.*

(2) *The formula set $\Gamma \cap \mathcal{P}(\Gamma, A, \mathcal{T})$ is a necessary antecedent set of the formula A with respect to Γ.*

Proof. We prove (1) first. According to Definition 7.2 we design a halting P-procedure, whose input is the proof tree $\mathcal{T}$ and whose output is a formula set, as follows. In accordance with Example 7.2, the procedure starts from the root of the proof tree and searches its branches layer by layer. When the proof tree branches, we search the sequents on the same layer from left to right.

The procedure continues until we reach the leaves of the tree. More specifically, we determine the antecedent set of each node by the first three items of Definition 7.2. Since a proof tree is finite, this search procedure terminates and hence we obtain the antecedent set $\mathcal{P}(\Gamma, A, \mathcal{T})$ of the formula A with respect to Γ and the proof tree $\mathcal{T}$.

Now we prove (2). The intersection of Γ and the antecedent set of the proof tree $\mathcal{T}$,

$$\Delta = \Gamma \cap \mathcal{P}(\Gamma, A, \mathcal{T}),$$

is the necessary antecedent set of A with respect to Γ and $\mathcal{T}$. According to the construction of $\mathcal{P}(\Gamma, A, \mathcal{T})$, we shall not be able to generate the proof tree $\mathcal{T}$ starting from $\Gamma \vdash A$ if any formula in Δ is deleted. Hence Δ is the minimal formula set, for which $\Delta \vdash A$. That is, Δ is the necessary antecedent set of A with respect to the sequent $\Gamma \vdash A$ and the proof tree $\mathcal{T}$. □

Example 7.4. Given the sequent $A, A \to B, B \to C \vdash C$, consider its proof tree $\mathcal{T}$ as follows:

$$\dfrac{\dfrac{A \vdash A^{*2}, B, C \quad A, B^{*2} \vdash B^{*1}, C}{A, (A \to B)^{*2} \vdash B^{*1}, C}\,(2) \qquad C^{*1}, A, A \to B \vdash C\,(3)}{A, A \to B, (B \to C)^{*1} \vdash C}\,(1)$$

Node (1) is an instance of the $\to$-L rule. According to Definition 7.2 (3), the antecedent set of C with respect to $\mathcal{T}$ is

$$\{(B \to C)^{*1}, B^{*1}, C^{*1}\}.$$

Node (2) is an instance of the $\to$-L rule as well. Similarly, the antecedent set of B^{*1} is

$$\{(A \to B)^{*2}, A^{*2}, B^{*2}\}.$$

According to Definition 7.2 (1), the antecedents of A^{*2}, B^{*1} and C are A, B^{*2} and C^{*1} respectively. As per Definition 7.2, the antecedent set of C with respect to $\mathcal{T}$ is

$$\{A \to B, B \to C, A, B, C\}.$$

According to Lemma 7.1, the necessary antecedent set of C with respect to $\mathcal{T}$ is

$$\{A, A \to B, B \to C\}.$$

These examples employ the method of Lemma 7.1 in constructing the necessary antecedent sets. Hence Lemma 7.1 can be regarded as a constructive definition of the concept of necessary antecedent, while Definition 7.1 is not constructive. It only specifies the properties of the necessary antecedent set.

Since the provability of $\Gamma \vdash A$ is the condition of Definition 7.1, the proof tree exists and thus the construction procedure of Lemma 7.1 can be performed.

Since a provable sequent may have several proof trees, the necessary antecedent sets of A do not have to be unique either. Nonetheless each necessary antecedent set must correspond to a proof tree. In this case the notation $B \mapsto_{\Delta} A$ in Definition 7.1 will be written as $B \mapsto_{\mathcal{T}} A$ with $\mathcal{T}$ being indispensable. However without causing confusion, for simplicity, we sometimes omit the subscript $\mathcal{T}$ in $B \mapsto_{\mathcal{T}} A$.

7.2 New conjectures and new axioms

In this section we study new conjectures and new axioms of a formal theory Γ. A new conjecture is a concept related to the models of Γ, whereas a new axiom refers to the formal proof of Γ. The formal theory Γ needs to be expanded whenever a new conjecture is proposed.

Definition 7.3 (New conjecture). We call a sentence A a *new conjecture* of a formal theory Γ if there exist two models $\mathbf{M}$ and $\mathbf{M}'$ such that

$$\text{both } \mathbf{M} \models \Gamma,\ \mathbf{M} \models A \text{ and } \mathbf{M}' \models \Gamma,\ \mathbf{M}' \models \neg A \text{ hold.}$$

Definition 7.4 (New axiom). We call a sentence A a *new axiom* of a formal theory Γ if neither $\Gamma \vdash A$ nor $\Gamma \vdash \neg A$ is provable.

If a sentence A is a new axiom of Γ, we say that the sentence A and the formal theory Γ are logically independent. We can directly deduce the following lemma by the soundness and completeness of the **G** system.

Lemma 7.2. *A sentence A is a new axiom of Γ if and only if A is a new conjecture of Γ.*

If we add new axioms, we can expand the formal theory Γ in the following way.

Definition 7.5 (N-expansion). Suppose that A is a new axiom of a formal theory Γ. The set $\Gamma \cup \{A\}$ is called an **N**-*expansion* of Γ with respect to A.

If a sentence A is a new axiom of a formal theory Γ, then A must be a description in first-order language of a new proposed conjecture. In fact, neither A nor $\neg A$ is a formal consequence of Γ according to Definition 7.4. Thus the new axiom A is not a result of formal inferences on Γ.

As in Lemma 7.2, it is only through constructing models that we can verify whether A is a new axiom of Γ. Whenever we add a new axiom A to Γ, we make an **N**-expansion of Γ, which is a new version of the formal theory containing A.

7.3 Refutation by facts and maximal contraction

In this section we discuss the concept of refutation by facts of a formal theory Γ. We also discuss its corresponding concept in proof theory, i.e., the formal refutation of Γ. Whenever a formal theory is refuted by facts, one needs to revise it and the result of the revision is called a *maximal contraction*.

Definition 7.6 (Model of refutation by facts). Suppose that Γ is a formal theory and A is a sentence such that $\Gamma \models \neg A$ holds. If there exists a model $\mathbf{M}$ such that $\mathbf{M} \models A$ holds, then we say that $\mathbf{M}$ is a *model of refutation by facts* of Γ with respect to A. We also say that Γ is *refuted* by the model $\mathbf{M}$ with respect to A. Let

$$\Gamma_{\mathbf{M}(A)} = \{B \mid B \in \Gamma, \mathbf{M} \models B, \mathbf{M} \models A\}.$$

We call $\mathbf{M}$ an *ideal model of refutation by facts* of Γ with respect to A, or *ideal refutation model* for short, if $\Gamma_{\mathbf{M}(A)}$ is maximal. This means that there is not another model $\mathbf{M}'$ of refutation by facts with respect to A such that $\Gamma_{\mathbf{M}(A)} \subset \Gamma_{\mathbf{M}'(A)}$.

Note that $\mathbf{M}$ is not a model of Γ, it is a counterexample of Γ. $\Gamma_{\mathbf{M}(A)}$ is the subset of Γ whose elements are consistent with A. This allows $\mathbf{M}$ to be a model of $\Gamma_{\mathbf{M}(A)}$.

From now on, all the models of refutation by facts discussed in this book are ideal unless specified otherwise. When we say that A is a refutation by facts of Γ, we mean that $\Gamma \models \neg A$ and there exists an ideal model $\mathbf{M}$ of refutation by facts such that $\mathbf{M} \models A$.

Since the ideal model $\mathbf{M}$ of refutation by facts of Γ with respect to A is not unique, there exists a set $\Gamma_{\mathbf{M}(A)}$ for each model $\mathbf{M}$ of refutation by facts. Such sets constitute a class

$$\mathcal{R}(\Gamma, A) = \{\Gamma_{\overline{\mathbf{M}}(A)} \mid \overline{\mathbf{M}} \text{ is the ideal model of refutation by facts of } \Gamma \text{ with respect to } A\}.$$

In summary, if a formal theory Γ is refuted by A, it shows that some axioms in Γ are refuted by facts. The refutations are in the form of evidence that is a counterexample of Γ. They can be described by a model $\mathbf{M}$ in the meta-language such that A holds in this model. $\Gamma_{\mathbf{M}(A)}$ is a subset of Γ consisting of all the sentences that hold in $\mathbf{M}$, i.e., all the sentences that are not refuted by the fact A.

Since A has a model $\mathbf{M}$ for which there is definite evidence, it has to be accepted. In order to revise Γ we must delete those sentences in Γ that do not hold in $\mathbf{M}$, and retain the sentences in $\Gamma_{\mathbf{M}(A)}$.

The concept in proof theory that corresponds to refutation by facts is termed *formal refutation* in first-order languages.

Definition 7.7 (Formal refutation and maximal contraction). If $\Gamma \vdash \neg A$ holds and $\neg A$ is not a valid sentence, then we call A a *formal refutation* of Γ. We call a formal theory Λ the *maximal contraction* of Γ with respect to the formal refutation A if Λ is the maximal subset of Γ that is consistent with A. Let $\mathcal{C}(\Gamma, A)$ denote the set of all the maximal contractions of Γ with respect to A.

The following lemma shows that the formal refutation and refutation by facts are two corresponding concepts.

Lemma 7.3. *Let Γ be a formal theory and A be a sentence. A is a formal refutation of Γ if and only if $\Gamma \models \neg A$ and there exists an ideal model $\mathbf{M}$ of refutation by facts such that $\mathbf{M} \models A$.*

Proof. The conclusion of the lemma follows directly from the soundness and completeness of the **G** system in Chapter 3. □

Theorem 7.1.

$$\mathcal{C}(\Gamma, A) = \mathcal{R}(\Gamma, A).$$

Proof. We first prove that $\mathcal{C}(\Gamma, A) \subseteq \mathcal{R}(\Gamma, A)$. Suppose that $\Lambda \in \mathcal{C}(\Gamma, A)$. Since Λ is consistent with A, there exists an $\mathbf{M}'$ such that both $\mathbf{M}' \models \Lambda$ and $\mathbf{M}' \models A$. Hence $\mathbf{M}'$ is a model of refutation by facts with respect to A. $\mathbf{M}'$ has to be maximal. Otherwise, there would exist another model $\mathbf{M}''$ such that $\mathbf{M}'' \models A$ and $\Gamma_{\mathbf{M}''(A)} \supset \Gamma_{\mathbf{M}'(A)} \supseteq \Lambda$, where $\supset$ denotes the proper inclusion. However, this is impossible from the definition of Λ.

Now we prove that $\mathcal{R}(\Gamma, A) \subseteq \mathcal{C}(\Gamma, A)$. Suppose that $\Lambda \in \mathcal{R}(\Gamma, A)$. By definition, there exists an ideal model $\overline{\mathbf{M}}$ of refutation by facts of Γ with respect to A such that $\Lambda = \Gamma_{\overline{\mathbf{M}}(A)}$. Assume that there exists a Λ' consistent with A such that $\Lambda \subset \Lambda' \subseteq \Gamma$. In this case there exists an $\overline{\mathbf{M}'}$ such that both $\overline{\mathbf{M}'} \models A$ and $\overline{\mathbf{M}'} \models \Lambda'$. Thus $\Gamma_{\overline{\mathbf{M}}(A)} \subset \Gamma_{\overline{\mathbf{M}'}(A)}$, which contradicts the assumption that $\overline{\mathbf{M}}$ is an ideal model of refutation by facts. Hence $\Lambda \in \mathcal{C}(\Gamma, A)$. □

Example 7.5 (Maximal contraction). Let the formal theory Γ be the following set of sentences:

$$\{A, A \to B, B \to C, E \to F\}.$$

It is not difficult to prove that $\Gamma \vdash C$. Suppose that Γ is refuted by a model $\mathbf{M}$ with respect to $\neg C$ such that $\mathbf{M} \models \neg C$. In this case $\neg C$ is a formal refutation of Γ that has to be accepted. By Definition 7.7, there are three maximal subsets of Γ that are consistent with $\neg C$:

$$\{A, A \to B, E \to F\}, \quad \{A, B \to C, E \to F\}, \quad \{A \to B, B \to C, E \to F\}.$$

They are all maximal contractions of Γ with respect to $\neg C$. This example shows that the maximal contraction of a formal theory with respect to its formal refutation is **not** unique.

As we can see from Definition 7.6, it is only when we interpret a formal theory that we may find a refutation by facts. A refutation by facts is defined in the meta-language environment of the formal theory and is a proposition that holds in some model. However, the motivation for this model is different from the ideas of Chapter 2. There we focused on models that satisfied a theory, here we concentrate on models of refutation by facts, which are *not* models of the theory and which are models of evidence-based counterexamples to the theory.

In the first part of this book we studied the properties of a *fixed* formal theory in a first-order language. However, when we view a theory as one stage in an ongoing, creative, axiomatization process, then we can expect to find counterexamples to its conclusions and must learn how to revise the theory, in order to best bring it to consistency with the new facts. Ideal models of refutation by facts and formal refutations are the mathematical concepts needed to perform these revisions.

We see in this chapter that the revision of a refuted theory Γ has two steps. We first contract Γ to its maximal contraction Γ' with respect to its formal refutation A. Then we make an **N**-expansion of Γ', adding A as a new axiom, so as to obtain another version, Γ''.

The concept of maximal contraction follows Occam's Razor. This is a guiding principle for the creation of theories, which requires that *"entities are not to be multiplied beyond necessity"* [Flew, 1979]. When we are talking about revising a theory to deal with a refutation, this may be re-stated as *"make as few changes as possible to the theory to make it consistent with the counterexample"*. This is the motivation behind the maximal contraction $\Gamma_{\mathbf{M}(A)}$, since it contains the maximal subset of Γ consistent with A, i.e., Γ has been changed by the least amount necessary.

In summary, Sections 7.2 and 7.3 show that, for a given formal theory Γ, neither its new conjectures nor its refutations by facts are its logical consequences. They are defined in the meta-language environment of the formal theory. They are proposed when interpretations of the theory are made and reflect the mutual interactions between the formal theory and its models which describe reality. It is the scientists who decide whether to accept a formal theory Γ. Their decision depends on whether the formal consequences of Γ are refuted by facts. It is in this sense that the author once referred to the theoretical framework of revision and refutation by facts as *open logic* [Li, 1992].

7.4 R-calculus

We know that if Γ is a formal theory and $\Gamma \vdash \neg A$ is provable, then the maximal contraction Λ of Γ with respect to the formal refutation A is the maximal subset of Γ that is consistent with A. In this section we discuss, for given Γ and A, how to find all the maximal contractions.

The question is: for a given formal theory Γ and its formal refutation A, can we design a set of calculus rules, that can be used to deduce all the maximal contractions of Γ with respect to A? The answer is yes. The set of calculus rules is called **R**-*calculus*.

Definition 7.8 (R-refutation). Let Γ be a formal theory and Δ be a formal theory consisting of finitely many atomic formulas or negations of atomic formulas. If Δ and Γ are

inconsistent, then we call Δ an **R**-*refutation* of Γ.

We call the ideal model of refutation by facts of an **R**-refutation the *model of* **R**-*refutation* and define it more specifically as follows.

Definition 7.9 (Model of R-refutation). Let $\Delta = \{A_1, \ldots, A_n\}$ be a formal theory consisting of finitely many atomic formulas or negations of atomic formulas such that $\Gamma \models \neg A_1 \vee \cdots \vee \neg A_n$. If there is a model $\mathbf{M}$ such that $\mathbf{M} \models \Delta$, then we call $\mathbf{M}$ the *model of refutation by facts* of Γ with respect to Δ, or say that Γ is *refuted* by $\mathbf{M}$ with respect to Δ. Let

$$\Gamma_{\mathbf{M}(\Delta)} = \{B \mid B \in \Gamma, \mathbf{M} \models B, \mathbf{M} \models \Delta\}.$$

We call $\mathbf{M}$ a *model of* **R**-*refutation* of Γ with respect to Δ if $\Gamma_{\mathbf{M}(\Delta)}$ is maximal, that is, there does not exist another model $\mathbf{M}'$ of refutation by facts with respect to Δ such that $\Gamma_{\mathbf{M}(\Delta)} \subset \Gamma_{\mathbf{M}'(\Delta)}$.

Another form of Δ is $A_1 \wedge \cdots \wedge A_n$. Thus the model of **R**-refutation of Γ with respect to Δ is just the ideal model of refutation by facts of Γ with respect to $A_1 \wedge \cdots \wedge A_n$.

The motivation for defining Δ as a finite set of atomic formulas and negations of atomic formulas is the following: we introduce Δ to describe the facts, which are propositions supported by experiments and observations. In modern days, the information acquired from these experiments is in the form of digital data, which can be represented by constants, functions or sets of data with common attributes. Therefore, the factual propositions extracted from these functions and sets can be represented either by equations or predicates or their negation. This implies that Δ is a finite set of atomic formulas and negations of atomic formulas, which in general, are consistent with each other.

Definition 7.10 (R-contraction). We call a formal theory Λ an **R**-*contraction* of a formal theory Γ with respect to an **R**-refutation Δ if Λ is the maximal subset of Γ that is consistent with Δ.

Definition 7.11 (R-configuration). Let Γ be a finite formula set and Δ a finite formal theory consisting of atomic formulas or negations of atomic formulas. We call $\Delta \mid \Gamma$ an **R**-*configuration*.

If Γ is a formal theory with Δ being the **R**-refutation of Γ, then we call $\Delta \mid \Gamma$ an *inconsistent* **R**-*configuration*.

For convenience, Δ and Γ on either side of "|" in the **R**-configuration can be regarded as either sets or sequences of sentences. They can also be written in the form of A, B, Δ and A, B, Γ.

Lemma 7.4. *If* $\Delta = \{A_1, \ldots, A_n\}$ *and* $\Delta \mid \Gamma$ *is an inconsistent* **R**-*configuration, then*

$$\Gamma \vdash \neg(A_1 \wedge \cdots \wedge A_n)$$

is provable.

Proof. As in (2) of Lemma 3.7, if Δ and Γ are inconsistent, then $\Gamma, \Delta \vdash \neg(A_1 \wedge \cdots \wedge A_n)$ is provable. By applying the $\wedge$-*L* rule and $\neg$-*R* rule on the sentences in Δ we can see that $\Gamma \vdash \neg(A_1 \wedge \cdots \wedge A_n)$ is provable. □

For an inconsistent **R**-configuration $\Delta \mid \Gamma$, Δ can be regarded as a formula $A_1 \wedge \cdots \wedge A_n$. Lemma 7.4 indicates that $\Gamma \vdash \neg(A_1 \wedge \cdots \wedge A_n)$ is provable. According to the definition of the formal refutation, $A_1 \wedge \cdots \wedge A_n$ is a formal refutation of Γ. Hence an **R**-refutation can be regarded as a formal refutation composed only of atomic sentences, negations of atomic sentences, and $\wedge$. The **R**-contraction of Γ with respect to the **R**-refutation Δ is just the maximal contraction of Γ with respect to the formal refutation $A_1 \wedge \cdots \wedge A_n$.

We shall use the following notation in the rest of this chapter.

Definition 7.12 (R-transition).

$$\Delta \mid \Gamma \Longrightarrow \Delta' \mid \Gamma'$$

is called an **R**-*transition*. It transforms the **R**-configuration $\Delta \mid \Gamma$ into the **R**-configuration $\Delta' \mid \Gamma'$.

In particular, the **R**-transition

$$\Delta \mid A, \Gamma \Longrightarrow \Delta \mid \Gamma$$

denotes the transform of the **R**-configuration $\Delta \mid A, \Gamma$ into $\Delta \mid \Gamma$. As a result, A in the sentence sequence A, Γ on the right-hand side of "$\mid$" is deleted.

Hereafter we introduce **R**-calculus. Informally, we construct **R**-calculus as follows:

1. A refutation by facts is the basis for the revision of a formal theory and it has to be accepted. For an **R**-configuration, Δ is the **R**-refutation of Γ. Hence in **R**-calculus Δ has to be accepted and retained. It is Γ that is to be revised.
2. To revise Γ, we delete the sentences in Γ that are inconsistent with Δ, so as to obtain all the **R**-contractions of Γ with respect to Δ. This is the purpose of defining **R**-calculus. In this sense **R**-calculus is a mechanism for revising Γ so as to make it consistent with Δ.
3. More specifically, **R**-calculus is constructed by defining formal inference rules to delete sentences in Γ according to the semantics of every logical connective symbol or quantifier symbol. The remaining sentences form a maximal subset of Γ, which is consistent with Δ.

Definition 7.13 (R-calculus). **R**-*calculus* is a formal inference system on **R**-configurations. It consists of four sets of rules: **R**-axiom, **R**-logical connective symbol rules, **R**-quantifier symbol rules, and **R**-cut rules.

For ease of understanding, we define each of these rules separately with explanations.

Definition 7.14 (R-axiom).

$$A, \Delta \mid \neg A, \Gamma \Longrightarrow A, \Delta \mid \Gamma.$$

The **R**-axiom shows that if the formal theory on the right-hand side of the **R**-configuration contains $\neg A$, then it is inconsistent with the formal refutation A on the left-hand side and hence $\neg A$ on the right-hand side of the **R**-configuration must be deleted.

The rules on logical connective symbols and quantifier symbols are defined below. They are all written in the form of a fraction, with which we are familiar.

The fraction here means that if the **R**-transition in its numerator holds, then the **R**-transition in its denominator also holds.

Definition 7.15 (R-$\wedge$ rule).

$$\frac{\Delta \mid A,\Gamma \Longrightarrow \Delta \mid \Gamma}{\Delta \mid A \wedge B,\Gamma \Longrightarrow \Delta \mid \Gamma} \qquad \frac{\Delta \mid B,\Gamma \Longrightarrow \Delta \mid \Gamma}{\Delta \mid A \wedge B,\Gamma \Longrightarrow \Delta \mid \Gamma}$$

This rule indicates that if A is deleted, then $A \wedge B$ must be deleted. Similarly, if B is deleted, then $A \wedge B$ must be deleted.

Take the rule on the left-hand side as an example. If $\Delta = \{\neg A\}$, then according to the **R**-axiom, A in the numerator of the rule must be deleted. According to the **G** system, $\Delta \vdash \neg A$ being provable indicates that $\Delta \vdash \neg A \vee \neg B$ is provable and hence $\Delta \vdash \neg(A \wedge B)$ is provable. This shows that $A \wedge B$ is inconsistent with Δ and must be deleted as well.

Definition 7.16 (R-$\vee$ rule).

$$\frac{\Delta \mid A,\Gamma \Longrightarrow \Delta \mid \Gamma \qquad \Delta \mid B,\Gamma \Longrightarrow \Delta \mid \Gamma}{\Delta \mid A \vee B,\Gamma \Longrightarrow \Delta \mid \Gamma}$$

This rule shows that if A and B are deleted respectively, then $A \vee B$ must be deleted.

For instance, if $\Delta = \{\neg A, \neg B\}$, then according to the **R**-axiom, A and B in the numerator should be deleted respectively. According to the **G** system, $\Delta \vdash \neg A$ and $\Delta \vdash \neg B$ being both provable indicates that $\Delta \vdash \neg A \wedge \neg B$ is provable and hence $\Delta \vdash \neg(A \vee B)$ is provable. It follows that $A \vee B$ is inconsistent with Δ and must also be deleted.

Definition 7.17 (R-$\rightarrow$ rule).

$$\frac{\Delta \mid \neg A,\Gamma \Longrightarrow \Delta \mid \Gamma \qquad \Delta \mid B,\Gamma \Longrightarrow \Delta \mid \Gamma}{\Delta \mid A \rightarrow B,\Gamma \Longrightarrow \Delta \mid \Gamma}$$

The **R**-$\rightarrow$ rule can be treated as a special case of the **R**-$\vee$ rule.

Definition 7.18 (R-$\forall$ rule).

$$\frac{\Delta \mid A[t/x],\Gamma \Longrightarrow \Delta \mid \Gamma}{\Delta \mid \forall x A(x),\Gamma \Longrightarrow \Delta \mid \Gamma}$$

where t is a term.

The **R**-$\forall$ rule can be interpreted as: if there exists a term t such that $A[t/x]$ is inconsistent with Δ, then $\forall x A(x)$ is inconsistent with Δ and hence it must be deleted.

For instance, if $\Delta = \{\neg A[t/x]\}$, then according to the **R**-axiom, $A[t/x]$ in the numerator of the rule must be deleted. According to the **G** system, $\Delta \vdash \neg A[t/x]$ being provable indicates that $\Delta \vdash \neg \forall x A(x)$ is provable. This shows that $\forall x A(x)$ is inconsistent with Δ and must also be deleted.

Definition 7.19 (R-∃ rule).

$$\frac{\Delta \mid A[y/x], \Gamma \Longrightarrow \Delta \mid \Gamma}{\Delta \mid \exists x A(x), \Gamma \Longrightarrow \Delta \mid \Gamma}$$

where y is either x or an arbitrary eigen-variable, that is, the variable y is different from all the free variables in the denominator of the **R**-∃ rule.

This rule can be interpreted as: for every eigen-variable y, if $A[y/x]$ is deleted, then $\exists x A(x)$ must also be deleted.

For instance, if $\Delta = \{\neg A[y/x]\}$, then according to the **R**-axiom, $A[y/x]$ in the numerator of the rule must be deleted. According to the **G** system, $\Delta \vdash \neg A[y/x]$ being provable indicates that $\Delta \vdash \neg \exists x A(x)$ is provable. This shows that $\exists x A(x)$ is inconsistent with Δ and must also be deleted.

Definition 7.20 (R-cut rule-I).

$$\frac{\Gamma_1, A, \Gamma_2 \vdash C \qquad A \mapsto_{\mathcal{T}} C \qquad \Delta \mid C, \Gamma_2 \Longrightarrow \Delta \mid \Gamma_2}{\Delta \mid \Gamma_1, A, \Gamma_2 \Longrightarrow \Delta \mid \Gamma_1, \Gamma_2}$$

The numerator of the **R**-cut rule-I specifies the following conditions.

(1) $\Gamma_1, A, \Gamma_2 \vdash C$ is provable. This indicates that the formula C is a formal consequence of Γ_1, A, Γ_2.

(2) The condition $A \mapsto_{\mathcal{T}} C$ holds. This indicates that A is a necessary antecedent of C with respect to the proof tree $\mathcal{T}$ of $\Gamma_1, A, \Gamma_2 \vdash C$.

(3) The **R**-transition $\Delta \mid C, \Gamma_2 \Longrightarrow \Delta \mid \Gamma_2$ holds. This indicates that the formal consequence C of Γ_1, A, Γ_2 is refuted by Δ and hence must be deleted.

The **R**-cut rule-I shows that, when the conditions in the numerator are satisfied, A must be deleted from the right-hand side of the **R**-configuration $\Delta \mid \Gamma_1, A, \Gamma_2$ in the denominator, since A is a necessary antecedent of the formal consequence C.

The **R**-cut rule-I has another equivalent form that is often used in proofs.

Definition 7.21 (R-cut rule-II).

$$\frac{\Gamma_1, A \vdash B \quad A \mapsto_{\mathcal{T}} B \quad B, \Gamma_2 \vdash C \quad \Delta \mid C, \Gamma_2 \Longrightarrow \Delta \mid \Gamma_2}{\Delta \mid \Gamma_1, A, \Gamma_2 \Longrightarrow \Delta \mid \Gamma_1, \Gamma_2}$$

The numerator of the rule specifies the following conditions.

(1) Both $\Gamma_1, A \vdash B$ and $B, \Gamma_2 \vdash C$ are provable. This indicates that the formula C is a formal consequence of Γ_1, A, Γ_2, and B is required as a lemma in the proof of C. $\Gamma_1, A \vdash B$ indicates that the lemma B is provable.

(2) The condition $A \mapsto_{\mathcal{T}} B$ holds. This indicates that A is a necessary antecedent of B with respect to $\mathcal{T}$, the proof tree of $\Gamma_1, A, \Gamma_2 \vdash B$.

(3) The **R**-transition $\Delta \mid C, \Gamma_2 \Longrightarrow \Delta \mid \Gamma_2$ holds. This indicates that the formal consequence C of Γ_1, A, Γ_2 is refuted by Δ and must be deleted.

The **R**-cut rule-II shows that, when the conditions in the numerator are satisfied, A must be deleted from the right-hand side of the **R**-configuration $\Delta \mid \Gamma_1, A, \Gamma_2$ in the denominator, since A is a necessary antecedent of the formal consequence C.

Lemma 7.5. *The* **R***-cut rule-I holds if and only if the* **R***-cut rule-II holds.*

Proof. **Necessity:** Since both $\Gamma_1, A \vdash B$ and $B, \Gamma_2 \vdash C$ are provable, we denote their proof trees by $\mathcal{T}_1$ and $\mathcal{T}_2$ respectively. Then according to the cut rule of the **G** system, $\Gamma_1, A, \Gamma_2 \vdash C$ is also provable and its proof tree $\mathcal{T}$ is composed of $\mathcal{T}_1$, $\mathcal{T}_2$ and an instance of the cut rule.

Since B is a requisite lemma in the proof of C, B is an antecedent of C. The condition $A \mapsto_{\mathcal{T}_1} B$ indicates that A is a necessary antecedent of B.

Hence according to (4) of Definition 7.2, A is an antecedent of C as well. Then the definition of necessary antecedent indicates that $A \mapsto_{\mathcal{T}} C$ holds. In addition, the **R**-transition $\Delta \mid C, \Gamma_2 \Longrightarrow \Delta \mid \Gamma_2$ is a condition of the **R**-cut rule-II.

The above arguments show that, if the conditions in the numerator of the **R**-cut rule-II are satisfied, then the conditions in the numerator of the **R**-cut rule-I are satisfied as well. This means that one can deduce the **R**-cut rule-II from the **R**-cut rule-I.

Sufficiency: The conditions of the **R**-cut rule-I indicate that $\Gamma_1, A, \Gamma_2 \vdash C$ is provable. Let its corresponding proof tree be $\mathcal{T}$ such that the condition $A \mapsto_{\mathcal{T}} C$ holds. As in the axiom rule, $C, \Gamma_2 \vdash C$ is provable and C is a requisite lemma in the proof. In addition, the **R**-transition $\Delta \mid C, \Gamma_2 \Longrightarrow \Delta \mid \Gamma_2$ is a condition of the **R**-cut rule-I.

The above arguments show that, if the conditions in the numerator of the **R**-cut rule-I are satisfied, then the conditions in the numerator of the **R**-cut rule-II are satisfied as well. This implies that one can deduce the **R**-cut rule-I from the **R**-cut rule-II. □

Lemma 7.6 (R-¬ derived rule).

$$\frac{\Delta \mid A', \Gamma \Longrightarrow \Delta \mid \Gamma}{\Delta \mid A, \Gamma \Longrightarrow \Delta \mid \Gamma}$$

holds, where A and A' are specified by the following table:

A	$\neg(B \wedge C)$	$\neg(B \vee C)$	$\neg\neg B$	$\neg(B \to C)$	$\neg\forall x B(x)$	$\neg\exists x B(x)$
A'	$\neg B \vee \neg C$	$\neg B \wedge \neg C$	B	$B \wedge \neg C$	$\exists x \neg B(x)$	$\forall x \neg B(x)$

Proof. The above table indicates that $A \vdash A'$ is provable. Hence $A, \Gamma \vdash A'$ is also provable and $A \mapsto_{\mathcal{T}} A'$ holds with $\mathcal{T}$ being the proof tree of the sequent $A, \Gamma \vdash A'$. If $\Delta \mid A', \Gamma \Longrightarrow \Delta \mid \Gamma$ holds, then according to the **R**-cut rule-I,

$$\frac{A, \Gamma \vdash A' \qquad A \mapsto_{\mathcal{T}} A' \qquad \Delta \mid A', \Gamma \Longrightarrow \Delta \mid \Gamma}{\Delta \mid A, \Gamma \Longrightarrow \Delta \mid \Gamma}$$

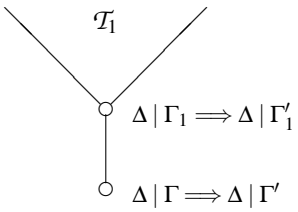
holds. Thus the **R**-¬ derived rule holds as well. □

Note that A in the derived rule is a composite formula. The formula A' is an expansion of the formula A with respect to ¬ and is equivalent to A. The derived rule can be interpreted as: if A' is deleted, then A must be deleted as well.

Definition 7.22 (R-inference tree and R-proof tree). Given an **R**-transition $\Delta \mid \Gamma \Longrightarrow \Delta \mid \Gamma'$, a tree $\mathcal{T}$ is called an **R**-*inference tree* of the **R**-transition if each node of $\mathcal{T}$ is an instance of the **R**-transition and the following are satisfied.

(1) A single node tree is an **R**-inference tree, if its node is an instance of the **R**-transition.

(2) Suppose that $\mathcal{T}_1$ is an **R**-inference tree whose root is the **R**-transition $\Delta \mid \Gamma_1 \Longrightarrow \Delta \mid \Gamma_1'$. If the following fraction (a) is an instance of a rule of **R**-calculus, then the tree structure



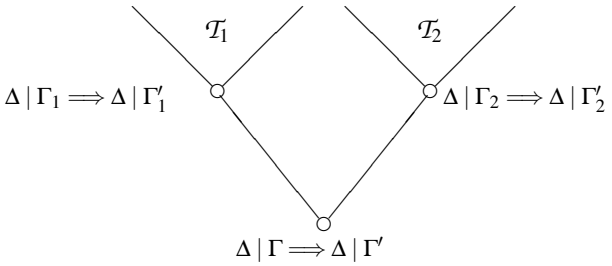
$$\frac{\Delta \mid \Gamma_1 \Longrightarrow \Delta \mid \Gamma_1'}{\Delta \mid \Gamma \Longrightarrow \Delta \mid \Gamma'}\ (a)$$

is an **R**-inference tree of $\Delta \mid \Gamma \Longrightarrow \Delta \mid \Gamma'$.

(3) Suppose that $\mathcal{T}_1$ and $\mathcal{T}_2$ are **R**-inference trees whose roots are $\Delta \mid \Gamma_1 \Longrightarrow \Delta \mid \Gamma_1'$ and $\Delta \mid \Gamma_2 \Longrightarrow \Delta \mid \Gamma_2'$ respectively. If the fraction

$$\frac{\Delta \mid \Gamma_1 \Longrightarrow \Delta \mid \Gamma_1' \qquad \Delta \mid \Gamma_2 \Longrightarrow \Delta \mid \Gamma_2'}{\Delta \mid \Gamma \Longrightarrow \Delta \mid \Gamma'}\ (b)$$

is an instance of a rule of **R**-calculus, then the tree structure



is an **R**-inference tree of $\Delta \mid \Gamma \Longrightarrow \Delta \mid \Gamma'$.

If $\mathcal{T}$ is a finite **R**-inference tree of the **R**-transition $\Delta \mid \Gamma \Longrightarrow \Delta \mid \Gamma'$ and its leaf nodes are all instances of the **R**-axiom, then $\mathcal{T}$ is called an **R**-*proof tree* of $\Delta \mid \Gamma \Longrightarrow \Delta \mid \Gamma'$.

Definition 7.23 (R-provable). We say that an **R**-transition is *provable* if its **R**-proof tree exists. Otherwise, we say that the **R**-transition $\Delta \mid \Gamma \Longrightarrow \Delta \mid \Gamma'$ is *unprovable*.

We call

$$\Delta \mid \Gamma \Longrightarrow \cdots \Longrightarrow \Delta \mid \Gamma_n \Longrightarrow \cdots \Longrightarrow \Delta \mid \Gamma'$$

an **R**-*transition sequence* and denote it as

$$\Delta \mid \Gamma \Longrightarrow^* \Delta \mid \Gamma'$$

with $\Longrightarrow^*$ denoting finite or countably infinite transitions. We say that $\Delta \mid \Gamma \Longrightarrow^* \Delta \mid \Gamma'$ is *provable* if every **R**-transition in the **R**-transition sequence $\Delta \mid \Gamma \Longrightarrow^* \Delta \mid \Gamma'$ is provable.

Definition 7.24 (R-termination). For a given **R**-configuration $\Delta \mid \Gamma$, if Δ and Γ are consistent, then $\Delta \mid \Gamma$ is called an **R**-*termination*.

In summary, an **R**-configuration $\Delta \mid \Gamma$ includes two parts: the left part and the right part. For an inconsistent **R**-configuration, Γ on its right-hand side is a formal theory and Δ on its left-hand side is the **R**-refutation of Γ, where Δ consists of atomic sentences or negations of atomic sentences. The role of Δ in the **R**-rules is to determine which sentences in Γ to delete.

The **R**-cut rule is different from the **R**-logical connective symbol rules and **R**-quantifier symbol rules. When a formal consequence of a formal theory is refuted, the **R**-cut rule can be used to delete necessary antecedents of the formal consequence. Since the proof tree of a sequent is not unique, we may obtain different **R**-transitions for different proof trees.

7.5 Some examples

In this section we demonstrate how to use **R**-calculus with three examples.

Example 7.6 (Application of R-cut ruie). Let Γ be the formal theory given in Example 7.5 in Section 7.3:

$$\Gamma = \{A, A \to B, B \to C, E \to F\}.$$

Applying the rules about $\to$ of **G** system, we can prove that $\Gamma \vdash C$ is provable. Suppose that the formal consequence C of Γ is refuted by facts, that is, $\neg C$ holds. There are three maximal contractions of Γ with respect to $\neg C$:

$$\{A, A \to B, E \to F\}, \quad \{A, B \to C, E \to F\}, \quad \{A \to B, B \to C, E \to F\}.$$

Applying the $\to$ rules of **G** system, we can deduce every maximal contraction listed above. Consider $\{A, A \to B, E \to F\}$ first and let

$$\Gamma_1 = \{A, A \to B\}, \quad \Gamma_2 = \{E \to F\}.$$

Applying modus ponens rule **G** system we know that

$$\Gamma_1, B \to C, \Gamma_2 \vdash C$$

is provable. $B \to C \mapsto C$ holds, since $B \to C$, an element of Γ, is a necessary antecedent of C. According to the **R**-axiom we also have

$$\neg C \mid C, \Gamma_2 \Longrightarrow \neg C \mid \Gamma_2.$$

Thus we can apply the **R**-cut rule-I to obtain

$$\neg C \mid \Gamma_1, B \to C, \Gamma_2 \Longrightarrow \neg C \mid \Gamma_1, \Gamma_2.$$

Here Γ_1, Γ_2 is just $\{A, A \to B, E \to F\}$ which is the first maximal contraction listed above.

Using the **R**-cut rule we can deduce the second maximal contraction $\{A, B \to C, E \to F\}$. Let

$$\Gamma_1 = \{A\}, \quad \Gamma_2 = \{B \to C, E \to F\}.$$

According to the **G** system we know that both

$$\Gamma_1, A \to B \vdash B \text{ and } B, \Gamma_2 \vdash C$$

are provable. $A \to B \mapsto B$ holds since $A \to B$, an element of Γ, is a necessary antecedent of B. Further, the **R**-axiom indicates that

$$\neg C \mid C, \Gamma_2 \Longrightarrow \neg C \mid \Gamma_2.$$

By the **R**-cut rule-II,

$$\neg C \mid \Gamma_1, A \to B, \Gamma_2 \Longrightarrow \neg C \mid \Gamma_1, \Gamma_2$$

holds. Here Γ_1, Γ_2 is just $\{A, B \to C, E \to F\}$ which shows the deduction of the second maximal contraction by **R**-calculus. Finally, let

$$\Gamma_1 = \varnothing, \quad \Gamma_2 = \{A \to B, B \to C, E \to F\}.$$

Using the **R**-cut rule we can deduce the third maximal contraction $\{A \to B, B \to C, E \to F\}$.

Example 7.7.[1] Let

$$\Delta = \{A, B\}, \quad \Gamma = \{A \to C, C \to \neg B\}.$$

We can prove that A and B individually are consistent with Γ, but that $\{A, B\}$ is inconsistent with Γ. Under such circumstances we can still use **R**-calculus to obtain the maximal contraction of Γ with respect to Δ.

According to the **R**-axiom, both

$$A, B \mid \neg A \Longrightarrow A, B \mid \varnothing$$

and

$$A, B \mid \neg B \Longrightarrow A, B \mid \varnothing.$$

Then according to the **R**-$\vee$ rule,

$$A, B \mid \neg A \vee \neg B \Longrightarrow A, B \mid \varnothing.$$

[1] This example was provided by Jie Luo.

Since $\Gamma \vdash A \to \neg B$ is provable by the **G** system, that is, $\Gamma \vdash \neg A \vee \neg B$ is provable. It is also easy to prove that $A \to C$ is a necessary antecedent of $\neg A \vee \neg B$. Using the **R**-cut rule-I, we can prove that

$$A, B \mid A \to C, C \to \neg B \Longrightarrow A, B \mid C \to \neg B.$$

So $\{C \to \neg B\}$ is a maximal contraction of Γ with respect to Δ. Similarly, we can use **R**-calculus to obtain another maximal contraction $\{A \to C\}$ of Γ with respect to Δ.

The following example is on the rationality of **R**-calculus.

Example 7.8. Let Γ' be $\{\neg A \wedge B\}$ and Δ be $\{A\}$. Evidently, $\Gamma' \vdash \neg A$ is provable. According to the **R**-$\wedge$ rule we have

$$A \mid \neg A \wedge B \Longrightarrow A \mid \varnothing.$$

However, if we let Γ' be $\{\neg A,\ B\}$ and Δ still be $\{A\}$, then the **R**-axiom indicates that

$$A \mid \neg A,\ B \Longrightarrow A \mid B.$$

This seems to be irrational because generally speaking, the formal theories $\{\neg A \wedge B\}$ and $\{\neg A, B\}$ seem to be the same, or at least their semantics are the same. Hence the results remain the same after the deletions by **R**-calculus.

Our justification for this is the following: as an axiom of the refuted formal theory, $\neg A \wedge B$ is inconsistent with A and thus should be deleted. However, the formal theory $\{\neg A, B\}$ contains two axioms but only $\neg A$ is inconsistent with A. In this case, $\neg A \wedge B$ is just a formal consequence of the theory.

7.6 Special theory of relativity

Einstein explained how he discovered the special theory of relativity in [Einstein, 1921]. Let us use **R**-calculus to verify his informal reasoning. We use the predicate R to denote the principle of relativity, N_1, N_2, N_3 the Newton's three laws of motion respectively, and E the law of universal gravitation. We use the atomic formula $B[c]$ to denote the photon is a particle. Finally, the Galilean transformation V can be described by the sentence

$$\forall x(B(x) \to A(x)).$$

We use the following formal theory to describe the laws in classical physics before the special theory of relativity:

$$\{B[c], \forall x(B(x) \to A(x)), R, N_1, N_2, N_3, E\}.$$

We know that

$$B[c], \forall x(B(x) \to A(x)) \vdash A[c] \tag{7.1}$$

is provable in the **G** system. This sequent can be interpreted as: classical physics predicts that the velocity of light in K' depends on the velocity of K' relative to K.

However, multiple experiments show that the velocity of light does not depend on the velocity of the luminous body, i.e., they support the negation $\neg A[c]$. In this case, $A[c]$ is refuted by experiments. With brilliant logical intuition, Einstein concluded that the Galilean transformation should be deleted.

We can now use **R**-calculus to verify the correctness of Einstein's intuition. Let

$$\Gamma = \{B[c], \forall x(B(x) \to A(x)), R, N_1, N_2, N_3, E\},$$
$$\Gamma' = \{B[c], R, N_1, N_2, N_3, E\}.$$

Since $\neg A[c]$ is supported by experiments and observations, it is a refutation by facts and has to be accepted. Moreover, as $\Gamma \vdash A[c]$ is provable, $\neg A[c]$ is a formal refutation of Γ, and $\{B[c], \neg A[c]\}$ is an **R**-refutation of Γ.

$$B[c], \neg A[c] \mid A[c], \Gamma' \Longrightarrow B[c], \neg A[c] \mid \Gamma'$$

is an instance of the **R**-axiom, and so is

$$B[c], \neg A[c] \mid \neg B[c], \Gamma' \Longrightarrow B[c], \neg A[c] \mid \Gamma'.$$

According to the **R**-$\to$ rule, we know that the **R**-transition

$$B[c], \neg A[c] \mid B[c] \to A[c], \Gamma' \Longrightarrow B[c], \neg A[c] \mid \Gamma'$$

holds, which means $B[c] \to A[c]$ is to be deleted. Finally, according to the **R**-$\forall$ rule we know that the **R**-transition

$$B[c], \neg A[c] \mid \forall x(B(x) \to A(x)), \Gamma' \Longrightarrow B[c], \neg A[c] \mid \Gamma'$$

also holds. This amounts to having $\forall x(B(x) \to A(x))$ deleted, in other words, the Galilean transformation should be deleted.

The above example demonstrates that mathematical logic plays the following two roles in scientific discovery.

Firstly, the form of (7.1) as a provable sequent suggests that, in general, we may be able to use the **G** system to help predict new results in physics.

Secondly, the result deduced by **R**-calculus confirms Einstein's intuition that the Galilean transformation should be deleted. This shows that in scientific discovery, especially when the existing theory does not coincide with the results of experiments, i.e., when there are refutations by facts, **R**-calculus can help to make the correct revision to the theory.

7.7 Darwin's theory of evolution

We should point out that in the above section, when there are refutations by facts, there is only one correct choice, which is to accept the constancy of the velocity of light and to delete the Galilean transformation.

In the following example we will see that, when there are refutations by facts, it is possible that scientists might face several choices. In such circumstances, **R**-calculus can deduce all the possible choices, i.e., all the maximal contractions of the existing theory consistent with the refutations. This will support scientists to build their revised theory.

Using the **R**-cut rule, we can give a logical verification for Darwin's theory of evolution and adapt Example 7.6 to explain why the controversies over Darwin's theory have lasted for 150 years.

Let us first consider the theory described in the introduction of [Darwin 1859], which was prevalent up to that time. We may call it the theory of immutability. We first need to introduce a first-order language $\mathscr{B}$ to describe this theory. To do so, we introduce the predicate symbols A, B and C as follows:

A stands for "*Each species has been independently created*." (p7, ibid.).
B stands for "*Species are immutable*." (ibid.)
C stands for "*[Species] belonging to the same genera are not lineal descendants of some other and generally extinct species*." (ibid.)

According to the semantics of $\rightarrow$, the formula $A \rightarrow B$ is interpreted as the sentence: "If each species has been independently created, then species are immutable."

And the formula $B \rightarrow C$ is interpreted as the sentence: "If species are immutable, then species in the same genera are not lineal descendants of some other and generally extinct species."

Thus the theory of immutability is described in $\mathscr{B}$ by the set of formulas

$$\Omega: \ \{A, A \rightarrow B, B \rightarrow C\}.$$

As we learned from the previous chapters, we can say that the theory of immutability is a model of the formal theory Ω, or Ω is the formal theory of immutability.

Through analyzing the fossil record and by observing species that were obviously related but had diverged due to isolation in a different environment, Darwin concluded:

"*[Species] belonging to the same genera are lineal descendants of some other and generally extinct species*." (p.7, [Darwin 1859])

Let us call this *the statement of genera ancestors*. It is denoted in $\mathscr{B}$ by $\neg C$.

$\neg C$ is a **R**-refutation with respect to Ω since C is proved to be a logical consequence of Ω.

Having established that species evolve, Darwin turned his attention to "*how the innumerable species inhabiting this world have been modified*." (p.5, [Darwin 1859]) i.e., the mechanism of evolution. Based on his observation of how species in a genera had adapted to different environments he proposed the law of natural selection in his book [Darwin 1859]. He expressed it as:

"*As many more individuals of each species are born than can possibly survive; and as, consequently, there is a frequently recurring struggle for existence, it follows that any being, if it vary however slightly in any manner profitable to itself, under the complex and*

sometimes varying conditions of life, will have a better chance of surviving, and thus be naturally selected." (p.6, [Darwin 1859]).

Let us introduce the following predicate symbols E and F into the first-order language $\mathscr{B}$. Let:

E stand for "*As many more individuals of each species are born than can possibly survive*."
F stand for "*there is a frequently recurring struggle for existence, ... will have a better chance of surviving*."

The principle of natural selection can be described in $\mathscr{B}$ by the formula

$$E \to F.$$

We should note that the facts collected by Darwin support the statement of genera ancestors and the principle of natural selection. These facts are described in $\mathscr{B}$ by

$$\{\neg C, E \to F\}.$$

Using the model checking method [Gallier, 1986] and assigning the truth value T to all following symbols, we can prove that $E \to F$ is consistent with $A, A \to B$, and $B \to C$. In other words, the principle of natural selection does **not** contradict the theory of the immutability Ω.

Thus, the key is to find and delete the principles contained in Ω which lead to contradiction with $\{\neg C\}$.

To do so, let Δ be $\{\neg C\}$ and Γ be $\{A, A \to B, B \to C, E \to F\}$. The deduction of the **R**-configuration $\Delta \mid \Gamma$ is done by applying the **R**-cut rule as the following: Let

$$\Gamma_1 = \varnothing \quad \Gamma_2 = \{A \to B, B \to C, E \to F\}.$$

Firstly, all three premises of the **R**-cut rule hold because

1. $\Gamma_1, A, \Gamma_2 \vdash C$;
2. A is a necessary antecedent of C, i.e., $A \mapsto C$ holds;
3. By applying the **R**-axiom,

$$\neg C \mid C, \Gamma_2 \Longrightarrow \neg C \mid \Gamma_2.$$

Thus, by applying the **R**-cut rule,

$$\neg C \mid \Gamma_1, A, \Gamma_2 \Longrightarrow \neg C \mid \Gamma_1, \Gamma_2$$

holds. This means that A should be deleted from the theory of immutability. Let

$$\Lambda_1 \quad \text{be} \quad \{A \to B, B \to C, E \to F\}.$$

Λ_1 is the union of Γ_1, Γ_2, and is an **R**-contraction of Γ with respect to $\neg C$. In other words, Λ_1 is a maximal subset of Γ which is consistent with $\neg C$. Let

$$\Xi_1 \quad \text{be} \quad \{\neg C\} \cup \{A \to B, B \to C, E \to F\}.$$

Using the method of model checking, we can prove that Ξ_1 is consistent. Ξ_1 contains Darwin's statement of genera ancestors $\neg C$ and the principle of natural selection $E \to F$. Moreover, Ξ_1 inherits the principles of the theory of immutability of species $A \to B$ and $B \to C$. Note that Ξ_1 is a solution deduced formally by **R**-calculus. This indicates that the above symbolic deduction is a logical verification of Ξ_1.

Darwin's theory of Evolution, proposed in [Darwin 1859], contains Ξ_1 but, in fact, Darwin went a step further. He added his belief: "*each species has been independently created is erroneous*." It is obvious that this belief can be described by $\neg A$. Darwin put $\neg A$ together with Ξ_1, and formed his theory of evolution, which is described symbolically by $\{\neg A\} \cup \Xi_1$.

In the introduction of [Darwin 1859], Darwin claimed that "*I can entertain no doubt, after the most deliberate study and dispassionate judgement of which I am capable, that the view which most naturalists entertain, and which I formerly entertained namely, that each species has been independently created is erroneous*."

Hence, if we let

$$\Xi^+ \quad \text{be} \quad \{\neg C, A \to B,\ B \to C,\ E \to F\} \cup \{\neg A\}$$

then the core of Darwin's theory of evolution should be described by Ξ^+.

The above discussion shows that $\neg A$ is consistent with Darwin's two famous contributions described by $\{\neg C, E \to F\}$. But $\neg A$ is just a belief, and neither Darwin nor later biologists have been able to find direct evidence to support $\neg A$. This is the root of the still ongoing controversy about Darwin's Theory. We can easily see that, if we accept $\{\neg C, E \to F\}$ we can derive two other logically rational **R**-contractions of Γ with respect to Δ as we saw in Example 7.6:

$$\begin{aligned} \Lambda_2: &\quad \{A, B \to C, E \to F\}, \\ \Lambda_3: &\quad \{A, A \to B, E \to F\}. \end{aligned}$$

Putting Δ together with Λ_2 and Λ_3 we obtain two new theories of evolution. They are

$$\begin{aligned} \Xi_2: &\quad \{\neg C\} \cup \{A, B \to C, E \to F\}, \\ \Xi_3: &\quad \{\neg C\} \cup \{A, A \to B, E \to F\}. \end{aligned}$$

It is obvious that Ξ_2 and Ξ_3 are different from Darwin's theory of evolution described by Ξ^+. The existence of these two theories indicates the following:

Firstly, both Λ_2 and Λ_3 are **R**-contractions of Γ with respect to $\neg C$. These mean that Ξ_2 and Ξ_3 are revisions of the theory of immutability with respect to $\neg C$.

Secondly, both Ξ_2 and Ξ_3 contain $\neg C$ and $E \to F$. These mean that they include Darwin's two main contributions: the statement of genera ancestors and the principle of natural selection.

Thirdly, both Ξ_2 and Ξ_3 contain A. This means that the theories believe "*each species has been independently created*."

Fourthly, both Ξ_2 and Ξ_3 are inconsistent with Ξ^+ since Ξ^+ contains $\neg A$. This means that Darwin's theory of evolution believes the negation of A, but the other two do not.

The above results show that the statement of genera ancestors and the principle of natural selection are both logically independent of A.

The existence of the other two theories is the "*origin*" of the controversies over Darwin's theory. Neither A nor $\neg A$ has gathered direct evidence to decide between these possible theories from a logical standpoint.

Sensitive readers may feel that using $E \to F$ to describe the principle of natural selection is too simple and may lose the logical relations between the propositions introduced above. However, we can introduce more predicate symbols to describe the more detailed logical structure of the principle. For example, we can introduce three new predicate symbols P, B, and Q in $\mathscr{B}$ and let:

P stand for "*there is a frequently recurring struggle for existence*."
B stand for "*species are immutable*."
Q stand for "*any being, it vary however slightly $\cdots$ and thus be naturally selected*."

Thus the principle of natural selection can be described by

$$E \to (P \to (\neg B \to Q)).$$

The above formula describes logical relations between the statements appearing in the principle of natural selection. We can verify that $E \to (P \to (\neg B \to Q))$ is still consistent with $\{A, A \to B, B \to C\}$. Therefore, the basic results contained in the above formal verification still remain valid.

7.8 Reachability of R-calculus

As we saw in the previous sections, for a given **R**-refutation Δ, **R**-calculus can deduce all the **R**-contractions of Γ with respect to Δ. This is called the reachability of **R**-calculus, which we shall prove in this section.

Definition 7.25 (R-reachability). If for any given inconsistent **R**-configuration $\Delta \mid \Gamma$ and an arbitrary **R**-contraction Γ' of Γ with respect to Δ, there always exists an **R**-transition sequence such that

$$\Delta \mid \Gamma \Longrightarrow^* \Delta \mid \Gamma'$$

is provable, and $\Delta \mid \Gamma'$ is an **R**-termination, then we say that **R**-calculus is **R**-*reachable*.

Theorem 7.2 (R-reachability). **R**-*calculus is* **R**-*reachable.*

Proof. Suppose that $\Delta \mid \Gamma$ is a given inconsistent **R**-configuration with $\Delta = \{A_1, A_2, \ldots, A_n\}$. Γ' is an arbitrary **R**-contraction of Γ with respect to Δ. In what follows we prove that the **R**-transition sequence

$$\Delta \mid \Gamma \Longrightarrow^* \Delta \mid \Gamma'$$

is provable with $\Delta \mid \Gamma'$ being an **R**-termination.

Let $\Gamma'' = \Gamma - \Gamma'$. In the following we show that for every $B \in \Gamma''$, we can use **R**-calculus to delete B. First, let $\Gamma_1 = \Gamma'' - \{B\}$ and $\Gamma_2 = \Gamma'$. Since $\Gamma_1, B \vdash B$ is provable, according to Definition 7.2, B is an antecedent of B and $B \in \Gamma$; hence $B \mapsto B$. According to the definition of **R**-contraction, $\Gamma_2 \cup \{B\}$ is inconsistent with Δ, that is, $\Delta \mid \Gamma_2, B$ is an inconsistent **R**-configuration. As per Lemma 7.4, $\Gamma_2, B \vdash \neg(A_1 \wedge \cdots \wedge A_n)$ is provable and B is the lemma requisite in the proof. In addition, the **R**-axiom indicates that

$$\Delta \mid \neg(A_1 \wedge \cdots \wedge A_n), \Gamma_2 \Longrightarrow \Delta \mid \Gamma_2$$

is provable. Hence all the conditions in the numerator of the **R**-cut rule-II are satisfied and thus

$$\Delta \mid \Gamma_1, B, \Gamma_2 \Longrightarrow \Delta \mid \Gamma_1, \Gamma_2$$

is provable, i.e., B is deleted. Using the same method we can delete every element in Γ''. In this way we shall obtain an **R**-transition sequence

$$\Delta \mid \Gamma \Longrightarrow^* \Delta \mid \Gamma'$$

such that every **R**-transition in the **R**-transition sequence is provable, which amounts to the **R**-transition sequence being provable.

Since Γ' is a maximal subset of Γ consistent with Δ, according to Definition 7.24, $\Delta \mid \Gamma'$ is an **R**-termination. Thus **R**-calculus is **R**-reachable. □

The converse theorem of Theorem 7.2 does not hold. In fact, for an arbitrary **R**-transit-ion sequence

$$\Delta \mid \Gamma \Longrightarrow^* \Delta \mid \Gamma'$$

with $\Delta \mid \Gamma'$ being an **R**-termination, Γ' is not necessarily a maximal contraction of Γ with respect to Δ. Consider the following example.

Example 7.9. Let Γ be

$$\{A, A \to B, B \to C, A \to E, E \to C\}.$$

According to the **G** system, $\Gamma \vdash C$ holds. Consider the formal refutation $\neg C$ of Γ. As we saw in Example 7.6, we can use the **R**-cut rule to delete $A \to B$. Also, since

$$A, A \to E, E \to C \vdash C,$$

we can apply the **R**-cut rule again to delete A and obtain

$$\{B \to C, A \to E, E \to C\}.$$

Nonetheless the above formula is not a maximal subset of Γ with respect to $\neg C$ because the **R**-contraction is

$$\{A \to B, B \to C, A \to E, E \to C\}.$$

Lemma 7.7. *For any* **R***-logical connective symbol rule,* **R***-quantifier symbol rule or* **R***-cut rule, if the* **R***-configuration on the left-hand side of* $\Longrightarrow$ *in the* **R***-transition in the denominator of the rule is an* **R***-termination, then the* **R***-configuration on the left-hand side of* $\Longrightarrow$ *in at least one of the* **R***-transitions in the numerator of the rule is an* **R***-termination.*

Proof. In what follows we shall prove by contradiction that each **R**-rule has the property specified by the lemma. The symbols in the proof are the same as in Definitions 7.15–7.20.

(1) For the **R**-$\wedge$ rule, suppose that $\Delta \mid A, \Gamma$ is not an **R**-termination. Then Δ is inconsistent with $\{A\} \cup \Gamma$. According to (2) of Lemma 3.7, $\Delta, A, \Gamma \vdash \neg B$ is provable. By the $\neg$-*R* rule, this amounts to $\Delta, A, B, \Gamma \vdash$ being provable. Then the $\wedge$-*L* rule indicates that $\Delta, A \wedge B, \Gamma \vdash$ is provable. As per the $\neg$-*R* rule, this amounts to $\Delta, \Gamma \vdash \neg(A \wedge B)$ being provable. Then according to the definition of consistency, $\Delta \cup \Gamma$ is inconsistent with $A \wedge B$, which contradicts $\Delta \mid A \wedge B, \Gamma$ being an **R**-termination. Thus $\Delta \mid A, \Gamma$ is an **R**-termination.

(2) For the **R**-$\vee$ rule, suppose that neither $\Delta \mid A, \Gamma$ nor $\Delta \mid B, \Gamma$ is an **R**-termination. According to the definition of **R**-termination, Δ is inconsistent with both $\{A\} \cup \Gamma$ and $\{B\} \cup \Gamma$. As per (2) of Lemma 3.7, both $\Delta, A, \Gamma \vdash \neg(A \vee B)$ and $\Delta, B, \Gamma \vdash \neg(A \vee B)$ are provable. By the $\vee$-*L* rule, this amounts to $\Delta, A \vee B, \Gamma \vdash \neg(A \vee B)$ being provable. Then the $\neg$-*R* rule indicates that $\Delta, \Gamma \vdash \neg(A \vee B)$ is provable. By the definition of consistency, $\Delta \cup \Gamma$ is inconsistent with $A \vee B$, which contradicts $\Delta \mid A \vee B, \Gamma$ being an **R**-termination. Thus at least one of $\Delta \mid A, \Gamma$ and $\Delta \mid B, \Gamma$ is an **R**-termination.

(3) The proof for the **R**-$\longrightarrow$ rule is similar to that for the **R**-$\vee$ rule.

(4) The proofs for the **R**-$\forall$ rule and **R**-$\exists$ rule are similar to that for the **R**-$\wedge$ rule.

(5) For the **R**-cut rule, suppose that $\Delta \mid C, \Gamma_2$ is not an **R**-termination. By the definition of **R**-termination, Δ is inconsistent with $\{C\} \cup \Gamma_2$. Then according to (2) of Lemma 3.7, $\Delta, C, \Gamma_2 \vdash \neg C$ is provable. The $\neg$-*R* rule indicates that $\Delta, \Gamma_2 \vdash \neg C$ is provable. Hence according to Lemma 3.6, $\Delta, \Gamma_1, A, \Gamma_2 \vdash \neg C$ is provable as well. Since $\Gamma_1, A, \Gamma_2 \vdash C$ is provable as a condition, $\Delta, \Gamma_1, A, \Gamma_2 \vdash C$ is provable as well. Then the definition of consistency indicates that $\Delta \cup \Gamma_1 \cup \{A\} \cup \Gamma_2$ is inconsistent. This contradicts $\Delta \mid \Gamma_1, A, \Gamma_2$ being an **R**-termination. Hence $\Delta \mid C, \Gamma_2$ is an **R**-termination. □

Lemma 7.8. *If* $\Delta \mid \Gamma$ *is an* **R***-termination, then there does not exist any formal theory* $\Gamma' \subset \Gamma$ *such that the* **R***-transition*

$$\Delta \mid \Gamma \Longrightarrow \Delta \mid \Gamma'$$

is provable.

Proof. We prove the lemma by contradiction. Suppose that there exists a $\Gamma' \subset \Gamma$ such that the **R**-transtition

$$\Delta \mid \Gamma \Longrightarrow \Delta \mid \Gamma'$$

is provable. It suffices to prove that there exists a path in its **R**-proof tree $\mathcal{T}$ connecting the root and a leaf node such that all the **R**-configurations on the left-hand side of $\Longrightarrow$ in the **R**-transitions of the nodes on the path are **R**-terminations. Nonetheless, the leaf node of this path is an instance of the **R**-axiom and thus the **R**-configuration on the left-hand side of $\Longrightarrow$ in its **R**-transition cannot be an **R**-termination, which leads to a contradiction. Now according to the structural inductive definition of the **R**-proof tree, i.e., Definition 7.22, we make a structural induction on $\mathcal{T}$ to prove the existence of such a path.

(1) If $\mathcal{T}$ is a single node tree, then it is

$$\Delta \mid \Gamma \Longrightarrow \Delta \mid \Gamma'.$$

$\Delta \mid \Gamma$ is an **R**-termination and thus the path from the root to itself suffices.

(2) The formula (a) in (2) of Definition 7.22 can only be an instance of the **R**-$\wedge$ rule, **R**-$\forall$ rule, **R**-$\exists$ rule or **R**-cut rule. Lemma 7.7 indicates that the **R**-configuration $\Delta \mid \Gamma_1$ in the formula (a) is an **R**-termination. According to the inductive hypothesis, there exists a path in the subtree $\mathcal{T}_1$ in (2) of Definition 7.22 connecting its root and one of its leaf nodes such that all the **R**-configurations on the left-hand side of $\Longrightarrow$ in the **R**-transitions on the path are **R**-terminations. Hence, if we add the path represented by the formula (a) to the above-mentioned path, then we shall obtain a path in $\mathcal{T}$ connecting its root and one of its leaf nodes such that all the **R**-configurations on the left-hand side of $\Longrightarrow$ in the **R**-transitions on the path are **R**-terminations.

(3) The formula (b) in (3) of Definition 7.22 can only be an instance of the **R**-$\vee$ rule or **R**-$\rightarrow$ rule. Lemma 7.7 indicates that at least one of the **R**-configurations on the left-hand side of $\Longrightarrow$ in the two **R**-transitions in the numerator of the rule is an **R**-termination. Suppose that $\Delta \mid \Gamma_1$ in the formula (b) is an **R**-termination. According to the inductive hypothesis, there exists a path in the subtree $\mathcal{T}_1$ in (3) of Definition 7.22 connecting its root and one of its leaf nodes such that all the **R**-configurations on the left-hand side of $\Longrightarrow$ in the **R**-transitions on the path are **R**-terminations. Hence, if we add the path connecting $\Delta \mid \Gamma \Longrightarrow \Delta \mid \Gamma'$ and $\Delta \mid \Gamma_1 \Longrightarrow \Delta \mid \Gamma_1'$ in the formula (b) to the above-mentioned path, then we shall obtain a path in $\mathcal{T}$ connecting its root and one of its leaf nodes such that all the **R**-configurations on the left-hand side of $\Longrightarrow$ in the **R**-transitions on the path are **R**-terminations.

However, the **R**-configuration on the left-hand side of $\Longrightarrow$ in the **R**-axiom cannot be an **R**-termination since its two formula sets on the two sides of $\mid$ contain A and $\neg A$ respectively and thus cannot be consistent.

Hence the hypothesis is false and the lemma is proved. □

7.9 Soundness and completeness of R-calculus

Each rule of **R**-calculus is a deletion rule on some sentence. Since such deletions are determined by the semantics of the logical connective symbols and quantifier symbols, the **R**-rules are also calculus rules on these logical symbols. As a result, one has to investigate

the soundness and completeness of **R**-calculus. In this section we first explain what the soundness and completeness of **R**-calculus are. Then under the prerequisite of the **R**-reachability, we prove that **R**-calculus is both sound and complete.

Definition 7.26 (R-soundness). Let $\Delta \mid \Gamma$ be an inconsistent **R**-configuration and Γ' be an **R**-contraction of Γ with respect to Δ. That is, there exists a provable **R**-transition sequence

$$\Delta \mid \Gamma \Longrightarrow^* \Delta \mid \Gamma'.$$

If there exists a model $\mathbf{M}$ of **R**-refutation such that both $\mathbf{M} \models \Delta$ and $\Gamma_{\mathbf{M}(\Delta)} = \Gamma'$ hold, then we say that **R**-calculus is **R**-*sound*.

Theorem 7.3 (R-soundness). **R**-*calculus is* **R**-*sound.*

Proof. For an inconsistent **R**-configuration $\Delta \mid \Gamma$, let Γ' be an **R**-contraction of Γ with respect to Δ. The definition of **R**-contraction indicates that Γ' and Δ are consistent. Hence $\Gamma' \cup \Delta$ is satisfiable, i.e., there exists a model $\mathbf{M}$ such that $\mathbf{M} \models \Gamma' \cup \Delta$ holds. Since Γ' is the maximal subset of Γ that is consistent with Δ, $\Gamma_{\mathbf{M}(\Delta)} = \Gamma'$ holds. Hence $\mathbf{M}$ is the model of **R**-refutation of Γ with respect to Δ. Thus **R**-calculus is **R**-sound. □

Definition 7.27 (R-completeness). If for an arbitrary inconsistent **R**-configuration $\Delta \mid \Gamma$ and an arbitrary model $\mathbf{M}$ of **R**-refutation of Γ with respect to Δ, there always exists a provable **R**-transition sequence

$$\Delta \mid \Gamma \Longrightarrow^* \Delta \mid \Gamma_{\mathbf{M}(\Delta)},$$

then we say that **R**-calculus is **R**-*complete*.

Theorem 7.4 (R-completeness). **R**-*calculus is* **R**-*complete.*

Proof. For an inconsistent **R**-configuration $\Delta \mid \Gamma$ with $\Delta = \{A_1, \ldots, A_n\}$, if the model $\mathbf{M}$ is a model of **R**-refutation of Γ with respect to Δ, then $\mathbf{M}$ is a model of refutation by facts of Γ with respect to $A_1 \wedge \cdots \wedge A_n$. According to Theorem 7.1, $\Gamma_{\mathbf{M}(A_1 \wedge \cdots \wedge A_n)}$ is a maximal contraction of Γ with respect to $A_1 \wedge \cdots \wedge A_n$, i.e., an **R**-contraction of Γ with respect to Δ. Since $\Gamma_{\mathbf{M}(A_1 \wedge \cdots \wedge A_n)} = \Gamma_{\mathbf{M}(\Delta)}$, $\Gamma_{\mathbf{M}(\Delta)}$ is an **R**-contraction of Γ with respect to Δ. According to Theorem 7.2 on reachability, there exists a provable **R**-transition sequence

$$\Delta \mid \Gamma \Longrightarrow^* \Delta \mid \Gamma_{\mathbf{M}(\Delta)}.$$

Thus **R**-calculus is **R**-complete. □

7.10 Basic theorem of testing

All Γ in the examples of Section 7.5 are finite formal theories, that is, they are all consistent sets of sentences. The following example shows that even if Γ is inconsistent, **R**-calculus is still able to deduce every maximal subset of Γ that is consistent with Δ.

Example 7.10 (Inconsistent formula set).[2] Let

$$\Delta = \{x \doteq x\}, \quad \Gamma = \{f(x) \doteq y, f(y) \doteq z, \neg(f(f(x)) \doteq z)\}.$$

Γ is not a formal theory. In fact, since $f(x) \doteq y$, we can substitute the variable y in $f(y) \doteq z$ by $f(x)$ to obtain $f(f(x)) \doteq z$. This formula is inconsistent with $\neg(f(f(x)) \doteq z)$.

By using the **R**-cut rule-I, we can obtain all the maximal subsets of Γ that are consistent with Δ. For instance, let $\Gamma_1 = \{f(x) \doteq y\}$ and $\Gamma_2 = \{\neg(f(f(x)) \doteq z)\}$. First, the transitivity of $\doteq$ indicates that

$$\Gamma_1, f(y) \doteq z \vdash f(f(x)) \doteq z$$

is provable. The $\neg$-*L* rule and axiom rule further indicate that

$$f(f(x)) \doteq z, \Gamma_2 \vdash \neg(x \doteq x)$$

is provable. Hence according to the cut rule of the **G** system,

$$\Gamma_1, f(y) \doteq z, \Gamma_2 \vdash \neg(x \doteq x)$$

is provable. It is not difficult to prove that $f(y) \doteq z$ is a necessary antecedent of $\neg(x \doteq x)$. Then by the **R**-axiom, the **R**-transition

$$x \doteq x \mid \neg(x \doteq x) \Longrightarrow x \doteq x \mid \varnothing$$

holds. The **R**-cut rule indicates that $f(y) \doteq z$ should be deleted, i.e.,

$$x \doteq x \mid \Gamma \Longrightarrow x \doteq x \mid \{f(x) \doteq y,\ \neg(f(f(x)) \doteq z)\}$$

holds, so $\{f(x) \doteq y, \neg(f(f(x)) \doteq z)\}$ is a maximal subset of Γ that is consistent with $x \doteq x$. The other two maximal consistent subsets

$$\{f(y) \doteq z, \neg(f(f(x)) \doteq z)\} \text{ and } \{f(x) \doteq y, f(y) \doteq z\}$$

can be deduced similarly.

In the beginning of this chapter, we explained that the purpose of establishing **R**-calculus is: for a given inconsistent **R**-configuration $\Delta \mid \Gamma$, to delete the sentences in Γ that are inconsistent with Δ. Example 7.10 shows that **R**-calculus can accomplish much more. This is the reason why the **R**-calculus we have defined is for **R**-configurations instead of inconsistent **R**-configurations only. In fact, Theorem 7.2 on **R**-reachability of inconsistent **R**-configurations can be generalized to the following form on **R**-configurations.

Theorem 7.5 (Basic theorem of testing). *Let Δ be an arbitrary formal theory consisting of finitely many atomic sentences or negations of atomic sentences, and Γ be an arbitrary finite formula set. If Γ' is an arbitrary maximal subset of Γ that is consistent with Δ, then there exists an* **R***-transition sequence*

$$\Delta \mid \Gamma \Longrightarrow^* \Delta \mid \Gamma'$$

that is provable.

[2] This example was created by Yuping Zhang. His purpose was to give a counterexample for **R**-calculus. But it turned out to be an inspiration for the basic theorem of testing.

Proof. Let $\Delta = \{A_1, A_2, \ldots, A_n\}$ and $\Gamma'' = \Gamma - \Gamma'$. In what follows we prove that for every $B \in \Gamma''$, we can use **R**-calculus to delete B. First, let $\Gamma_1 = \Gamma'' - \{B\}$ and $\Gamma_2 = \Gamma'$. Since $\Gamma_1, B \vdash B$ is provable, by Definition 7.2, B is an antecedent of B. Then $B \in \Gamma$ indicates that $B \mapsto B$. Since Γ_2 is a maximal subset of Γ that is consistent with Δ, $\Gamma_2 \cup \{B\}$ is inconsistent with Δ. According to (2) of Lemma 3.7, $\Gamma_2, B, \Delta \vdash \neg(A_1 \wedge \cdots \wedge A_n)$ is provable. Invoking the $\wedge$-*L* rule and $\neg$-*R* rule on the formulas in Δ, we can obtain that $\Gamma_2, B \vdash \neg(A_1 \wedge \cdots \wedge A_n)$ is provable and B is the requisite lemma in the proof. The **R**-axiom rule further indicates that

$$\Delta \mid \neg(A_1 \wedge \cdots \wedge A_n), \Gamma_2 \Longrightarrow \Delta \mid \Gamma_2$$

is provable. Hence all the conditions in the numerator of the **R**-cut rule-II are satisfied and thus

$$\Delta \mid \Gamma_1, B, \Gamma_2 \Longrightarrow \Delta \mid \Gamma_1, \Gamma_2$$

is provable, that is, B is deleted. We can delete every element in Γ'' in the same way. Thus we obtain an **R**-transition sequence

$$\Delta \mid \Gamma \Longrightarrow^* \Delta \mid \Gamma'$$

such that every **R**-transition in the sequence is provable, i.e., the **R**-transition sequence is provable. □

The proof of the basic theorem of testing provides a theoretical framework for formal revision of complex systems. In the development of complex systems, it is impractical to ensure the consistency of a version Γ. Instead, the revisions are made when testing shows a need for change. In software development, this change is usually realized through the debugging process.

Testing checks whether a system satisfies the requirements and whether it is consistent. If the system fails the tests, then a revision is required. Expressed in a first-order language, the testing results can be regarded as a formal theory Δ consisting of finitely many atomic sentences or negations of atomic sentences.

Hence the revisions can be accomplished by deleting the sentences in the current version, which are inconsistent with Δ, by **R**-calculus. The basic theorem of testing shows that such revisions might be accomplished by software tools developed on the basis of **R**-calculus. The correctness of the formal method of system revision is ensured by its reachability, soundness and completeness.

In 1985 Gärdenfors and his collaborators introduced the concept of changeability for formal theories and defined three different forms of changes: expansion, contraction and revision [AGM, 1985]. These are all proof-theoretic concepts. In particular, the **N**-expansion and **R**-contraction, introduced in this chapter, have a purpose similar to but more specific than those introduced in [AGM, 1985]. The essential novelty of our work is that we have developed a formal inference system, which is able to mechanically deduce the **R**-contractions.

In summary, this chapter has accomplished the following:

Firstly, at any point in time, a theory is tested and challenged by facts obtained from experiment. The existing theory (the current version), to some extent, can be described by a formal theory Γ of a first-order language $\mathscr{L}$. The facts can also be described by a formal theory Δ, which consists of atomic formulas and the negations of atomic formulas, because the information acquired from experiments in modern days is digitalized and can be represented by equations, predicates or their negation.

In general, they are consistent with each other. Thus, each step of a scientific discovery can be described formally by the **R**-configuration $\Delta \mid \Gamma$.

Secondly, if the **R**-configurations Δ and Γ are inconsistent with each other, then Δ is said to be an **R**-refutation with respect to Γ and $\Delta \mid \Gamma$ is called an inconsistent **R**-configuration. In this case, the relation between Δ and Γ is interpreted to mean that some logical consequences deduced from the existing theory Γ contradict the facts Δ supported by experiments. In this circumstance, we say that the existing theory Γ has met a refutation by facts Δ. This means:

Inconsistent R-configurations lead to scientific discoveries.

The goal of defining **R**-calculus is to deduce every formal theory Λ, which is a maximal subset of Γ and is consistent with Δ. Λ is called an **R**-contraction of Γ with respect to Δ.

The process of **R**-contraction follows Occam's Razor, which can be interpreted here as the following: "delete only those formulas from Γ which lead to inconsistency with Δ." This makes the least possible change to Γ, subject to consistency with the facts.

Thirdly, we formalize the actions of deleting the principles of the existing theory that are inconsistent with Δ, by **R**-rules. Each rule of **R**-calculus is expressed by a fraction. We have proved that **R**-calculus is sound, complete and reachable even though the existing theory itself is not consistent.

Finally, the main difference between the **G** system and **R**-calculus is the following: The **G** system is used to generate sequents such as $\Gamma \vdash A$, where Γ, in general, is a formal theory, which describes an axiom system in a specific domain and is the only premise for deductions. The purpose of applying **G**-rules is to deduce all logical consequences of the theory Γ.

In contrast, the **R**-calculus is used to revise a theory whose consequences have been refuted by experiment. In this case, we have two premises Δ and Γ. Δ is the set of experimental facts, which are used to revise the existing theory. Γ is a formal theory of our existing beliefs and may contain mistakes. When a logical consequence of Γ is inconsistent with Δ, the **R**-calculus is applied with the goal to delete formulas that are inconsistent with Δ, and to find all maximal subsets of Γ that are consistent with Δ.

The **G** system is invented for mathematicians to construct correct proofs in mathematical research. Whereas the **R**-calculus is invented for scientists to create new theories in the process of scientific discovery.

Chapter 8

Version Sequences

Scientific research is always carried out in the context of a specific methodology or strategy, whether this is conscious or not. These methodologies guide the generation of sequences of versions in the axiomatization process. They directly effect the success and quality of the research.

A research methodology usually specifies the workflow of the research and the tasks in each phase. It is a kind of programme for the research. The proscheme introduced in Chapter 6 can be used to describe simple research methodologies. The advantage of a proscheme consists of being able to use statements to describe the workflow of research methodologies and to use the concepts and methods of first-order languages to describe the process of axiomatization.

For this idea to be applied to a specific problem $\wp$ that is the subject of the research, we need to make the following basic assumptions:

1. The natural phenomena and the scientific experiments related to this problem can be explicitly observed.

2. The results of experiments and observations are measured in the form of data.

3. The problem can be described by means of propositions and the truth values of the propositions are determined by the observed data.

This chapter introduces a proscheme, called *OPEN*, which abstractly specifies actual research methodology. Using the proscheme OPEN, in this chapter we will introduce the fundamental properties that an ideal proscheme must possess.

The basic workflow of OPEN is as follows.

(1) Formulate a set of initial conjectures as a solution to the problem $\wp$. These conjectures form an axiom system, which is expressed as a formal theory Γ of a first-order language. Γ is our initial version of the axiom system for the problem.

(2) A new version can be generated as follows. We will treat the propositions of problem $\wp$ differently, according to the logical relations between each proposition and the current version Γ_n. As each proposition can be described by a sentence A of the first-order language, this amounts to verifying whether A is a formal consequence of Γ_n. There are four situations as explained below.

(a) $\Gamma_n \vdash A$ is provable and A is in accordance with the observed results and experimental data. In this case, the current version remains unchanged and we say that Γ_n *interprets rationally* the observed phenomena.

(b) $\Gamma_n \vdash A$ is provable and the interpretation of A predicts some phenomena that have not yet been observed. As a result, experiments have been performed, which confirm

the prediction. In this case, we say that the current version of the theory *predicts* the new phenomena and the current version remains unchanged.

(c) $\Gamma_n \vdash \neg A$ is provable, while the results of observation support A. In this case, the current version is refuted by the fact A and Γ_n needs to be revised. A new version Γ_{n+1} will be generated. More specifically, we take the maximal contraction of Γ_n with respect to A to obtain a new version Γ'. We then add A as a new conjecture to Γ' to obtain Γ_{n+1}.

(d) Both $\Gamma_n \vdash A$ and $\Gamma_n \vdash \neg A$ are unprovable. In this case, the one of A or $\neg A$ that accords with the results of experiments is added as a new conjecture to Γ_n to obtain Γ_{n+1}.

(3) For each proposition in $\wp$, we repeat the operations specified in (a)–(d) to generate new versions of the theory.

(4) A version sequence is formed by the versions in the order they are generated.

The second objective of this chapter is to describe criteria for evaluating research methodologies in first-order languages. We give an outline of these criteria as follows:

Since many research methodologies can be described by proschemes, the criteria can be characterized by means of the following properties of the output version sequences of proschemes.

(1) **Convergence of sequence.** A proscheme is reliable if, under its guidance, one can find or approach the truth about the problem being considered. The convergence of a proscheme means that the output sequence is convergent and its limit is the set of all the true propositions about the problem. If the output version sequence of the proscheme cannot approach the set of all the true propositions about the problem, then this proscheme is not a reliable one.

(2) **Commutativity between the limit operation and formal inference.** The commutativity between limit operation and formal inference means that the limit of the sequence of theory closures of versions is the same as the theory closure of the limit of the version sequence.

Most scientific research deals with finite axiom systems. Commutativity means that, in each phase of the axiomatization process, the output versions of a proscheme can be finite formal theories, which guarantees the operability of the versions. It ensures that formal inference does not affect the limit of a version sequence. Only those proschemes that possess commutativity are reliable ones.

(3) **Independence of sequence.** From the viewpoint of mathematical aesthetics, those axiom systems that possess independence are ideal. If the output sequence possesses independence, it means that, in each phase of the axiomatization process, the output versions of the proscheme all possess independence, i.e., the axioms contained in each version are independent of each other. In this case, the limit of the output version sequence also possesses independence.

This chapter takes the proscheme `OPEN` as an example and proves its convergence, commutativity, and independence. The process of axiomatization and version sequences will be discussed in Section 8.1. The proscheme `OPEN` will be defined in Section 8.2. It will be proved in Section 8.3 that the output version sequence of `OPEN` possesses convergence and in Section 8.4 that the output version sequence also possesses commutativity. The

independence of the output version sequence of OPEN will be addressed in Section 8.5. A formal definition of ideal proschemes will be given in Section 8.6.

8.1 Versions and version sequences

We saw in Chapter 6 how the knowledge of a domain could be axiomatized with an evolving sequence of theories. This process can be described by a version sequence. In this section we introduce the concepts of version and version sequence of a formal theory.

Definition 8.1 (Version of a formal theory). If Γ is a formal theory and A is a sentence, then according to the logical relationship between A and Γ, there are three kinds of versions of Γ with respect to A as follows.

(1) If A is a formal consequence of Γ, then we call Γ itself an *E-type version* of Γ with respect to A.

(2) If A is a new axiom of Γ, then we call $\Gamma \cup \{A\}$ an *N-type version* of Γ with respect to A.

(3) If A is a formal refutation of Γ, i.e., $\Gamma \vdash \neg A$ is provable, then we call any maximal contraction of Γ with respect to A an *R-type version* of Γ with respect to A.

We call the formal theory Γ' a *version* of Γ with respect to A if Γ' is an E-type, N-type, or R-type version of Γ with respect to A.

Definition 8.2 (Version sequence). We call a sequence of formal theories $\Gamma_1, \Gamma_2, \ldots, \Gamma_n, \ldots$ a *version sequence* if for every $i \geqslant 1$, Γ_{i+1} is a new version of Γ_i. Γ_1 is called the *initial theory* and Γ_i is called the *i-th version* of Γ_1.

In software development, for instance, we often call Windows 3.1 *version 3.1* of Windows. Here the second Windows refers to the version sequence of Windows. Sometimes we adopt this convention in this book and call Γ_i the *i-th version* of Γ.

Lemma 8.1 (Monotonic and non-monotonic version sequences). (1) *A version sequence $\{\Gamma_n\}$ is an increasing sequence if and only if for every $n \geqslant 1$, Γ_{n+1} is an N-type or E-type version of Γ_n.*

(2) *A version sequence $\{\Gamma_n\}$ is a decreasing sequence if and only if for every $n \geqslant 1$, Γ_{n+1} is an R-type or E-type version of Γ_n.*

(3) *A version sequence $\{\Gamma_n\}$ is a non-monotonic sequence if and only if the sequence is neither increasing nor decreasing.*

Proof. The conclusions readily follow from the definitions. □

According to this lemma, the Lindenbaum sequence, resolvent sequence, default sequence and sequence of T-generic sets introduced in Chapter 6 are all increasing version sequences.

8.2 The Proscheme OPEN

In this section we formally define the proscheme OPEN. We need to make the following three assumptions on the scientific problem $\wp$ to be investigated.

(1) Every proposition about the problem $\wp$ can be described by a mathematical proposition in the domain $M_\wp$.

(2) Every constant, variable, function, or relation in such propositions can be described by using a first-order language $\mathscr{L}_\wp$.

(3) There exists an interpretation map $I_\wp$ between $\mathscr{L}_\wp$ and the domain $M_\wp$ such that $(M_\wp, I_\wp)$ is a model of the first-order language $\mathscr{L}_\wp$.

We define scientific problems as follows.

Definition 8.3 ($\mathscr{L}_\wp$, $\mathbf{M}_\wp$ and $Th(\mathbf{M}_\wp)$). Let $\wp$ denote a scientific problem. $\mathscr{L}_\wp$ is a first-order language for $\wp$ consisting of the set of constant symbols, the set of function symbols and the set of predicate symbols that describe $\wp$. These sets can be the empty set, finite sets, or countably infinite sets. We call the model $\mathbf{M}_\wp$ of $\mathscr{L}_\wp$ a *scientific problem*. $Th(\mathbf{M}_\wp)$ is the set of sentences of the first-order language $\mathscr{L}_\wp$ whose interpretations in $\mathbf{M}_\wp$ are true.

Henceforth we often abbreviate a scientific problem $\mathbf{M}_\wp$ to problem $\mathbf{M}$. $Th(\mathbf{M})$ is a countable set of sentences whose elements are interpreted as true propositions in $\mathbf{M}$, that is, they are all supported by the experimental data. In scientific research, such propositions are usually not discovered the same time. In order to describe the chronological order of their discovery, we use the countable sequence of sentences $\{A_n\}$ of $\mathscr{L}_\wp$ to denote all the sentences in $Th(\mathbf{M})$. [1]

In this section we introduce the proscheme OPEN in the following way: we first use terms such as version and version sequence to describe the workflow of a research process; then we introduce the proscheme that implements this workflow.

(1) According to the above definition, since the interpretation of each A_i in $\mathbf{M}$ is true, each A_i is a criterion to determine whether a version of a formal theory should be accepted.

(2) The formal theory Γ is the initial conjectured solution to the problem $\mathbf{M}$. If Γ is true, then it is a proper subset of $Th(\mathbf{M})$. Otherwise it contains sentences inconsistent with $Th(\mathbf{M})$.

(3) The proscheme OPEN takes Γ, the initial formal theory, as input. It then takes, one by one, the elements A_n in the sequence $\{A_n\}$ as inputs and revises Γ_n accordingly. A new version Γ_{n+1} is thus generated and output. The inputs of the proscheme OPEN are Γ and the sequence $\{A_n\}$ and the outputs of OPEN form a version sequence.

(4) When the proscheme OPEN takes A_n as input, it outputs a new version Γ_{n+1} according to the logical relationship between A_n and the current version Γ_n. There are three different situations.

[1] Recall that A_n refers to an equivalence class of sentences; it is easy to see this makes the ordering well-defined.

(a) If $\Gamma_n \vdash A_n$ is provable, then $\Gamma_{n+1} := \Gamma_n$, i.e., Γ_{n+1} is an E-type version of Γ_n.

(b) If A_n is a new axiom of Γ_n, then $\Gamma_{n+1} := \Gamma_n \cup \{A_n\}$, i.e., Γ_{n+1} is an N-type version of Γ_n with respect to A_n.

(c) If $\Gamma_n \vdash \neg A_n$ is provable, then Γ_n is refuted by facts with respect to A_n. Since A_n is the nth element of $Th(\mathbf{M})$, it has to be accepted. In this case Γ_{n+1} is generated in two steps:
   i. first take an R-version Λ of Γ_n with respect to A_n, that is, Λ is a maximal subset of Γ_n that is consistent with A_n;
   ii. then expand Λ by adding the new axiom A_n to obtain $\Gamma_{n+1} := \Lambda \cup \{A_n\}$.

The examples in Chapter 7 show that the maximal contraction of Γ_n with respect to its formal refutation A is not unique. Thus for a given version Γ_n and its formal refutation A, the R-type version of Γ_n is also not unique. Hence there may be several choices for Γ_{n+1}. In this case, choosing an R-type version arbitrarily may not ensure that the output version sequence converges to $Th(\mathbf{M})$. To guarantee this convergence, we need to select an R-type version Λ satisfying the following conditions.

(1) Λ should contain all the new axioms already accepted by every version before the nth version, because these new axioms are true in $\mathbf{M}$. Hence in the proscheme OPEN we need to construct a sentence set Δ to store all the new axioms accepted by the first n versions. Thus when selecting the R-type version Λ, we have to ensure that Λ contains Δ.

(2) Even if an R-type version Λ contains Δ, it is still possible that it loses information during the axiomatization process. For instance, let $\Gamma = \{A \wedge B\}$. Then both $\Gamma \vdash A$ and $\Gamma \vdash B$ are provable. Suppose that A is refuted by facts with respect to $\mathbf{M}$. In this case the maximal subset of Γ that is consistent with $\neg A$ is the empty set. Hence when an R-type version of Γ with respect to $\neg A$ is generated, the sentence B is a formal consequence of Γ that is not refuted by $\neg A$ and thus should be retained. However, the sentence B was deleted together with $A \wedge B$.

To avoid losing B, we need to introduce another sentence set Θ when designing the proscheme OPEN. The sentences that are stored in Θ should have the following properties: each of them is a formal consequence, A_m, $m < n$, of some version within the first n versions; as an element of $\{A_i\}$, A_m was once the input of the proscheme OPEN. Thus after the proscheme OPEN selects an R-type version Λ satisfying condition (1), it needs to examine if each A_m contained in Θ is also contained in $Th(\Lambda)$. If not, then this sentence would be lost in revision and should be put into Γ_{n+1}. Since Θ only contains finitely many elements, such examination always halts.

Henceforth we call a maximal contraction that satisfies the above conditions an *acceptable contraction*. Of course, the acceptable contractions are not unique either. Hence, for the given problem $\mathbf{M}$ and initial version Γ, there may be many version sequences that

are output by the proscheme OPEN. They form an (infinite) tree structure with the initial theory Γ as its root. Each branch of the tree is a version sequence output by the proscheme.

In the following, we define the proscheme OPEN, where $R(\Gamma_n, A_n)$ is a maximal contraction of Γ_n with respect to A_n, and $(\Gamma_n - R(\Gamma_n, A_n)) \cap (\Delta_n \cup \Theta_n) = \varnothing$.

Definition 8.4 (Proscheme OPEN).

proscheme OPEN(Γ: theory; $\{A_n\}$: formula sequence)
Γ_n: theory;
Θ_n, Θ_{n+1}: theory;
Δ_n, Δ_{n+1}: theory;
proscheme OPEN*(Γ_n: theory; A_n: formula; **var** Γ_{n+1}: theory)
begin
  if $\Gamma_n \vdash A_n$ **then**
    begin
      $\Gamma_{n+1} := \Gamma_n$;
      $\Theta_{n+1} := \Theta_n \cup \{A_n\}$;
      $\Delta_{n+1} := \Delta_n$
    end
  else if $\Gamma_n \vdash \neg A_n$ **then**
    begin
      $\Gamma_{n+1} := R(\Gamma_n, A_n)$;
      $\Gamma_{n+1} := \Gamma_{n+1} \cup \{A_n\}$;
      loop until (**for every** $B_i \in \Delta_n \cup \Theta_n$, $\Gamma_{n+1} \vdash B_i$)
        loop for every $B_i \in \Delta_n \cup \Theta_n$
          if $\Gamma_{n+1} \vdash B_i$ **then skip**
          else if $\Gamma_{n+1} \vdash \neg B_i$ **then**
            $\Gamma_{n+1} := R(\Gamma_{n+1}, B_i)$;
            $\Gamma_{n+1} := \Gamma_{n+1} \cup \{B_i\}$
          else $\Gamma_{n+1} := \Gamma_{n+1} \cup \{B_i\}$
        end loop
      end loop
      $\Theta_{n+1} := \Theta_n$;
      $\Delta_{n+1} := \Delta_n \cup \{A_n\}$
    end
  else
    $\Gamma_{n+1} := \Gamma_n \cup \{A_n\}$;
    $\Theta_{n+1} := \Theta_n$;
    $\Delta_{n+1} := \Delta_n \cup \{A_n\}$
end
begin
  $n := 1$;
  $\Gamma_n := \Gamma$;
  $\Theta_n := \varnothing$; $\Theta_{n+1} := \varnothing$;
  $\Delta_n := \varnothing$; $\Delta_{n+1} := \varnothing$;

loop
OPEN*$(\Gamma_n, A_n, \Gamma_{n+1})$;
print Γ_{n+1};
$n := n+1$
end loop
end

Θ_n, Θ_{n+1} and Δ_n, Δ_{n+1} are all subsets of $Th(\mathbf{M})$ and thus they share the type theory.

Example 8.1 (Managing Θ_n).[2] Let $\Gamma = \{C, C \to A, \neg A \vee \neg B\}$ and $\{A_n\} = \{A, B, \ldots\}$. Since $\Gamma \vdash A$ is provable, we have

$$\Gamma_1 = \Gamma, \quad \Delta_1 = \varnothing, \quad \Theta_1 = \{A\}.$$

Since $\Gamma_1 \vdash \neg B$ and $\Delta_1 = \varnothing$, we can take $R(\Gamma_1, B) = \{C, \neg A \vee \neg B\}$. Thus

$$\Gamma_2' = \{C, \neg A \vee \neg B\} \cup \{B\}.$$

Since $A \in \Theta_1$ and $\Gamma_2' \vdash \neg A$ is provable, we need to make a contraction on Γ_2' according to the refutation by facts A and take $R(\Gamma_2', A) = \{C, B\}$. In this way we have

$$\Gamma_2 = \{C, B\} \cup \{A\}, \quad \Delta_2 = \{B\}, \quad \Theta_2 = \Theta_1.$$

This example shows that in OPEN*, when we examine whether Θ_n loses sentences after the contraction, there are three possibilities:

1. $\Gamma_{n+1} \vdash B_i$ is provable and thus B_i is not lost.
2. $\Gamma_{n+1} \vdash \neg B_i$ is provable, and in this case we need to find the maximal contraction of Γ_{n+1} with respect to B_i.
3. Neither $\Gamma_{n+1} \vdash B_i$ nor $\Gamma_{n+1} \vdash \neg B_i$ is provable, and so we need to add B_i as a new axiom into Γ_{n+1}.

Example 8.2. Let $\Gamma = \{D, D \to A, E, E \to B, \neg A \vee \neg B \vee \neg C\}$ and $\{A_n\} = \{A, B, C, \ldots\}$. Since $\Gamma \vdash A$ is provable, we have

$$\Gamma_1 = \Gamma, \quad \Delta_1 = \varnothing, \quad \Theta_1 = \{A\}.$$

Since $\Gamma_1 \vdash B$ is provable, we have

$$\Gamma_2 = \Gamma, \quad \Delta_2 = \varnothing, \quad \Theta_2 = \{A, B\}.$$

Since $\Gamma_2 \vdash \neg C$ is provable and $\Delta_2 = \varnothing$, we can take $R(\Gamma_2, C) = \{D, D \to A, E, \neg A \vee \neg B \vee \neg C\}$ and thus

$$\Gamma_3' = \{D, D \to A, E, \neg A \vee \neg B \vee \neg C\} \cup \{C\}.$$

[2]Examples 8.1 and 8.2 are provided by Jie Luo and Shengming Ma.

Since $A \in \Theta_2$ and $\Gamma'_3 \vdash A$ is provable, it is unnecessary to retrieve A. Since $B \in \Theta_2$ and $\Gamma'_3 \vdash \neg B$ is provable, we take $R(\Gamma'_3, B) = \{D, E, \neg A \vee \neg B \vee \neg C, C\}$ and thus

$$\Gamma''_3 = \{D, E, \neg A \vee \neg B \vee \neg C, C\} \cup \{B\}.$$

Now $\Gamma''_3 \vdash \neg A$ is provable. Hence we can make another contraction on Γ''_3 according to the refutation by facts A. If we take $R(\Gamma''_3, A) = \{D, E, C, B\}$, then

$$\Gamma_3 = \{D, E, C, B\} \cup \{A\}, \quad \Delta_3 = \{C\}, \quad \Theta_3 = \{A, B\}.$$

This example shows that when we examine whether the sentences in Δ_n and Θ_n are lost in the contractions, sometimes it is insufficient to examine Δ_n and Θ_n one by one only once. We need to examine them repeatedly to ensure that no sentences that should be accepted are lost.

8.3 Convergence of the proscheme

In this and the following two sections, we use the proscheme `OPEN` as an example to study three basic properties, convergence, commutativity and independence of a general proscheme. We first prove that `OPEN` is convergent.

Definition 8.5 (Output version sequence of the proscheme). Suppose that $\mathbf{M}$ is a scientific problem with $Th(\mathbf{M}) = \{A_n\}$. Let Γ be a formal theory and F be a proscheme. If Γ is the initial input and $\{A_n\}$ is the input sequence of F, then we call the version sequence $\{\Gamma_n\}$ output by F the *output version sequence* of the proscheme F with respect to the problem $\mathbf{M}$ and initial theory Γ.

Theorem 8.1 (Convergence of `OPEN`). *Suppose that $\wp$ is a scientific problem and $\mathscr{L}_\wp$ is a first-order language on $\wp$. Let $\mathbf{M}$ be an arbitrary model of $\mathscr{L}_\wp$ and Γ be a finite formal theory in $\mathscr{L}_\wp$. Then with respect to $\mathbf{M}$ and the initial theory Γ, every output version sequence $\{\Gamma_n\}$ of the proscheme `OPEN` converges. Further, the sequence $\{Th(\Gamma_n)\}$ of theory closures also converges and*

$$\lim_{n\to\infty} Th(\Gamma_n) = Th(\mathbf{M}).$$

Proof. We first prove that the output version sequence $\{\Gamma_n\}$ converges. In fact, since Γ is a finite formal theory, $\Gamma - Th(\mathbf{M})$ is also a finite formal theory. For every $B \in \Gamma - Th(\mathbf{M})$, $\neg B \in Th(\mathbf{M})$. Without loss of generality, suppose that $\neg B = A_n$. If $B \in \Gamma_n$, then $\neg B = A_n$ constitutes a refutation by facts of Γ_n. According to the proscheme `OPEN`, each maximal contraction of Γ_n with respect to $\neg B$ cannot contain B. Hence $B \notin \Gamma_{n+1}$.

Thus there is a natural number N such that $\Gamma_N \cap (\Gamma - Th(\mathbf{M})) = \varnothing$, and the output version sequence $\{\Gamma_n\}_{n>N}$ is an increasing sequence. As a result, the output version sequence $\{\Gamma_n\}$ converges.

We now prove that the sequence $\{Th(\Gamma_n)\}$ of theory closures of the output version sequence $\{\Gamma_n\}$ converges to $Th(\mathbf{M})$. The proof is done in two steps.

1. We first prove that $Th(\mathbf{M}) \subseteq \{Th(\Gamma_m)\}_*$. For every $A_i \in Th(\mathbf{M})$, since $Th(\mathbf{M})$ is consistent, according to the compactness theorem, there exists a finite subset

$$\Sigma_m = \{B_{m_1}, \ldots, B_{m_j}\}$$

of $Th(\mathbf{M})$ such that $\Sigma_m \vdash A_i$.

According to the definition of the proscheme OPEN, for every $B_{m_i} \in Th(\mathbf{M})$, there should exist some n_i such that either $B_{m_i} \in Th(\Gamma_{n_i})$ or $\neg B_{m_i} \in Th(\Gamma_{n_i})$. In either case we have $B_{m_i} \in Th(\Gamma_{n_i+1})$. And the constructions of Δ and Θ in the proscheme OPEN further ensure that for $n \geqslant n_i + 1$, $B_{m_i} \in Th(\Gamma_n)$. Let $N = \max\{n_1, \ldots, n_j\}$. When $n \geqslant N+1$, $A_i \in Th(\Gamma_n)$. Hence we have

$$A_i \in \bigcup_{n=1}^{\infty} \bigcap_{m=n}^{\infty} Th(\Gamma_m), \text{ i.e., } A_i \in \{Th(\Gamma_m)\}_*.$$

2. Next we prove by contradiction that $\{Th(\Gamma_m)\}^* \subseteq Th(\mathbf{M})$. Suppose that there exists a sentence A such that both $A \in \{Th(\Gamma_m)\}^*$ and $A \notin Th(\mathbf{M})$. There are only two possible situations as follows.

(a) Neither $Th(\mathbf{M}) \vdash A$ nor $Th(\mathbf{M}) \vdash \neg A$ is provable. This is possible only when Γ contains a sentence that is logically independent of $Th(\mathbf{M})$. This is impossible because $Th(\mathbf{M})$ is complete.

(b) $\neg A \in Th(\mathbf{M})$, which is also impossible. In fact, suppose that the opposite is true. Then according to the definition of the proscheme OPEN, there should exist an i such that A_i is $\neg A$ and hence there should exist an N such that $\neg A \in Th(\Gamma_N)$. Thus $\neg A \in Th(\Gamma_m)$ holds for $m > N$. Since $A \in \{Th(\Gamma_m)\}^*$, there exists an infinite subsequence $\{n_k\}$ such that $A \in Th(\Gamma_{n_k})$. Thus there should exist an $n_k > N$ such that $A \in Th(\Gamma_{n_k})$. Since $\neg A \in Th(\Gamma_{n_k})$, this contradicts the consistency of $Th(\Gamma_{n_k})$.

In summary,

$$\{Th(\Gamma_m)\}^* \subseteq Th(\mathbf{M}) \subseteq \{Th(\Gamma_m)\}_*.$$

Thus $\{Th(\Gamma_m)\}_* = \{Th(\Gamma_m)\}^* = Th(\mathbf{M})$. The theorem is proved. □

Theorem 8.1 can be interpreted as follows. Firstly, $Th(\mathbf{M})$ is the set of all the sentences of $\mathscr{L}$ that are true in $\mathbf{M}$. It contains all the essential characteristics of $\mathbf{M}$. Secondly, the functionality of the proscheme OPEN is to delete the defects in the initial conjecture Γ, i.e., the sentences that are false in $\mathbf{M}$, and then to add those sentences not in Γ that are true in $\mathbf{M}$. These operations are accomplished by generating new versions iteratively and the output version sequence converges to $Th(\mathbf{M})$.

The proscheme OPEN provides a mechanism for this by introducing two sets Θ and Δ. The set Δ is used to store new axioms that were accepted in previous versions. The set Θ is used to store the input sentences that are formal consequences of some previous version but are not accepted by OPEN directly. Only when Θ and Δ are used in the way prescribed by the proscheme OPEN can we ensure that the output version sequence converges to $Th(\mathbf{M})$.

Many people think that, so long as the mutual interactions between conjectures and refutations, or those between theories and experiments, are cyclic and repeat indefinitely, the entire truth of the problem can be gradually approximated. Theorem 8.1 indicates that, only by designing the proscheme carefully and introducing such mechanisms as Θ and Δ to regulate the maximal contraction, can the generated version sequences approximate to the entire truth of the problem.

8.4 Commutativity of the proscheme

The limit of a sequence of formal theories is formed from the unions and intersections of sentence sets, whereas the closure of a formal theory is deduced through formal inference. We might ask, what is the relationship between the theory closure of the limit of a sequence and the limit of a sequence of theory closures?

In this section we prove that they are identical for the proscheme `OPEN`. In other words, for `OPEN`, the limit operation is commutative with formal inference. For a given formal theory Γ, the theory closure $Th(\Gamma)$ is the set of formal consequences of Γ. Hence Th is a map between sets of formulas. The commutativity between the limit operation and formal inference means that Th is a continuous function.

In general, the limit operation and the formal inference of formal theory sequences are not commutative. Consider the following example.

Example 8.3. Suppose that A and A_n are mutually different sentences. Consider the sequence $\{\Sigma_n\}$ with

$$\Sigma_n = \{A_n, A_n \to A\},$$

where $n = 1, 2, \ldots$. It is not difficult to verify that both

$$\lim_{n\to\infty} \Sigma_n = \varnothing \quad \text{and} \quad \lim_{n\to\infty} Th(\Sigma_n) = Th(\{A\}).$$

This example indicates that for $\{\Sigma_n\}$, the limit operation and the formal inference are not commutative.

Let us invoke the proscheme `OPEN`. Suppose that the initial formal theory Γ being input is the empty set and the input sequence is

$$A_1, A_1 \to A,\ A_2, A_2 \to A,\ \ldots,\ A_n, A_n \to A,\ \ldots.$$

After the $(2n)$th cycle of manipulations of the proscheme `OPEN`, its output version is

$$\Gamma_{2n} = \bigcup_{m=1}^{n} \{A_m, A_m \to A\}.$$

Since $\{\Gamma_n\}$ is an increasing sequence, its limit is

$$\lim_{n\to\infty} \Gamma_n = \bigcup_{m=1}^{\infty} \{A_m, A_m \to A\}.$$

It is not difficult to verify that the output version sequence $\{\Gamma_n\}$ is commutative. This shows that commutativity is dependent on the proscheme used.

Theorem 8.2 (Commutativity of OPEN). *Suppose that $\wp$ is a scientific problem and $\mathscr{L}_\wp$ is a first-order language on $\wp$. Let* **M** *be an arbitrary model of $\mathscr{L}_\wp$ and Γ be a finite formal theory in $\mathscr{L}_\wp$. Then every version sequence $\{\Gamma_n\}$ generated by the proscheme* OPEN *with respect to* **M** *and Γ satisfies*

$$\lim_{n\to\infty} Th(\Gamma_n) = Th(\lim_{n\to\infty} \Gamma_n).$$

Proof. Suppose that the sequence $\{A_n\}$ is $Th(\mathbf{M})$. According to Theorem 8.1, every version sequence $\{\Gamma_n\}$ generated by the proscheme OPEN with respect to **M** and Γ is convergent, and

$$\{Th(\Gamma_n)\}_* = \{Th(\Gamma_n)\}^* = Th(\mathbf{M}).$$

Thus it suffices to prove that

$$\{Th(\Gamma_n)\}_* \subseteq Th(\{\Gamma_n\}_*) \subseteq Th(\{\Gamma_n\}^*) \subseteq \{Th(\Gamma_n)\}^*,$$

which can be done in the following two steps.

(1) We first prove that $Th(\{\Gamma_n\}^*) \subseteq \{Th(\Gamma_n)\}^*$.

For every $A \in Th(\{\Gamma_n\}^*)$, i.e., $\{\Gamma_n\}^* \vdash A$ is provable, according to the compactness theorem, there exists

$$\{A_{n_1}, \ldots, A_{n_k}\} \subseteq \{\Gamma_n\}^* \text{ such that } A_{n_1}, \ldots, A_{n_k} \vdash A$$

is provable. By the definition of $\{\Gamma_n\}^*$, $A_{n_i} \in \{\Gamma_n\}^*$, $i = 1, \ldots, k$, which implies that there exists a subsequence of $\{\Gamma_n\}$:

$$\Gamma_{n_{i_1}}, \ldots, \Gamma_{n_{i_j}}, \ldots, \text{ where } j \text{ is any natural number,}$$

such that A_{n_i} is an element of every $\Gamma_{n_{i_j}}$ in this sequence and thus is an element of $Th(\Gamma_{n_{i_j}})$. Hence $A_{n_i} \in \{Th(\Gamma_n)\}^*$, that is, $\{A_{n_1}, \ldots, A_{n_k}\} \subset \{Th(\Gamma_n)\}^*$. Then according to Theorem 8.1, $\{Th(\Gamma_n)\}^* = Th(\mathbf{M})$. $\{Th(\Gamma_n)\}^*$ is closed under formal inference and we have

$$A \in Th(\{A_{n_1}, \ldots, A_{n_k}\}) \subset \{Th(\Gamma_n)\}^*.$$

(2) Next we prove that $\{Th(\Gamma_n)\}_* \subseteq Th(\{\Gamma_n\}_*)$.

Let A be an arbitrary formula of $\mathscr{L}_\wp$. If $A \in \{Th(\Gamma_n)\}_*$, then $A \in Th(\mathbf{M})$ since $\{Th(\Gamma_n)\}_* = Th(\mathbf{M})$ according to Theorem 8.1. Hence there exists an N such that $A_N = A$. By the definition of the proscheme OPEN, there are only three possible cases to consider.

(a) A_N is a new axiom of Γ_N. By the definition of the proscheme OPEN, for every $n > N$, $A_N \in \Gamma_n$, that is, $A_N \in \{\Gamma_n\}_*$.

(b) A_N is a formal refutation of Γ_N. By the definition of the proscheme OPEN, we also have $A_N \in \Gamma_{N+1}$, and for $n > N$, $A_N \in \Gamma_n$. Thus we have $A_N \in \{\Gamma_n\}_*$ as well.

(c) A_N is a formal consequence of Γ_N. According to the compactness theorem, there exists

$$\{A_{n_1},\ldots,A_{n_k}\} \subseteq \Gamma_N \text{ such that } A_{n_1},\ldots,A_{n_k} \vdash A_N$$

is provable. By the definition of the proscheme OPEN, either $\{A_{n_1},\ldots,A_{n_k}\} \subset \Gamma_n$ holds for every $n > N$, or $A_N \in \Theta_N$ and there exists an $n_0 > N$ such that in generating Γ_{n_0}, A_N was "retrieved", that is, $A_N \in \Gamma_{n_0}$. Hence for every $n > n_0$, $A_N \in \Gamma_n$. In summary, $A_N \in Th(\{\Gamma_n\}_*)$.

Thus in any case we have $A \in Th(\{\Gamma_n\}_*)$. □

What does it mean when a proscheme is commutative in this way? To understand this, note that, in the axiomatizing process, one usually starts with a finite set of conjectures. In the process of evolving a theory through revisions, the revised axiom sets, Γ_n , remain finite. However, in general, $Th(\mathbf{M})$ contains infinitely many independent sentences.

Commutativity means that we can evolve a theory finitely by just considering its axioms. The limit of this sequence $\{\Gamma_n\}$ will have exactly the same consequences as if we took the sequence of theory closures $\{Th(\Gamma_n)\}$ and formed its limit. Theorem 8.2 says even more: that the complete theory $Th(\mathbf{M})$ can be generated from the limit of a sequence of finite axiom sets. More generally, for those proschemes that possess commutativity, it is feasible to approximate a problem **M** using versions containing a finite number of axioms.

8.5 Independence of the proscheme

We say an axiom system is independent if its axioms are mutually independent. Independence is an aesthetic criterion for evaluating the quality of theoretical research and for understanding the essential features of a theory. In this section we will investigate the independence of OPEN.

Lemma 8.2 (Independence of sequence limit). *If for every natural number n, Γ_n is an independent formal theory and $\{\Gamma_n\}$ is convergent, then*

$$\lim_{n\to\infty} \Gamma_n$$

is an independent formal theory as well.

Proof. It suffices to prove that $\{\Gamma_n\}_*$ is an independent formal theory. For every $A \in \{\Gamma_n\}_*$, there should exist an N such that for $n > N$, $A \in \Gamma_n$. Since Γ_n is an independent theory, $Th(\Gamma_n - \{A\}) \neq Th(\Gamma_n)$, i.e., $\Gamma_n - \{A\} \vdash A$ is unprovable. Since $A \in \Gamma_n$, $\Gamma_n \vdash A$ is provable and thus

$$\left(\bigcap_{n=N}^{\infty} (\Gamma_n - \{A\})\right) \vdash A \text{ is unprovable, but } \bigcap_{n=N}^{\infty} \Gamma_n \vdash A \text{ is provable.}$$

Hence $\{\Gamma_n\}_* - \{A\} \vdash A$ is unprovable, but $\{\Gamma_n\}_* \vdash A$ is provable. By definition, this is actually $Th(\{\Gamma_n\}_* - \{A\}) \neq Th(\{\Gamma_n\}_*)$. Thus $\{\Gamma_n\}_*$ is an independent theory. □

Neither an version in the output version sequence of OPEN nor the limit of the output version sequence of OPEN is guaranteed to be an independent theory, even if the initial theory Γ of OPEN is an independent theory. Let us examine the following example:

Example 8.4. Suppose that a first-order language $\mathscr{L}$ has a constant symbol set $\{a,b,c\}$ and only one unary predicate $P(x)$. Also suppose that the model of the problem is $\mathbf{M}$, whose set $Th(\mathbf{M})$ of true sentences is

$$P[a],P[b],P[c],\forall xP(x),\exists xP(x),\ldots.$$

Evidently, the independent theory with respect to $\mathbf{M}$ is $\{\forall xP(x)\}$.

(1) If the initial theory is $\Gamma=\varnothing$ and the input sequence is $Th(\mathbf{M})$, then the output version sequence of OPEN is

$$\Gamma_1=\{P[a]\},\ \Gamma_2=\{P[a],P[b]\},\ \Gamma_3=\{P[a],P[b],P[c]\},\ \Gamma_4=\{P[a],P[b],P[c],\forall xP(x)\}.$$

The limit of this sequence is $\{P[a],P[b],P[c],\forall xP(x)\}$.

(2) If the initial theory is $\Gamma=\{P[a]\}$ and the input sequence is $Th(\mathbf{M})$, then the output version sequence of OPEN is also

$$\Gamma_1=\{P[a]\},\ \Gamma_2=\{P[a],P[b]\},\ \Gamma_3=\{P[a],P[b],P[c]\},\ \Gamma_4=\{P[a],P[b],P[c],\forall xP(x)\}.$$

The limit of this sequence is $\{P[a],P[b],P[c],\forall xP(x)\}$.

(3) If the initial theory is $\Gamma=\{\forall xP(x)\}$ and the input sequence is $Th(\mathbf{M})$, then the output version sequence of OPEN is

$$\Gamma_1=\Gamma_2=\Gamma_3=\Gamma_4=\{\forall xP(x)\}.$$

The limit of this sequence is $\{\forall xP(x)\}$.

In the first two cases, the initial conjectures of the proscheme OPEN are both independent theories, whereas neither of the limits of the output version sequences $\{\Gamma_n\}$ is an independent theory. It is only in the third case that the limit of the output version sequence is an independent theory.

This example shows that the proscheme OPEN does not ensure the independence of the limit of the output version sequence. The reason is that, given Γ_n and a new input A_n, although neither $\Gamma_n\vdash A_n$ nor $\Gamma_n\vdash\neg A_n$ is provable, it is still possible that Γ_n contains formal consequences of A_n. For instance, in the first case in the above example, $\Gamma_3\vdash\forall xP(x)$ is unprovable but $P[a]$, $P[b]$ and $P[c]$ in Γ_3 are all formal consequences of $\forall xP(x)$.

We can improve the proscheme OPEN so that it ensures the independence of the limit of its output version sequence. Specifically, when neither $\Gamma_n\vdash A_n$ nor $\Gamma_n\vdash\neg A_n$ is provable or when a refutation by facts is added to the new version as a new axiom, we determine Γ_{n+1} in two steps as follows.

Suppose that $\Gamma_n=\{B_1,B_2,\ldots,B_{n_k}\}$. First, we examine the elements B_i in Γ_n one by one from 1 to n_k. If

$$(\Gamma_n-\{B_i\}),A_n\vdash B_i$$

is provable, then we let $\Gamma_n = \Gamma_n - \{B_i\}$. After n_k steps of such operations, we obtain a final Γ_n, which is independent of A_n. Next we let

$$\Gamma_{n+1} = \Gamma_n \cup \{A_n\}.$$

This improvement on the proscheme OPEN ensures that if Γ_n is an independent theory, then so is Γ_{n+1}. We can call the improved proscheme $OPEN^+$, then $\texttt{OPEN}^+$ is independent.

The improved proscheme $\texttt{OPEN}^+$ fits more closely with our expectations of a mathematical theory. In practice, independence of the axioms is not the first priority. Instead, when a new revision of a theory is proposed, later examination finds those axioms in the new version that are logical consequences of others and some axioms are deleted to make the axiom set independent. This is what happened with Kepler's laws after Newton's laws of motion and gravitation were added to physics. It is also exactly what $\texttt{OPEN}^+$ does. In this way each new version is further revised to make its axioms independent and thus the limit of the sequence is also independent. However, in practical terms $\texttt{OPEN}^+$ consumes more time and storage than OPEN.

Independence may be aesthetically pleasing and, for a scientific theory, may be useful in that it allows us to see what is fundamental in the theory. However, for information technology, this may not be so important because the priority here is to make computation efficient. In general, independence makes computation inefficient. For example, in the design of a CPU for a computer, it is only necessary to include the instructions for plus one, minus one and jump in order to implement the whole of arithmetic. However, this would be very slow and inefficient. So a real CPU contains no less than 100 instructions, simply on the grounds of speed. As another example, we showed in Chapter 4 that a programming language need only contain six statements to compute any decidable problem. However, it would be impractical to actually programme in such a language and real languages contain many more syntactic ingredients to make the writing of programs easier. Furthermore, various pre-written libraries are provided to reuse well-tested functions and to avoid reinventing the wheel.

So the process of designing software systems, knowledge bases and integrated circuits can be accomplished using a proscheme similar to OPEN, which is non-independent but more efficient.

8.6 Reliable proschemes

As mentioned above, all research follows some kind of methodology or paradigm, either consciously or unconsciously. The methodology determines the quality of research. For those research problems that can be embodied in a proscheme, we have shown that the proscheme should be convergent, commutative and, ideally, should ensure independence. A proscheme possessing these three properties can be called an *ideal research methodology*. In what follows, we give a more general definition for the convergence, commutativity and independence of proschemes.

Definition 8.6 (convergence). Suppose that $\mathscr{L}$ is a first-order language with **M** being an arbitrary model of $\mathscr{L}$. Let F be a proscheme. Suppose $\{A_n\}$ is a finite or countably infinite consistent input sequence of sentences. If for every finite formal theory Γ of $\mathscr{L}$, the output version sequence $\{\Gamma_n\}$ of F with respect to $\{A_n\}$ converges, and

$$\lim_{n\to\infty} Th(\Gamma_n) = Th(\mathbf{M}),$$

then we say that the proscheme F possesses *convergence*.

Corollary 8.1. *The proscheme* `OPEN` *possesses convergence.*

Proof. Let the input sequence be $\{A_n\} = Th(\mathbf{M})$. Then the corollary is proved by Theorem 8.1. □

Definition 8.7 (commutativity). Suppose that $\mathscr{L}$ is a first-order language with **M** being an arbitrary model of $\mathscr{L}$. Let F be a proscheme. Suppose $\{A_n\}$ is a finite or countably infinite consistent input sequence of sentences. If for every finite formal theory Γ of $\mathscr{L}$, the output version sequence $\{\Gamma_n\}$ of F with respect to $\{A_n\}$ converges, and

$$\lim_{n\to\infty} Th(\Gamma_n) = Th(\lim_{n\to\infty} \Gamma_n),$$

then we say that the proscheme F possesses *commutativity*.

Corollary 8.2. *The proscheme* `OPEN` *possesses commutativity.*

Proof. Let the input sequence be $\{A_n\} = Th(\mathbf{M})$. Then the corollary is proved by Theorem 8.2. □

Definition 8.8 (independence). Suppose that $\mathscr{L}$ is a first-order language with **M** being an arbitrary model of $\mathscr{L}$. Let F be a proscheme. Suppose $\{A_n\}$ is a finite or countably infinite consistent input sequence of sentences. If for every independent finite formal theory Γ of $\mathscr{L}$, the output version sequence $\{\Gamma_n\}$ of F with respect to $\{A_n\}$ converges, and every output version Γ_n of F is an independent theory, then we say that the proscheme F possesses *independence*.

Corollary 8.3. *The proscheme* `OPEN` *does not possess independence, but the proscheme* $\texttt{OPEN}^+$ *possesses independence.*

Proof. According to Section 8.5, the corollary is proved. □

From Theorems 8.1 and 8.2, we can deduce the following two theorems directly.

Theorem 8.3. *Suppose that* **M** *is a scientific problem,* $\{A_n\}$ *is a finite or countably infinite input sequence of the proscheme* `OPEN` *and is consistent, and* $Th(\{A_n\}) = Th(\mathbf{M})$. *Let* $\{\Gamma_n\}$ *be the output version sequence of the proscheme* `OPEN` *with respect to* $\{A_n\}$ *and the initial theory* Γ. *Then* $\{\Gamma_n\}$ *is convergent and*

$$\lim_{n\to\infty} Th(\Gamma_n) = Th(\mathbf{M}).$$

Proof. Let the initial formal theory of the proscheme OPEN be $\Gamma = \{B_1, \ldots, B_k\}$. According to the construction of the proscheme OPEN and the compactness theorem, i.e., Theorem 3.2, there exists a big enough $N > 0$ such that after the Nth execution cycle of OPEN*, for every $n > N$, we have

$$Th(\{A_1, \ldots, A_n\}) \subseteq Th(\Gamma_{n+1}) \subseteq Th(\{A_n\}).$$

By definition, since $\lim_{n\to\infty} Th(\{A_1, \ldots, A_n\}) = Th(\{A_n\})$, we have

$$Th(\{A_n\}) \subseteq \{Th(\Gamma_n)\}_* \subseteq \{Th(\Gamma_n)\}^* \subseteq Th(\{A_n\}).$$

Further, since $Th(\{A_n\}) = Th(\mathbf{M})$, $\{Th(\Gamma_n)\}_* = \{Th(\Gamma_n)\}^* = Th(\mathbf{M})$ holds. The theorem is proved. □

Theorem 8.4. *Suppose that* $\mathbf{M}$ *is a scientific problem,* $\{A_n\}$ *is a finite or countably infinite input sequence of the proscheme* OPEN *and is consistent, and* $Th(\{A_n\}) = Th(\mathbf{M})$. *Let* $\{\Gamma_n\}$ *be the output version sequence of the proscheme* OPEN *with respect to* $\{A_n\}$ *and the initial theory* Γ. *Then* $\{\Gamma_n\}$ *is convergent and*

$$\lim_{n\to\infty} Th(\Gamma_n) = Th(\lim_{n\to\infty} \Gamma_n).$$

Proof. The proof is similar to that of Theorem 8.2. □

We can now define reliable proschemes and ideal proschemes.

Definition 8.9 (Reliable proscheme). We say that the proscheme F is *reliable* if it possesses convergence and commutativity, and that it is *ideal* if it is reliable and also possesses independence.

Summarizing the proofs and discussions in the previous sections of this chapter we have the following.

Theorem 8.5. *Suppose that* $\mathscr{L}$ *is a first-order language with* $\mathbf{M}$ *being an arbitrary model of* $\mathscr{L}$. *Let* $\{A_n\}$ *be a finite or countably infinite input sequence of sentences of the proscheme* OPEN. *If* $\{A_n\}$ *is consistent and satisfies* $Th(\{A_n\}) = Th(\mathbf{M})$, *then* OPEN *is a reliable proscheme. Under the above conditions* OPEN^+ *is an ideal proscheme.*

Proof. According to Theorems 8.3 and 8.4 and Corollary 8.3, the conclusion is immediate. □

Compared with Theorem 8.3, Theorem 8.1 is almost trivial. The reason is that Theorem 8.1 requires the input sequence $\{A_n\}$ to be the same as $Th(\mathbf{M})$. Since the input initial formal theory Γ is a finite formal theory, according to the construction of the proscheme OPEN, this amounts to deleting all the sentences in Γ inconsistent with $Th(\mathbf{M})$ after finitely many steps of execution and hence accepting all the sentences in $Th(\mathbf{M})$ during the execution of the proscheme OPEN.

In contrast, Theorem 8.3 does not require inputting all of the $Th(\mathbf{M})$. It shows that it is sufficient to input a sequence $\{A_n\}$ satisfying $Th(\{A_n\}) = Th(\mathbf{M})$. The sequence

$\{A_n\}$ can be either finite or countably infinite. Thus Theorem 8.3 is more significant than Theorem 8.1. The limitation of both theorems is that, in real life, for the proscheme OPEN, it is usually difficult to specify an input sequence $\{A_n\}$ that satisfies $Th(\{A_n\}) = Th(\mathbf{M})$.

We should also point out that all the theorems in this chapter require that the initial formal theory Γ is finite. In fact, if Γ is a countably infinite formal theory, these theorems still hold.

For instance, to prove that Theorem 8.1 still holds if Γ is countably infinite, we can construct a new proscheme OPEN$^\sharp$ on the basis of the proscheme OPEN.

The proscheme OPEN$^\sharp$ has two countably infinite input sequences. One input sequence is $\Gamma = \{B_m\}$, whereas the other is $\{A_n\} = Th(\mathbf{M})$.

The workflow of OPEN$^\sharp$ is as follows:

1. The proscheme inputs A_n one by one. It begins by taking A_1 and an initial theory $\Gamma_0 := \{B_1, \ldots, B_N\}$, for some $N > 0$. It calls the proscheme OPEN$^*(\Gamma_0, A_1, \Gamma_1)$ to obtain Γ_1.
2. The proscheme also inputs $B_m \in \Gamma - \Gamma_0$ one by one, starting from B_{N+1}. It begins by generating a new revision Γ_2 according to the relationship between Γ_1 and B_{N+1}:
   (a) If $\Gamma_1 \vdash B_{N+1}$ is provable, then let $\Gamma_2 := \Gamma_1$.
   (b) If $\Gamma_1 \vdash \neg B_{N+1}$ is provable, then let $\Gamma_2 := \Gamma_1$.
   (c) If neither $\Gamma_1 \vdash B_{N+1}$ nor $\Gamma_1 \vdash \neg B_{N+1}$ is provable, then let $\Gamma_2 := \Gamma_1 \cup \{B_{N+1}\}$.
3. Next it takes A_2, Γ_2 and B_{N+2} as inputs and repeats the above workflow.

OPEN$^\sharp$ can also be written in the form of proscheme. We can use a similar method to prove that OPEN$^\sharp$ is a reliable proscheme.

Chapter 9

Inductive Inference

Induction has been studied for more than two thousand years, starting with Aristotle. Many philosophers have made important contributions on the subject, such as Bacon, Mill, Hume, Herschel, Poincaré, Peirce, Reichenbach, Carnap and Popper. The Chinese logician Mo [1993] has also made a profound study of its subtleties. Before exploring induction theoretically, we will give an overview of the relevant concepts.

Conjecture, induction, and inductive inference. As we saw in Chapter 6 and Chapter 8, new conjectures are the means by which we refine and expand an axiom system, thus evolving our description of a domain. Forming a conjecture is a sophisticated process and is not necessarily rational. It may simply be a belief. However, in this chapter, we restrict ourselves to 'rational conjectures'. If we do this then we can define symbolic rules to describe the process.

Induction is a kind of rational conjecture. For example, the philosopher Hume described seeing a flying bird in a nature reserve, which was a white swan named 'White'. Here "bird," "white," and "can fly" are specific attributes that he observed were true of the swan White. He might have induced from them that *every swan is a bird*, *every swan is white*, and *every swan can fly*. These three propositions are all general conjectures about swans. As Aristotle said in his great work, *The Organon*, "induction is a passage from particulars to universals [McKeon, 1941]."

Inductive inference is a mechanism of induction. In this chapter, inductive inference refers to using the symbols of first-order languages to describe objects, properties, and universal laws, establishing rules of calculus for logical connective symbols and quantifier symbols, and then using these rules to describe the *passage from particulars to universals*.

For instance, let $\mathscr{L}$ denote the first-order language that describes birds and their attributes. Let the model $\mathbf{M}$ describe the living environment of birds in this nature reserve. Let White be a constant of $\mathscr{L}$. If $P(x)$ and $B(x)$ are unary predicate symbols, which are interpreted in $\mathbf{M}$ as *x is white* and *x is a bird* respectively, then the inductive inference may be described by the following rule for the universal quantifier:

$$P[\text{White}] \longrightarrow \forall x P(x), \qquad B[\text{White}] \longrightarrow \forall x B(x).$$

The above example shows that starting from two atomic sentences $P[\text{White}]$ and $B[\text{White}]$, one can induce two universally quantified sentences $\forall x P(x)$ and $\forall x B(x)$. They can be interpreted as: starting from the instance "White is white," the proposition "every swan is white" is induced; starting from the instance "White is a bird" the proposition "every swan is a bird" is induced.

Following the same idea that we used in Chapter 3 to define formal inference, the mechanism of inductive inference can be described by the following rule of calculus for

the universal quantifier:

$$B[t] \longrightarrow \forall x B(x),$$

where t is a Herbrand term containing no variable, $B[t]$ is either an atomic sentence or the negation of an atomic sentence, $\forall x B(x)$ on the right of $\longrightarrow$ is called the *inductive consequence*, and this rule is called the *induction rule* for the universal quantifier.

Induction and refutation. Inductive consequences may hold in some cases, but may not in others. For example, the inductive consequence "every swan is a bird" obtained from "White is a bird" holds, while "every swan is white" induced from "White is white" does not hold, because in that nature reserve there was a black swan named Black. By using the terminology of first-order languages and models, the rule $P[\text{White}] \longrightarrow \forall x P(x)$ should be interpreted as: if $\mathbf{M} \models P[\text{White}]$ holds, then $\mathbf{M} \models \forall x P(x)$ also holds. Since $\mathbf{M} \models \neg P[\text{Black}]$ holds, $\mathbf{M} \models \forall x P(x)$ does not hold. This indicates that the rule $P[\text{White}] \longrightarrow \forall x P(x)$ is not sound in the same way as the corresponding rule of the **G** system.

In the sense of Chapter 7, $\neg P[\text{Black}]$ is a refutation by facts with respect to the inductive consequence $\forall x P(x)$. Therefore, if an inductive consequence is refuted by facts, then it does not hold; on the other hand, if it is not refuted by facts, then it should be provisionally accepted. In other words, when the inductive inference rule is used, one has to check the inductive consequence in the model. If we find a refutation by facts, then it is necessary to revise the formal theory. So induction and refutation are two aspects of the inductive inference process. They are complementary to each other and both of them are indispensable.

Inductive inference and formal inference. We have proved in Chapter 3 that formal inference systems are sound, i.e., if $\Gamma \vdash A$ holds, then for any model $\mathbf{M}$, $\mathbf{M} \models \Gamma$ implies that $\mathbf{M} \models A$. If the interpretation of a formal theory under a model is true, then the interpretations of its formal consequences under this model must also be true. This is the soundness property of formal inference systems.

Inductive inference is different from formal inference. The former is used in the axiomatization process and is a means for improving and refining formal theories. Each inductive consequence is a conjecture about a universal law made on the basis of particular instances. Being a conjecture, it can be either right or wrong and its truth cannot be judged from the truth of a single instance. The correctness of the inductive consequence can only be determined if it is not refuted through the entire axiomatization process.

As inductive inference rules generalize particular instances to universal laws, they are concerned with the generation of new conjectures and new versions. Formal inference is concerned only with the proof of logical consequences, and it is not involved in the generation of new versions.

If we use the terminology of first-order languages and let Γ denote the current version of a formal theory and $\longrightarrow$ denote the inductive inference relation, then the difference between formal inference and inductive inference is:
For formal inference,

$$\text{if } \Gamma \vdash A, \text{ then } Th(\Gamma) = Th(\Gamma \cup \{A\}).$$

This means that new versions cannot be created by formal inference.

For inductive inference,

$$\text{if } \Gamma \longrightarrow\!\triangleright \Gamma', \text{ then } Th(\Gamma) \subsetneq Th(\Gamma').$$

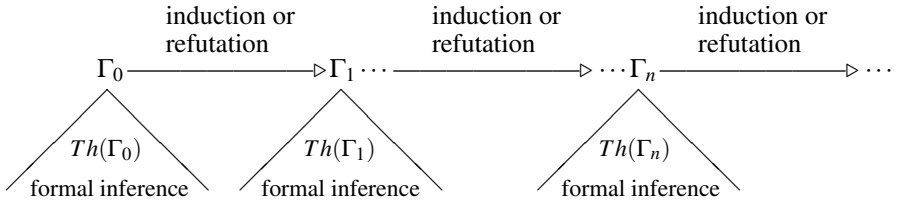
This means that inductive inference adds a new axiom to the system, so a new version is formed which is a proper enlargement of the old one.

Let Γ_n denote the nth version of the formal theory Γ. After applying inductive inference revision rules alternately many times, the versions that are generated form a process of axiomatization:

$$\Gamma_1, \Gamma_2, \ldots, \Gamma_n, \ldots.$$

This version sequence contains two kinds of versions. For example, the $(i+1)$th version Γ_{i+1} might be a new version obtained by applying the induction rule to Γ_i, while the $(j+1)$th version Γ_{j+1} might be a maximal contraction of Γ_j.

If $\longrightarrow\!\triangleright$ denotes both the inductive inference relation and the **R**-contraction relation and the sector region under the version Γ_n denotes the theory closure $Th(\Gamma_n)$ of Γ_n, then the relation between inductive inference and formal inference may be illustrated by the following diagram:



This diagram shows that both induction and revision lead to a change of versions and the evolution of knowledge. In contrast, formal inference takes place only within a particular version and it does not result in a change of theory version. In this sense, one could say that inductive inference and formal inference are orthogonal.

Reliability of inductive proscheme. For a given scientific problem, an inductive consequence may be interpreted as a conjectured law of nature concerning this problem. As a conjecture, it may be right or wrong. Thus a single isolated application of an induction rule does not have soundness. However, this does not mean that the reliability of inductive inference systems cannot be investigated.

What does it mean to say that induction is reliable? From the viewpoint of the axiomatization process, an inductive inference system might be considered reliable if every version sequence generated by applying it to all particular instances starting from arbitrary conjectures converges to all the universal laws about the scientific problem.

If we accept this point of view, then proving the reliability of an inductive inference system may be reduced to looking for a proscheme that gives a workflow such that:

1. it takes as input sentences describing particular instances one by one;

2. outputs a version sequence that has been processed by the inductive inference system;

3. it can be proved that this proscheme is convergent and commutative.

Section 9.1 discusses the question of how to describe particular instances in first-order languages. Section 9.2 discusses the necessity of inductive inference rules and introduces an inductive inference system **A**, which consists of the universal induction rule, the revision rule and the instance addition rule. Section 9.3 presents several types of versions related to inductive inference and introduces the concept of the axiomatization process of inductive inference. Section 9.4 describes an inductive proscheme, called *GUINA*[1]. The convergence and commutativity of the proscheme GUINA are proved in Sections 9.5 and 9.6 respectively. Section 9.7 discusses how to refine the proscheme GUINA so that it possesses independence.

9.1 Ground terms, basic sentences, and basic instances

As we said before, inductive inference is a mechanism for finding universal laws from particular instances. Universal laws refer to the properties of all the members in a domain, which can be described by universally quantified sentences in first-order languages. But what syntactic objects could be used to describe particular instances in first-order languages? This section answers this question.

Let $\wp$ be a scientific problem whose model is **M** and whose corresponding first-order language is $\mathscr{L}$. In this section we explain what particular instances refer to in **M** and how to describe them in the language $\mathscr{L}$.

1. The results of experiments related to the problem $\wp$ are data about simple attributes of particular objects. A common attribute shared by a set of data can be described by a predicate. A particular object in the model, which has such an attribute, is called a *basic instance* of the predicate, or *instances* for short.

   For example, we might observe that *the color of a particular swan named Fred is white*. This is an instance of the color attribute. Also the observation *the color of the swan named Bob is not white* is also an instance of the color attribute. Generally speaking, the basic instances of a model **M** are those atomic predicates or their negations that do not contain variables.

2. The basic properties of a set of elements in **M** are described by predicates or their negations in the first-order language $\mathscr{L}$. Since every instance is a proposition about a particular object and a predicate usually contains variables, the free variables in the predicate should be substituted by constant symbols when we use a predicate to describe an instance. In summary, each atomic sentence or negation of an atomic sentence describes an instance of **M** in $\mathscr{L}$.

[1] GUINA [gwi'na:] is a Chinese phonetic transcription of *induction*.

In the previous example of swans, the predicate $P(x)$ can be interpreted in $\mathbf{M}$ as *the color of the swan named x is white*. White is a constant symbol of $\mathscr{L}$ and the interpretation of the predicate $P[\text{White}]$ in $\mathbf{M}$ is *the color of the swan named White is white*. $\neg P[S^{100}0]$ is similarly interpreted as *the 100th swan is not white*.

3. The Herbrand domain H introduced in Definition 2.12 is a set consisting of all the terms t that contain no free variables. Each term in H is called a *ground term* and each ground term is interpreted as a particular object in $\mathbf{M}$. If $P(x)$ is a predicate, then $P[t]$ is interpreted as an instance in $\mathbf{M}$. For example, $P[S^{100}0]$ and $P[S^{100}0 \cdot S^{50}0]$ are both interpreted as instances in $\mathbf{M}$.

4. According to the principle of excluded middle, each atomic proposition in a domain M is either true or false. Henceforth, we call a true atomic proposition a *positive instance*, and we call a false one a *negative instance*. The complete set of instances of the model $\mathbf{M}$ is composed of all the positive instances and the negations of all the negative instances. This is called the set of *basic sentences* of the language $\mathscr{L}$ with respect to the model $\mathbf{M}$, which is denoted as $\Omega_{\mathbf{M}}$.

   If A is an atomic sentence and is interpreted as a positive instance in $\mathbf{M}$, then $A \in \Omega_{\mathbf{M}}$; if A is an atomic sentence and is interpreted as a negative instance in $\mathbf{M}$, then $\neg A \in \Omega_{\mathbf{M}}$. The set $\Omega_{\mathbf{M}}$ of basic sentences is interpreted as a set consisting of all the basic instances that are true in the model $\mathbf{M}$.

The concept of negative instance introduced in this section is different from the concept of refutation by facts introduced in Chapter 7. "A is a negative instance" refers to the atomic sentence A being false in the model $\mathbf{M}$, whereas $\neg A$ is true. However, A is a refutation by facts of Γ, which describes the relationship between the formal theory Γ and the sentence A, i.e., Γ is false in the model $\mathbf{M}$, whereas A is true in the model $\mathbf{M}$.

All of the above concepts: instances, basic sentences and the complete set of instances, can be defined using first-order languages and their models.

Definition 9.1 (Complete set of basic sentences of model M). Let $\mathscr{L}$ be a first-order language with $\mathbf{M}$ being its model. Let H be the Herbrand domain of $\mathscr{L}$. The complete set of basic sentences of the model $\mathbf{M}$ is defined as follows:

$$\Omega_1 = \{\, A \mid A \text{ is a predicate } P \text{ with no variables and } P_{\mathbf{M}} \text{ is true, or } A \text{ is } \neg P \text{ with } P \text{ being a predicate with no variables and } (\neg P)_{\mathbf{M}} \text{ is true} \,\};$$

$$\Omega_{n+1} = \Omega_n \cup \{\, A[t_1,\ldots,t_n] \mid t_j \in H,\ A \text{ is an } n\text{-ary predicate } P[t_1,\ldots,t_n] \text{ and } (P[t_1,\ldots,t_n])_{\mathbf{M}} \text{ is true, or } A[t_1,\ldots,t_n] \text{ is } \neg P[t_1,\ldots,t_n] \text{ and } (\neg P[t_1,\ldots,t_n])_{\mathbf{M}} \text{ is true} \,\};$$

$$\Omega = \bigcup_{i=1}^{\infty} \Omega_i.$$

The set Ω is called the *complete set of basic sentences* of $\mathscr{L}$ with respect to $\mathbf{M}$ and is denoted as $\Omega_{\mathbf{M}}$. The set $\Omega_{\mathbf{M}}$ is countable and, when ordered, it is called the *complete sequence of basic sentences* of $\mathscr{L}$ with respect to $\mathbf{M}$.

The complete set $\Omega_{\mathbf{M}}$ of basic sentences of the model $\mathbf{M}$ is interpreted as the complete set of instances in $\mathbf{M}$. It uniquely determines a Hintikka set (see Definition 2.13) whose model is $\mathbf{M}$.

9.2 Inductive inference system A

In this section we introduce the inductive inference system **A** that includes the universal induction rule, the revision rule and the instance expansion rule. In this section, we will demonstrate, through examples, the necessity of these rules, the unsoundness of the universal induction rule and other possible choices for induction rules.

First of all, let us show that inductive inference is necessary in "the passage from particulars to universals".

Example 9.1 (Necessity of inductive inference). For simplicity, suppose that the set of constant symbols of the first-order language $\mathscr{L}$ is $\{c_n\}$ and $\mathscr{L}$ does not contain any function symbols. Suppose also that $\mathscr{L}$ contains only one unary predicate $P(x)$. Then the Herbrand domain of $\mathscr{L}$ is simply the set $\{c_n\}$. The complete set $\Omega_{\mathbf{M}}$ of basic sentences of the model $\mathbf{M}$ is $\{P[c_n]\}$, i.e., for every n, $P[c_n]$ is a positive instance of $\mathbf{M}$.

In this case for the model $\mathbf{M}$, $\forall x P(x)$ holds. We certainly expect that

$$\{P[c_1], \ldots, P[c_n], \ldots\} \vdash \forall x P(x),$$

i.e., the universally quantified sentence $\forall x P(x)$ is a formal consequence of the complete set $\Omega_{\mathbf{M}}$ of basic sentences. According to Chapter 3, in order to prove that this sequent is provable, we need to apply the $\forall$-*R* rule. By the definition of the **G** system, the numerator of this rule must be provable. The numerator of the $\forall$-*R* rule is

$$\{P[c_1], \ldots, P[c_n], \ldots\} \vdash P(y). \qquad (*)$$

Because y in the sequent $(*)$ is an eigen-variable different from $\{c_n\}$, this formula cannot be an axiom and thus it is not provable. This shows that $\forall x P(x)$ is not a formal consequence of the sequence $\{P[c_n]\}$. If $\forall x P(x)$ is not a conclusion of formal proofs, then what kind of conclusion is it? It can only be an *inductive consequence* of $\{P[c_n]\}$, i.e., it is a conclusion *induced* from all the instances.

This example shows that in the axiomatization process, the inductive mechanism for the "passage from particulars to universals" is indispensable.

A new axiom that is generated by inductive inference is meaningful only in the context of a specific problem, while formal inference is sound in all situations. In order to emphasize this essential difference between inductive inference and formal inference, we use the following fraction to describe inductive inference rules:

$$\frac{\mathbf{condition}(\Gamma,\ P[t],\ \Omega_{\mathbf{M}})}{\Gamma \longrightarrow\!\triangleright\ \Gamma'}.$$

Γ and Γ' in the denominator of the fraction are formal theories with Γ being an old version and Γ' being the new version generated by the inductive inference rule. The premise

$$\textbf{condition}(\Gamma,\ P[t],\ \Omega_{\mathbf{M}})$$

in the numerator of the rule denotes the relationship between the current version Γ and the basic sentence $P[t]$. The rule can be interpreted as: if the premise **condition**$(\Gamma,\ P[t],\ \Omega_{\mathbf{M}})$ holds, then we can induce the new version Γ' from Γ. The numerator **condition**$(\Gamma,\ P[t],\ \Omega_{\mathbf{M}})$ gives the condition to apply the induction rule.

We will show the role of **condition**$(\Gamma, P[t], \Omega_{\mathbf{M}})$ in the following example.

Example 9.2 (Acceptable conjecture). Suppose that the scientific problem to be examined is **M**. Also suppose that $\Omega_{\mathbf{M}} = \{P[c_1],\ \neg P[c_2],\ Q[c_1],\ Q[c_2]\}$. Let $\Gamma = \{P[c_1],\ Q[c_1]\}$ and $Q[c_2]$[2] be the basic instance to be examined. If we induce the universal consequence $\forall x Q(x)$ from the basic instance $Q[c_2]$, then it is feasible to write the rule as

$$\frac{Q[c_2] \text{ and } \Gamma \text{ are consistent}}{\Gamma \longrightarrow\!\triangleright\ \forall x Q(x),\ \Gamma},$$

since in this case the inductive consequence $\{\forall x Q(x),\ P[c_1],\ Q[c_1]\}$ is a formal theory.

Now suppose the basic instance to be examined is $\neg P[c_2]$. The consequence induced from this basic instance is $\forall x \neg P(x)$, which can be written into a rule

$$\frac{\neg P[c_2] \text{ and } \Gamma \text{ are consistent}}{\Gamma \longrightarrow\!\triangleright\ \forall x \neg P(x),\ \Gamma}.$$

In this case, the inductive consequence is not acceptable because $\forall x \neg P(x)$ and $P[c_1]$ are inconsistent. Hence the newly generated version $\{\forall x \neg P(x),\ P[c_1],\ Q[c_1]\}$ is not a formal theory. The correct rule should be

$$\frac{\neg P[c_2] \text{ and } \Gamma \text{ are consistent}}{\Gamma \longrightarrow\!\triangleright\ \neg P[c_2],\ \Gamma}.$$

The above two cases show that the inductive inference rules should ensure the consistency of the new version generated. For this purpose we introduce the following relation.

Definition 9.2 (Acceptable relation). Suppose that Γ is a formal theory and $P[t]$ and $\neg P[t']$ are basic sentences with $t, t' \in H$ being ground terms.

(1) If $P[t]$ is consistent with Γ and there does not exist a ground term $t' \in H$ such that $\neg P[t'] \in Th(\Gamma)$, then we say that $P[t]$ is *acceptable* in Γ and denote it as $P[t] \bowtie \Gamma$.

(2) If $P[t]$ is consistent with Γ and there exists a ground term $t' \in H$ such that $\neg P[t'] \in Th(\Gamma)$, then we say that $P[t]$ is *non-acceptable* in Γ and denote it as $P[t] \not\bowtie \Gamma$.

[2] Starting from this example, the so-called basic instance $Q[c_2]$ actually refers to the interpretation of the basic sentence $Q[c_2]$ in **M**.

In the above example, according to (1) of Definition 9.2, $Q[c_2]$ is acceptable in Γ; according to (2) of Definition 9.2, $\neg P[c_2]$ is non-acceptable in Γ.

We are now ready to introduce the inductive inference rules. Suppose that $\mathbf{M}$ is a scientific problem and the complete set of basic sentences of $\mathbf{M}$ is $\Omega_{\mathbf{M}}$.

Definition 9.3 (Universal induction rule).

$$\frac{P[t] \bowtie \Gamma \quad P[t] \in \Omega_{\mathbf{M}}}{\Gamma \longrightarrow_i \forall x P(x), \Gamma}$$

The universal induction rule is a formal rule that induces a universally quantified sentence from a particular basic sentence. This rule shows that we can induce $\forall x P(x)$ from $P[t]$, for some ground term t, where $P[t]$ is an acceptable basic sentence in the current version Γ. The new version generated by this induction is $\forall x P(x), \Gamma$. The sentence $\forall x P(x)$ is called the *inductive consequence* of the rule. The subscript i of $\longrightarrow_i$ in the denominator of the rule denotes that this transition is formed from universal induction.

Definition 9.4 (Revision rule).

$$\frac{\Gamma \vdash \neg P[t] \quad P[t] \in \Omega_{\mathbf{M}}}{\Gamma \longrightarrow_r R(\Gamma, P[t]), P[t]}$$

This rule should be used when the basic sentence $P[t]$ is a formal refutation of the current version Γ. The generated new version is $R(\Gamma, P[t]), P[t]$ and it is called the *revision consequence* of the current version with respect to the formal refutation $P[t]$. $R(\Gamma, P[t])$ is a maximal contraction of Γ with respect to $P[t]$. The subscript r of $\longrightarrow_r$ in the denominator of the rule denotes that this transition is formed from a refutation.

Definition 9.5 (Instance addition rule).

$$\frac{P[t] \not\bowtie \Gamma \quad P[t] \in \Omega_{\mathbf{M}}}{\Gamma \longrightarrow_e P[t], \Gamma}$$

This rule shows that the basic sentence $P[t]$ is non-acceptable in the current version Γ. Thus we should accept the particular instance $P[t]$ as a new axiom of Γ, but we cannot apply the universal induction rule to introduce $\forall x P(x)$. So the new version is $\{P[t]\} \cup \Gamma$. The subscript a of $\longrightarrow_a$ in the denominator of the rule denotes that this transition is formed from an addition.

Universal induction, revision and instance addition are all rules of symbolic calculus that create new versions of a formal theory. Unless stated otherwise, in this chapter $\longrightarrow$ denotes all the above three transitions.

The following example shows that universal inductive inference does not possess soundness.

Example 9.3 (Relation between universal induction and soundness). For a given first-order language $\mathscr{L}$, let the Herbrand domain of $\mathscr{L}$ be $H = \{a, b\}$. Suppose that $\mathscr{L}$ contains only one unary predicate $P(x)$. Consider two models $\mathbf{M}_1$ and $\mathbf{M}_2$ of $\mathscr{L}$. Suppose that the

complete sets of basic sentences of $\mathscr{L}$ with respect to $\mathbf{M}_1$ and $\mathbf{M}_2$ are $\Omega_{\mathbf{M}_1} = \{P[a], P[b]\}$ and $\Omega_{\mathbf{M}_2} = \{P[a], \neg P[b]\}$ respectively.

Let the current version be $\Gamma = \varnothing$ and consider the basic sentence $P[a]$. Since $P[a] \bowtie \Gamma$ holds, we can use the universal induction rule to obtain

$$\varnothing \longrightarrow_{\triangleright} \{\forall x P(x)\}.$$

Here $\forall x P(x)$ is the inductive consequence of Γ and $P[a]$. It is not difficult to verify that both $\mathbf{M}_1 \models P[a]$ and $\mathbf{M}_2 \models P[a]$ hold, but $\mathbf{M}_1 \models \forall x P(x)$ holds and $\mathbf{M}_2 \models \forall x P(x)$ does not hold.

This example shows that inductive inference is not sound in the same sense as the formal inference systems discussed in Chapter 3. This is because the inductive inference rules search for new axioms that describe specific knowledge in a particular model. Inductive inference rules are not rules for logical connectives and quantifiers, while soundness is a property of rules for logical connectives and quantifiers.

Example 9.4 (About the revision rule). Suppose that the first-order language $\mathscr{L}$ is the same as in the above example with $\mathbf{M}_2$ being a model of $\mathscr{L}$ and the complete set of basic sentences of $\mathbf{M}_2$ is $\Omega_{\mathbf{M}_2} = \{P[a], \neg P[b]\}$.

(1) Let the initial version be $\Gamma_1 = \varnothing$. Since the basic sentence $P[a]$ is acceptable in Γ_1, by using the universal induction rule we can obtain

$$\varnothing \longrightarrow_{i\triangleright} \{\forall x P(x)\}.$$

The new version is $\Gamma_2 = \{\forall x P(x)\}$.

(2) Consider the relation between Γ_2 and the basic sentence $\neg P[b]$. According to the **G** system, $\forall x P(x) \vdash P[b]$ is provable, which is $\Gamma_2 \vdash P[b]$ being provable. Thus $\neg P[b]$ is a formal refutation of Γ_2. Using the revision rule on Γ_2 and $\neg P[b]$ we have

$$\Gamma_2 \longrightarrow_{r\triangleright} \{\neg P[b]\}.$$

Let the new version be $\Gamma_3 = \{\neg P[b]\}$.

This example shows that after applying the universal induction rule, we have to use the revision rule to revise any inconsistency between the inductive consequence and the complete set of instances. It also shows that universal induction and revision are complementary aspects of the inductive inference mechanism.

Notice that in the process of applying the induction rule the instance $P[a]$ is lost. At the time, this didn't matter because $\forall x P(x)$ implies $P[a]$. But when the revision rule deleted $\forall x P(x)$, we ended up with a version that does not include the valid instance $P[a]$. There are two methods of resolving this problem:

(1) Change the universal induction rule to:

Universal induction rule-I

$$\frac{P[t] \bowtie \Gamma \quad P[t] \in \Omega_{\mathbf{M}}}{\Gamma \longrightarrow_{i\triangleright} P[t],\ \forall x P(x), \Gamma}.$$

In this new induction rule, the new version retains the basic sentences that induced the inductive consequences. Since the basic sentence $P[a]$ is acceptable in the version $\Gamma_1 = \varnothing$, we can use the universal induction rule-I to obtain

$$\varnothing \longrightarrow\!\triangleright_i \{P[a],\ \forall x P(x)\}.$$

In this way the new version is $\Gamma_2' = \{P[a],\ \forall x P(x)\}$. Then by using the refutation revision rule on Γ_2' and the basic sentence $\neg P[b]$, we can obtain

$$\Gamma_3 = \{P[a],\ \neg P[b]\}.$$

By using the universal induction rule-I, it is ensured that the basic sentence $P[a]$ is no longer lost if revision ever deletes the universal sentence. However, this may mean that the new version no longer possesses independence.

(2) Another method that can both prevent the loss of basic sentences and keep the independence of Γ_2', is to design a proscheme containing mechanisms for storing instances similar to the sets Δ and Θ in the proscheme `OPEN` in Chapter 8.

One other justification for induction has been proposed in the literature. This is the so-called sufficient condition inference rule, which is defined as follows:

if $A \rightarrow B$ and B both hold, then A is ***induced****.*

This has meaning if the implication $\rightarrow$ is used in its common sense, implying causality. For instance, the sun rising implies it is day. If it is day, we can reasonably induce that the sun has risen.

If we try to express this as rule of inductive inference it would say:

$$\{A \rightarrow B,\ B,\ \Gamma\} \ \longrightarrow\!\triangleright\ \ \{A,\ A \rightarrow B,\ B,\ \Gamma\}.$$

However, if the implication $\rightarrow$ is logical implication, then this inference has no meaning. This is because, if we know that B holds, then $A \rightarrow B$ always holds. $A \rightarrow B$ is a formal consequence of B. One can verify this by noting that the sequent $B \vdash \neg A \vee B$ is provable in the **G** system, since $B \vdash C \vee B$ is provable for any formula C. But $\neg A \vee B$ is equivalent to $A \rightarrow B$. Hence $A \rightarrow B$ can be deleted from both sides of the above rule, and it becomes:

$$\{B,\ \Gamma\} \ \longrightarrow\!\triangleright\ \ \{A,\ B,\ \Gamma\}.$$

Since, in this rule, A can be any formula, even one that has no connection to B, we cannot simply translate this motivation for induction into a logical system. To really express the meaning of induction on sufficient conditions, we need to restrict our choice of sufficient condition A to ensure that it is, in some sense, causally related to B.

For instance, although this rather defeats the motivation for talking about *sufficient conditions*, we can require A in the rule to be a necessary antecedent in the sense of Chapter 7. The rule then has the following form (necessary antecedent induction):

$$\frac{A, \Gamma \vdash B \quad A \mapsto B}{B, \Gamma \longrightarrow\!\triangleright A, B, \Gamma}.$$

This rule is logically reasonable, if A is a necessary antecedent to B and if we know that B holds, then we can reasonably induce A holds.

However, the universal induction rule alone is enough for our purposes. We shall prove in Section 9.5 that there exists a well-designed proscheme that applies the universal induction rule, the revision rule and the addition rule to ensure the convergence of output formal theories and that its theory closure sequence converges to $Th(\mathbf{M})$. In this way we can fulfill the objective of inducing all the true propositions from particular instances.

9.3 Inductive versions and inductive process

A new version of a formal theory that is generated by inductive inference is called an *inductive version*.

Definition 9.6 (Inductive version). Suppose that Γ is a formal theory and P is a basic sentence.

If a formal theory Γ' is a new version generated by applying the universal induction rule to Γ and P, then we call Γ' a *universal inductive version* of Γ with respect to P, or an *I-type version* of Γ.

If a formal theory Γ' is a new version generated by applying the revision rule to Γ and P, then we call Γ' an *R-type version* of Γ with respect to P.

If a formal theory Γ' is a new version generated by applying the instance addition rule to Γ and P, then we call Γ' an *N-type version* of Γ with respect to P.

Definition 9.7 (Inductive sequence). We call the sequence

$$\Gamma_1, \Gamma_2, \ldots, \Gamma_n, \ldots$$

an *inductive sequence* if for every natural number n, Γ_{n+1} is an I-type, R-type or N-type version of Γ_n. An inductive sequence is also called an *inductive process*.

Lemma 9.1. *An inductive sequence $\{\Gamma_n\}$ is an increasing sequence if and only if for every $n \geqslant 1$, Γ_{n+1} is an I-type or N-type version of Γ_n.*

Proof. It follows immediately from the definition. □

9.4 The Proscheme GUINA

The purpose of the following sections is to introduce an inductive proscheme named GUINA. We will prove that it is a reliable proscheme, i.e., it possesses convergence and commutativity, and define the conditions under which it possesses independence. The basic design strategy of GUINA is as follows.

The proscheme GUINA inputs the initial theory Γ, which is also called the *initial conjecture* in this chapter, and the basic sentence sequence $\Omega_{\mathbf{M}}$. Each time a basic instance is input, GUINA calls its sub-procedure GUINA* once. Using the same mechanism as we

did for the proscheme OPEN, we need to do the following in GUINA to ensure the reliability of the output version sequence.

(1) Introduce a set Δ to store the basic sentences, which have previously induced universally quantified sentences. Δ is used in the following way: when a universally quantified formula is deleted due to refutation, any deleted instances used in the induction of that formula are added back into the new version.

(2) Introduce a set Θ to store the instances P_m, $m < n$, that were previously input in forming the first n versions. These instances are logical consequences of the corresponding versions. Θ is also used when formulas are deleted through refutation. The proscheme examines each P_m contained in Θ individually to see whether it is still a logical consequence of the current version and, if not, then adds it into the new version.

(3) The initial states of Δ and Θ are $\varnothing$.

In the same way as the proscheme OPEN, GUINA calls its sub-procedure GUINA* every time a basic sentence in $\Omega_{\mathbf{M}}$ is input. GUINA* takes the current version Γ_n and basic sentence $P_n[t]$ as inputs. It outputs a new version Γ_{n+1} according to their logical relationship as in the following situations.

1. $\Gamma_n \vdash P_n[t]$ is provable. The input basic sentence is a formal consequence of the current version Γ_n. In this case it is unnecessary to use the induction rules. The outputs of GUINA* are $\Gamma_{n+1} := \Gamma_n$, $\Theta_{n+1} := \{P_n[t]\} \cup \Theta_n$, and $\Delta_{n+1} := \Delta_n$.

2. $\Gamma_n \vdash \neg P_n[t]$ is provable. Since $P_n[t] \in \Omega_{\mathbf{M}}$, it has to be accepted. This shows that the formal consequence $\neg P_n[t]$ of Γ_n is refuted by $P_n[t]$. In this case, the new version can be obtained by the following two steps.

   (a) We first apply the revision rule and make a new version from the union of a maximal contraction of Γ_n and $\{P_n[t]\}$.

   (b) Then we examine the basic sentences in Θ_n and Δ_n individually and add to the new version those basic sentences that are not logical consequences of the current version.

   Now $\Theta_{n+1} := \{P_n[t]\} \cup \Theta_n$ and $\Delta_{n+1} := \Delta_n$.

3. Neither $\Gamma_n \vdash P_n[t]$ nor $\Gamma_n \vdash \neg P_n[t]$ is provable. There are two cases as follows.

   (a) $P_n[t] \not\bowtie \Gamma_n$. This means that $P_n[t]$ is a new basic sentence of Γ_n and there exists a t' such that $\neg P_n[t'] \in Th(\Gamma_n)$ holds. In this case we can only use the instance addition rule. The outputs are $\Gamma_{n+1} := \{P_n[t]\} \cup \Gamma_n$, $\Delta_{n+1} := \Delta_n$, and $\Theta_{n+1} := \{P_n[t]\} \cup \Theta_n$.

   (b) The above case does not hold, i.e., $P_n[t] \bowtie \Gamma_n$ holds. This means that $P_n[t]$ is a new basic sentence of Γ_n and there does not exist any t' such that $\neg P_n[t'] \in Th(\Gamma_n)$ holds. In this case we use the universal induction rule on $P_n[t]$ to obtain a new inductive version $\Gamma_{n+1} := \{\forall x P_n(x)\} \cup \Gamma_n$, $\Delta_{n+1} := \{P_n[t]\} \cup \Delta_n$, and $\Theta_{n+1} := \Theta_n$.

In what follows we give a description of the proscheme GUINA.

Definition 9.8 (Proscheme GUINA). Suppose that **M** is the model of the given problem whose complete set $\Omega_{\mathbf{M}}$ of basic sentences is $\{P_n[t]\}$.

proscheme GUINA(Γ: theory; $\{P_n[t]\}$: formula sequence)
Γ_n: theory;
Θ_n, Θ_{n+1}: theory;
Δ_n, Δ_{n+1}: theory;
proscheme GUINA*(Γ_n: theory; $P_n[t]$: basic sentence; **var** Γ_{n+1}: theory)
begin
if $\Gamma_n \vdash P_n[t]$ **then**
begin
$\Gamma_{n+1} := \Gamma_n$;
$\Theta_{n+1} := \Theta_n \cup \{P_n[t]\}$;
$\Delta_{n+1} := \Delta_n$
end
else if $\Gamma_n \vdash \neg P_n[t]$ **then**
begin
$\Gamma_{n+1} := \{P_n[t]\} \cup R(\Gamma_n, P_n[t])$;
loop until (for every $B_i \in \Delta_n \cup \Theta_n$, $\Gamma_{n+1} \vdash B_i$)
loop for every $B_i \in \Delta_n \cup \Theta_n$
if $\Gamma_{n+1} \vdash B_i$ **then skip**
else if $\Gamma_{n+1} \vdash \neg B_i$ **then**
$\Gamma_{n+1} := R(\Gamma_{n+1}, B_i) \cup \{B_i\}$
else $\Gamma_{n+1} := \Gamma_{n+1} \cup \{B_i\}$
end loop
end loop
$\Theta_{n+1} := \Theta_n \cup \{P_n[t]\}$;
$\Delta_{n+1} := \Delta_n$
end
else if $P_n[t] \not\bowtie \Gamma_n$ **then**
begin
$\Gamma_{n+1} := \Gamma_n \cup \{P_n[t]\}$;
$\Theta_{n+1} := \Theta_n \cup \{P_n[t]\}$;
$\Delta_{n+1} := \Delta_n$
end
else
begin
$\Gamma_{n+1} := \Gamma_n \cup \{\forall x P_n(x)\}$;
$\Theta_{n+1} := \Theta_n$;
$\Delta_{n+1} := \Delta_n \cup \{P_n[t]\}$
end
end

begin
  $n := 1$;
  $\Gamma_n := \Gamma$;
  $\Theta_n := \varnothing$; $\Theta_{n+1} := \varnothing$;
  $\Delta_n := \varnothing$; $\Delta_{n+1} := \varnothing$;
  loop
    GUINA$^*(\Gamma_n, P_n[t], \Gamma_{n+1})$;
    print Γ_{n+1};
    $n := n+1$
  end loop
end

In the proscheme $R(\Gamma_n, P_n[t])$ is a maximal contraction of Γ_n with respect to $P_n[t]$, and $(\Gamma_n - R(\Gamma_n, P_n[t])) \cap (\Delta_n \cup \Theta_n) = \varnothing$ holds. Both Θ_n and Δ_n are subsets of $\Omega_{\mathbf{M}}$ and hence their type is `theory`.

Definition 9.9 (Complete inductive sequence). If the proscheme GUINA takes Γ as its initial theory and the complete set $\Omega_{\mathbf{M}}$ of basic sentences of the model $\mathbf{M}$ as its input sequence, then the output version sequence $\{\Gamma_n\}$ of GUINA is called the *complete inductive sequence* of the proscheme GUINA with respect to the model $\mathbf{M}$ and initial theory Γ.

Lemma 9.2. *If the initial theory is a formal theory, then every element Γ_n in the complete inductive sequence $\{\Gamma_n\}$ of the proscheme GUINA with respect to the model $\mathbf{M}$ and initial conjecture Γ is a formal theory.*

Proof. It follows immediately from the construction of the proscheme GUINA. □

The following example can demonstrate the workflow of the proscheme GUINA.

Example 9.5 (Applications of GUINA). For a given first-order language $\mathscr{L}$ and its model $\mathbf{M}$, suppose that $\mathscr{L}$ contains two constant symbols a and c, but does not contain any function symbol. Also suppose that $\mathscr{L}$ contains only two unary predicates $P(x)$ and $Q(x)$. We know by definition that

the Herbrand domain of $\mathscr{L}$ is $H = \{a, c\}$,
the set of atomic sentences of $\mathscr{L}$ is $\mathbf{P} = \{P[a], P[c], Q[a], Q[c]\}$.
Let the complete set of basic sentences of $\mathscr{L}$ with respect to $\mathbf{M}$ be
$\Omega_{\mathbf{M}} = \{P[a], \neg P[c], Q[a], Q[c]\}$.

Finally, let the inputs of the proscheme GUINA be the initial theory $\Gamma = \varnothing$ and the complete sequence $\Omega_{\mathbf{M}}$ of basic sentences. The workflow of GUINA is as follows.

(1) When GUINA starts to execute, $\Theta_1 := \varnothing$, $\Delta_1 := \varnothing$, $\Gamma_1 := \varnothing$.

(2) The first time GUINA* is called, the inputs of GUINA* are Γ_1 and $P[a]$. Since $\Gamma_1 = \varnothing$, only the program segment after the **else begin** in the body of GUINA* can be executed. After the first call of GUINA*, we have

$$\Gamma_2 := \{\forall x P(x)\},$$

$\Theta_2 := \varnothing, \Delta_2 := \{P[a]\}$.

(3) GUINA* is called the second time. The inputs of GUINA* are Γ_2 and $\neg P[c]$. Since $\Gamma_2 \vdash P[c]$ is provable, the input $\neg P[c]$ of GUINA* in this second round of execution is a formal refutation of Γ_2. In this case, GUINA* uses the revision rule, i.e., executes the program segment delimited by the first **else if** in the body of GUINA*. After the second call of GUINA*, we have

$$\Gamma_3 := \{P[a], \neg P[c]\},$$

$\Theta_3 := \{\neg P[c]\}, \Delta_3 := \{P[a]\}$. $P[a]$ in Γ_3 is retrieved from Δ_2.

(4) GUINA* is called the third time with inputs Γ_3 and $Q[a]$. Since $Q[a] \bowtie \Gamma_3$, GUINA* uses the universal induction rule again and executes the program segment after the **else begin** in the body of GUINA*. After the third call of GUINA*, we have

$$\Gamma_4 := \{P[a], \neg P[c], \forall x Q(x)\},$$

$\Theta_4 := \{\neg P[c]\}, \Delta_4 := \{Q[a], P[a]\}$.

(5) GUINA* is called the fourth time. This time its inputs are Γ_4 and $Q[c]$. Since

$$\{P[a], \neg P[c], \forall x Q(x)\} \vdash Q[c]$$

is provable, GUINA* executes the program segment after the first **then** in its procedure body. After the fourth call of GUINA*, we have

$$\Gamma_5 := \{P[a], \neg P[c], \forall x Q(x)\},$$

$\Theta_5 := \{\neg P[c], Q[c]\}, \Delta_5 := \{Q[a], P[a]\}$.

Now the execution of GUINA terminates. It outputs a formal theory Γ_5. It is not difficult to verify that Γ_5 is an independent theory. With Γ_5 as the premise, we can further prove other formal consequences. For instance,

$$\Gamma_5 \vdash (\forall x P(x)) \to Q(y).$$

In fact, since $\Gamma_5 \vdash \neg P(c)$, according to the $\exists$-R rule, we can prove that $\Gamma_5 \vdash (\exists x \neg P(x))$ holds. Then according to the $\vee$-R rule, we can prove that $\Gamma_5 \vdash (\exists x \neg P(x)) \vee Q(y)$. Since both

$$(\exists x \neg P(x)) \vee Q(y) \vdash (\neg \forall x P(x)) \vee Q(y) \text{ and } (\neg \forall x P(x)) \vee Q(y) \vdash (\forall x P(x)) \to Q(y)$$

are provable, $\Gamma_5 \vdash (\forall x P(x)) \to Q(y)$ is provable.

We can make the following illustration of the above inductive process $\Gamma_1, \Gamma_2, \Gamma_3, \Gamma_4$, Γ_5 generated by the proscheme GUINA. Let $P(x)$ denote the Galilean transformation, $Q(x)$ the Lorentz transformation, a a rigid body in uniform motion, and c a photon.

Then the basic sentences contained in the set $\{P[a], \neg P[c], Q[a], Q[c]\}$ are all results of observations. From $P[a]$ being true, Galileo induced the Galilean transformation

$\forall xP(x)$, which is Γ_2. Experiments showed that $\neg P[c]$ is true, i.e., the Galilean transformation does not hold for the photon.

Because of this fact, Einstein introduced the principle of constancy of the velocity of light and abandoned the Galilean transformation, which resulted in Γ_3. Experiments had already found that $Q[c]$ is true, i.e., the motion of a photon satisfies the Lorentz transformation. Einstein induced that the motion of all particles can be described by the Lorentz transformation and established the special theory of relativity. Later, very precise experiments showed that, for many particles, $Q[a]$ is true. So the theory is accepted at present and is waiting for new evidence to challenge it.

The inductive process in this example is a formal description of the process explained in [Einstein, 1921].

According to the induction rules introduced in the previous section, one can only induce on the basic sentence $P[t]$ to obtain $\forall xP(x)$. But these induced sentences are only a subset of all the universal sentences in $Th(\mathbf{M})$. Our question is: for an arbitrary model $\mathbf{M}$, can we use the proscheme GUINA to make all the universal sentences in $Th(\mathbf{M})$ formal consequences of the inductive version? Or, at least, are they formal consequences of an inductive version somewhere in the output sequence? The answer is affirmative and it is a corollary of the following lemma. First of all let us make the following three technical preparations.

Firstly, suppose that V is the variable set of the first-order language $\mathscr{L}$ and the structure $\mathbf{M} = (\mathbb{M}, \mathbf{I})$. The interpretation of every sentence in $Th(\mathbf{M})$ of $\mathscr{L}$ is true in the model $(\mathbf{M}, \sigma)$. For $Th(\mathbf{M})$, only those elements in the domain $\mathbb{M}$ that can serve as the interpretations in $\mathbb{M}$ of some Herbrand terms (variables allowed) of $\mathscr{L}$, are meaningful. Let us denote all of these elements, i.e., the interpretation of the Herbrand domain of $\mathscr{L}$ in $\mathbb{M}$, as $\mathbf{H}_{\mathscr{L}}(\mathbf{M})$. Generally speaking, $\mathbf{H}_{\mathscr{L}}(\mathbf{M})$ is a subset of $\mathbb{M}$. Nonetheless, for simplicity we use $\mathbb{M}$ instead of it since we only discuss $\mathbf{H}_{\mathscr{L}}(\mathbf{M})$ in this chapter.

Secondly, we need to technically improve the universal formula $\forall xA$ as follows. According to the semantics of logical formulas in Section 2.5, $\mathbf{M} \models_\sigma \forall xA$ means that

$$(A)_{\mathbf{M}[\sigma[x:=a]]} = \mathrm{T} \text{ for any } a \in \mathbb{M}, \text{ i.e., } a \in \mathbf{H}_{\mathscr{L}}(\mathbf{M}).$$

The elements in the variable set V of $\mathscr{L}$ can be further divided into two categories.

For every formula A in $\mathscr{L}$, let $V_{app}(A)$ denote the set consisting of the free variables and bound variables in A. Let y be an eigen-variable with respect to the formula A, i.e., $y \notin V_{app}(A)$.

The formulas in $\mathscr{L}$ can be ordered as a sequence $\{A_n\}$, since they are countable. For each A_n, let y_n be an eigen-variable such that all y_n's are mutually different.

Let the set V'' be all $y_n \notin V_{app}(A_n)$, and let the set V' be all the free variables and bound variables appearing in the formula sequence $\{A_n\}$. Then $V = V' \cup V''$.

For simplicity, in the following, we use x to denote a variable in V' and use y to denote an eigen-variable in V'' corresponding to x.

Finally, for every assignment $\sigma : V \longrightarrow \mathbf{H}_{\mathscr{L}}(\mathbf{M})$ of the formula $\forall xA$, we can define a new assignment $\sigma' : V \longrightarrow \mathbf{H}_{\mathscr{L}}(\mathbf{M})$ as follows:

$$\sigma'(z) = \begin{cases} \sigma(x), & \text{if } z = y, \\ \sigma(z), & \text{otherwise.} \end{cases}$$

It is easy to prove that σ' and σ are in one-to-one correspondence. According to the substitution lemma, the following holds for any $a \in \mathbf{H}_{\mathscr{L}}(\mathbf{M})$:

$$(A[y/x])_{\mathbf{M}[\sigma'[x:=a]]} = A_{\mathbf{M}[\sigma'[x=(y)_{\mathbf{M}[\sigma'[x:=a]]}][x:=a]]} = A_{\mathbf{M}[\sigma[x:=a]]}.$$

Hence $A_{\mathbf{M}[\sigma[x:=a]]} = \mathrm{T}$ holds if and only if $(A[y/x])_{\mathbf{M}[\sigma'[x:=a]]} = \mathrm{T}$ holds for any $a \in \mathbf{H}_{\mathscr{L}}(\mathbf{M})$.

Lemma 9.3. *Suppose that* $\mathbf{M}$ *is a scientific problem and* $\mathscr{L}$ *is its corresponding first-order language with* Γ *being a formal theory of* $\mathscr{L}$*. Also suppose that the inputs of the proscheme* GUINA *are the complete sequence* $\Omega_{\mathbf{M}}$ *of basic sentences of* $\mathbf{M}$ *and the initial theory* Γ*, and the output version sequence of* GUINA *is* $\{\Gamma_n\}$*. For an arbitrary sentence* A *of* $\mathscr{L}$*, if* $\mathbf{M} \models A$*, then* $\{\Gamma_n\}_* \vdash A$ *is provable.*

Proof. (1) A is a basic sentence $P[t]$ with $t \in H$ and $\mathbf{M} \models P[t]$. In this case $P[t] \in \Omega_{\mathbf{M}}$. Let $P[t]$ be the N_1th element of $\Omega_{\mathbf{M}}$. By the definition of the proscheme GUINA, $P[t] \in \Gamma_n$ when $n > N_1$. Hence $P[t] \in \{\Gamma_n\}_*$ and $\{\Gamma_n\}_* \vdash P[t]$ is provable.

(2) A is a basic sentence $\neg P[t]$, $t \in H$ and $\mathbf{M} \models \neg P[t]$. In this case $\neg P[t] \in \Omega_{\mathbf{M}}$. Let $\neg P[t]$ be the N_2th element of $\Omega_{\mathbf{M}}$. By the definition of the proscheme GUINA, $\neg P[t] \in \Gamma_n$ when $n > N_2$. Hence $\neg P[t] \in \{\Gamma_n\}_*$ and $\{\Gamma_n\}_* \vdash \neg P[t]$ is provable.

(3) A is $A_1 \wedge A_2$ and $\mathbf{M} \models_\sigma A_1 \wedge A_2$ for every assignment σ. By the semantics of $\wedge$,

$$(A_1)_{\mathbf{M}[\sigma]} = \mathrm{T} \text{ and } (A_2)_{\mathbf{M}[\sigma]} = \mathrm{T}.$$

By the hypothesis of the structural induction, both $\{\Gamma_n\}_* \vdash A_1$ and $\{\Gamma_n\}_* \vdash A_2$ are provable. By the $\wedge$-*R* rule of the **G** system, $\{\Gamma_n\}_* \vdash A_1 \wedge A_2$ is provable.

(4) A is $A_1 \vee A_2$ and $\mathbf{M} \models_\sigma A_1 \vee A_2$ for every assignment σ. By the semantics of $\vee$,

$$(A_1)_{\mathbf{M}[\sigma]} = \mathrm{T} \text{ or } (A_2)_{\mathbf{M}[\sigma]} = \mathrm{T}.$$

By the hypothesis of the structural induction, $\{\Gamma_n\}_* \vdash A_1$ or $\{\Gamma_n\}_* \vdash A_2$ is provable. By the $\vee$-*R* rule of the **G** system, $\{\Gamma_n\}_* \vdash A_1 \vee A_2$ is provable.

(5) A is $A_1 \rightarrow A_2$ and the proof is similar to the case (4).

(6) A is $\exists xA_1$ and $\mathbf{M} \models_\sigma \exists xA_1$ for every assignment σ. By the semantics of $\exists$, there exists an $a \in \mathbb{M}$ such that $(A_1)_{\mathbf{M}[\sigma[x:=a]]} = \mathrm{T}$. By the definition of $Th(\mathbf{M})$, there exists a $t \in H$ and an assignment σ such that $(t)_{\mathbf{M}[\sigma]} = a$. By the substitution lemma

$$(A_1[t/x])_{\mathbf{M}[\sigma]} = (A_1)_{\mathbf{M}[\sigma[x:=(t)_{\mathbf{M}[\sigma]}]]} = (A_1)_{\mathbf{M}[\sigma[x:=a]]} = \mathrm{T}.$$

By the hypothesis of the structural induction, $\{\Gamma_n\}_* \vdash A_1[t/x]$ is provable. Hence the $\exists$-*R* rule of the **G** system indicates that $\{\Gamma_n\}_* \vdash \exists xA_1$ is provable.

(7) A is $\forall xA_1$. By the semantics of $\forall$, $(A_1)_{\mathbf{M}[\sigma[x:=a]]} = \mathrm{T}$ for every $a \in H_{\mathscr{L}}(\mathbf{M})$ and every σ. It has been proved that

$$(A_1[y/x])_{\mathbf{M}[\sigma'[x:=a]]} = (A_1)_{\mathbf{M}[\sigma[x:=a]]} = \mathrm{T}, \ \ y \notin V_{app}(A_1).$$

By the hypothesis of the structural induction, $\{\Gamma_n\}_* \vdash A_1[y/x]$ is provable. By the $\forall$-*R* rule of the **G** system, $\{\Gamma_n\}_* \vdash \forall xA_1$ is provable.

(8) $A = \neg A_1$. Then A_1 may have several forms as $B \wedge C, B \vee C, \neg B, B \to C, \exists xB(x)$, $\forall xB(x)$. In this case the proof of $\neg A_1$ can be reduced to proving the lemma for the corresponding decomposed formulas in the following table:

A_1	$B \wedge C$	$B \vee C$	$\neg B$	$B \to C$	$\forall xB$	$\exists xB$
$\neg A_1$	$\neg B \vee \neg C$	$\neg B \wedge \neg C$	B	$B \wedge \neg C$	$\exists x \neg B$	$\forall x \neg B$

According to (1)–(7) above, it can be proved that for every case in the above table, $\{\Gamma_n\}_* \vdash A$ is provable.

By structural induction, for every sentence A, if $\mathbf{M} \models A$, then $\{\Gamma_n\}_* \vdash A$ is provable. □

The above lemma immediately yields the following corollary.

Corollary 9.1. *Under the conditions of Lemma 9.3, if* $\forall xA \in Th(\mathbf{M})$*, then* $\{\Gamma_n\}_* \vdash \forall xA$ *is provable.*

9.5 Convergence of the proscheme GUINA

In this section we prove that the proscheme GUINA possesses convergence. In what follows let us prove the convergence of the proscheme GUINA.

Theorem 9.1 (convergence). *Let* $\mathscr{L}$ *be a first-order language with* $\mathbf{M}$ *being an arbitrary model of* $\mathscr{L}$ *and* Γ *being a finite formal theory of* $\mathscr{L}$*. If the inputs of the proscheme* GUINA *are the complete sequence* $\Omega_{\mathbf{M}}$ *of basic sentences and the initial theory* Γ*, and the output version sequence of* GUINA *is* $\{\Gamma_n\}$*, then the sequence* $\{\Gamma_n\}$ *converges, and*

$$\lim_{n\to\infty} Th(\Gamma_n) = Th(\mathbf{M}).$$

Proof. We prove this theorem in the following steps.

(1) We first prove that $Th(\mathbf{M}) \subseteq \{Th(\Gamma_n)\}_*$. It suffices to prove that for every formula A, if $A \in Th(\mathbf{M})$, then $A \in \{Th(\Gamma_n)\}_*$. We prove this by induction on the structure of A:

(a) A is an atomic sentence. Since $A \in Th(\mathbf{M})$ and A is interpreted as a positive instance in $\mathbf{M}$, $A \in \Omega_{\mathbf{M}}$. Suppose that A is P_N. By the definition of GUINA, P_N is a formal consequence of Γ_N, a new axiom of Γ_N, or a formal refutation of Γ_N. In any case, $P_N \in Th(\Gamma_{N+1})$. According to the design of the sets Δ and Θ, when $n > N$, $P_N \in Th(\Gamma_n)$. That is, $A \in \{Th(\Gamma_n)\}_*$.

(b) A is the negation of an atomic sentence. A is interpreted as a negative instance in $\mathbf{M}$. Suppose that A is $\neg P_N$ and $\neg P_N \in \Omega_{\mathbf{M}}$. By the definition of GUINA and using the same proof as (a), we know that $A \in \{Th(\Gamma_n)\}_*$.

(c) A is $P \vee Q$. According to the semantics of $\vee$, at least one of $P \in Th(\mathbf{M})$ and $Q \in Th(\mathbf{M})$. Assume that the former holds. By the hypothesis of the structural induction, we know that $P \in \{Th(\Gamma_n)\}_*$. Then according to the formal inference rule on $\vee$, we have $P \vee Q \in Th(\{Th(\Gamma_n)\}_*)$. That is, $A \in \{Th(\Gamma_n)\}_*$.

(d) Similarly we can prove the case when A is $P \wedge Q$ or A is $P \rightarrow Q$.

(e) A is $\exists x P(x)$ and $A \in Th(\mathbf{M})$. According to the semantics of $\exists$, there exists a $t \in H$ such that $P[t] \in Th(\mathbf{M})$. By the hypothesis of the structural induction, $P[t] \in \{Th(\Gamma_n)\}_*$. Then according to the $\exists$-R rule, $\exists x P(x) \in Th(\{Th(\Gamma_n)\}_*)$. That is, $A \in \{Th(\Gamma_n)\}_*$.

(f) A is $\forall x P(x)$ and $A \in Th(\mathbf{M})$. The conclusion can be proved by using Corollary 9.1.

(g) A is $\neg Q$ and $A \in Th(\mathbf{M})$. Since the proof for basic sentences has been given in (a) and (b), we can assume that Q is not a basic sentence. Hence Q can only be $B \wedge C$, $B \vee C$, $\neg B$, $B \rightarrow C$, $\forall x B$ or $\exists x B$ with B and C being two sentences of $\mathscr{L}$. Thus the forms of $\neg Q$ can be listed as in the following table:

Q	$B \wedge C$	$B \vee C$	$\neg B$	$B \rightarrow C$	$\forall x B$	$\exists x B$
$\neg Q$	$\neg B \vee \neg C$	$\neg B \wedge \neg C$	B	$B \wedge \neg C$	$\exists x \neg B$	$\forall x \neg B$

Applying the method used in (b)–(f), we can prove that every item in the second row of the above table belongs to $\{Th(\Gamma_n)\}_*$. Thus $A \in \{Th(\Gamma_n)\}_*$.

By structural induction, $Th(\mathbf{M}) \subseteq \{Th(\Gamma_n)\}_*$ is proved.

(2) Next we prove that $\{Th(\Gamma_n)\}^* \subseteq Th(\mathbf{M})$ holds. Suppose that there exists a sentence A such that $A \in \{Th(\Gamma_n)\}^*$ and $A \notin Th(\mathbf{M})$. According to Lemma 4.1, since $Th(\mathbf{M})$ is complete, $\neg A \in Th(\mathbf{M})$. Since $Th(\mathbf{M}) \subseteq \{Th(\Gamma_n)\}_*$, there must exist an N such that for $m > N$, $\neg A \in Th(\Gamma_m)$. Furthermore, since $A \in \{Th(\Gamma_n)\}^*$, there exists a subsequence $\{n_k\}$ such that $A \in Th(\Gamma_{n_k})$ for every natural number k. Thus, when $n_k > N$, both A and $\neg A$ belong to $Th(\Gamma_{n_k})$. This is a contradiction by Lemma 9.2, the output Γ_{n_k} of GUINA* is consistent. Hence $A \in Th(\mathbf{M})$.

The above two steps have proved that $\{Th(\Gamma_n)\}^* \subseteq Th(\mathbf{M}) \subseteq \{Th(\Gamma_n)\}_*$. Thus

$$\{Th(\Gamma_n)\}_* = \{Th(\Gamma_n)\}^* = Th(\mathbf{M}).$$

The theorem is proved. □

Theorem 9.1 can be interpreted as: for an arbitrary given scientific problem $\mathbf{M}$, the proscheme GUINA, starting from any conjecture, improves it by processing instances one by one as detailed above.

In the process of sequentially examining all the positive and negative instances of $\Omega_{\mathbf{M}}$, the sequence of theory closures of the versions output by GUINA approaches, in the limit, the set $Th(\mathbf{M})$.

9.6 Commutativity of the proscheme GUINA

In this section we prove that the version sequence output by the proscheme GUINA possesses the commutativity between limit operation and formal inference. That is, this proscheme is commutative.

Theorem 9.2 (commutativity). *Let $\mathscr{L}$ be a first-order language with* **M** *being an arbitrary model of $\mathscr{L}$ and Γ being a finite formal theory of $\mathscr{L}$. If the inputs of the proscheme* GUINA *are the complete sequence $\Omega_{\mathbf{M}}$ of basic sentences and the initial theory Γ, and the output version sequence of* GUINA *is $\{\Gamma_n\}$, then the sequence $\{\Gamma_n\}$ converges, and*

$$\lim_{n\to\infty} Th(\Gamma_n) = Th(\lim_{n\to\infty} \Gamma_n).$$

Proof. Since it has already been proved in Theorem 9.1 that $\lim_{n\to\infty} Th(\Gamma_n) = Th(\mathbf{M})$, now it suffices to prove that

$$\{Th(\Gamma_n)\}_* \subseteq Th(\{\Gamma_n\}_*) \subseteq Th(\{\Gamma_n\}^*) \subseteq \{Th(\Gamma_n)\}^*.$$

This can be done in two steps.

(1) We first prove that $Th(\{\Gamma_n\}^*) \subseteq \{Th(\Gamma_n)\}^*$. For every $A \in Th(\{\Gamma_n\}^*)$, $\{\Gamma_n\}^* \vdash A$ is provable. According to the compactness theorem, there exists a finite sequence

$$\{A_{n_1},\ldots,A_{n_k}\} \in \{\Gamma_n\}^* \text{ such that } \{A_{n_1},\ldots,A_{n_k}\} \vdash A$$

is provable. By the definition of $\{\Gamma_n\}^*$, $A_{n_i} \in \{\Gamma_n\}^*, i = 1,\ldots,k$. This means that there exists a subsequence of Γ_n:

$$\Gamma_{n_{i_1}},\ldots,\Gamma_{n_{i_j}},\ldots, \text{ where } j \text{ is any natural number.}$$

For any given $i \leqslant k$, A_{n_i} is an element of each $\Gamma_{n_{i_j}}$ in this sequence and thus is an element of $Th(\Gamma_{n_{i_j}})$. Hence $A_{n_i} \in \{Th(\Gamma_n)\}^*$, i.e.,

$$\{A_{n_1},\ldots,A_{n_k}\} \subset \{Th(\Gamma_n)\}^*.$$

According to Theorem 9.1, $\{Th(\Gamma_n)\}^* = Th(\mathbf{M})$ and thus $\{Th(\Gamma_n)\}^*$ is a theory closure. Hence

$$A \in Th(\{A_{n_1},\ldots,A_{n_k}\}) \subset \{Th(\Gamma_n)\}^*.$$

(2) Next we prove that $\{Th(\Gamma_n)\}_* \subseteq Th(\{\Gamma_n\}_*)$. For every $A \in \{Th(\Gamma_n)\}_*$, Theorem 9.1 indicates that $A \in Th(\mathbf{M})$.

Then Lemma 9.3 indicates that $\{\Gamma_n\}_* \vdash A$ holds, i.e., $A \in Th(\{\Gamma_n\}_*)$. Thus $\{Th(\Gamma_n)\}_* \subseteq Th(\{\Gamma_n\}_*)$. □

Corollary 9.2 (Reliability of GUINA). *For any complete sequence $\Omega_{\mathbf{M}}$ of basic sentences of any given problem* **M** *and any initial formal theory Γ, the proscheme* GUINA *is reliable.*

Proof. This corollary follows directly from Theorems 9.1 and 9.2. □

9.7 Independence of the proscheme GUINA

In this section we prove that if the initial conjecture Γ input to the proscheme GUINA is the empty set, then the output version sequence $\{\Gamma_n\}$ of GUINA possesses independence. That is, the proscheme GUINA is independent if Γ is the empty set.

Theorem 9.3 (Independence). *Let $\mathscr{L}$ be a first-order language with $\mathbf{M}$ being an arbitrary model of $\mathscr{L}$ and Γ being a finite formal theory of $\mathscr{L}$. Let the inputs of the proscheme GUINA be the complete sequence $\Omega_{\mathbf{M}}$ of basic sentences and the initial theory Γ, and the output version sequence of GUINA be $\{\Gamma_n\}$. If Γ is the empty set, then for every $n > 0$, Γ_n is an independent theory, and so is $\lim_{n\to\infty} \Gamma_n$.*

Proof. Let $\Gamma_1 = \Gamma$. The proof proceeds with two steps.

(1) We first prove that for every $n > 0$, Γ_n is an independent theory. We use the induction method to prove the conclusion. Suppose that the complete sequence $\Omega_{\mathbf{M}}$ of basic sentences is

$$P_1, \ldots, P_n, \ldots.$$

For simplicity, in what follows we abbreviate $P_n[t_m]$ as $P_n[t]$ with $t \in H$.

First, by the definition of GUINA, $\Gamma_2 = \{\forall x P_1\}$. This is an independent theory.

Suppose that Γ_n is an independent theory. By the definition of the proscheme GUINA, there are only four possible cases as follows.

(a) $\Gamma_n \vdash P_n[t]$ is provable. In this case $\Gamma_{n+1} = \Gamma_n$. Hence Γ_{n+1} is an independent theory.

(b) $\Gamma_n \vdash \neg P_n[t]$ is provable. In this case GUINA selects a maximal subset Λ of Γ_n that is consistent with $P_n[t]$. Λ is also an independent theory because Γ_n is an independent theory. By the definition of GUINA, Γ_{n+1} can be generated in two steps.

Firstly, we need to combine $P_n[t]$ with Λ. Since the basic sentence $P_n[t]$ is a new axiom of Λ, $\Lambda \cup \{P_n[t]\}$ is still an independent theory.

Secondly, GUINA needs to examine the elements in Θ_n and Δ_n individually and then take the union of $\Lambda \cup \{P_n[t]\}$ and those sentences P_{n_j} possibly lost due to the selection of Λ. Using the same method as in the above, we can prove that each time after P_{n_j} is incorporated, the sentence set obtained is still an independent theory. Thus Γ_{n+1} is an independent theory.

(c) Neither $\Gamma_n \vdash P_n[t]$ nor $\Gamma_n \vdash \neg P_n[t]$ is provable and $P_n[t] \bowtie \Gamma_n$. According to the definition of GUINA, $P_n[t]$ is just the first instance of the predicate P_n which GUINA encounters. In this case $\Gamma_{n+1} = \Gamma_n \cup \{\forall x P_n\}$, and $\Delta_{n+1} = \Delta_n \cup \{P_n[t]\}$. Thus Γ_{n+1} is an independent theory.

(d) Neither $\Gamma_n \vdash P_n[t]$ nor $\Gamma_n \vdash \neg P_n[t]$ is provable and $P_n[t] \not\bowtie \Gamma_n$. According to the definition of GUINA, there already exist basic sentences such as $\neg P_n[t']$ in $Th(\Gamma_n)$. Since neither $\Gamma_n \vdash P_n[t]$ nor $\Gamma_n \vdash \neg P_n[t]$ is provable and $\Gamma_{n+1} = \Gamma_n \cup \{P_n[t]\}$, $P_n[t] \notin Th(\Gamma_n)$ but $P_n[t] \in Th(\Gamma_{n+1})$. By definition, Γ_{n+1} is an independent theory.

The above four cases show that if Γ_n is an independent theory, then Γ_{n+1} is still an independent theory after GUINA's processing. Thus every Γ_n output by GUINA is an independent theory.

(2) Because every Γ_n is an independent theory and $\{\Gamma_n\}$ is convergent, according to Lemma 8.2, $\lim_{n\to\infty} \Gamma_n$ is also an independent theory. □

From this theorem and the results proved in Sections 9.5, 9.6, and 9.7, we can see that if the initial conjecture is the empty set, then the proscheme GUINA is an ideal proscheme.

Corollary 9.3. *If the initial formal theory* Γ *is the empty set, then the proscheme* GUINA *is not only reliable, but also ideal.*

Proof. The conclusion follows immediately from Theorems 9.1, 9.2, and 9.3. □

In summary, we have shown that inductive inference is a rational mechanism for evolution of theories about a particular domain. Inductive inference is the mechanism by which we make a formal passage from particular observations to conjectured general principles. The result of applying inductive inference is the generation of a new version of a theory.

The rationality of inductive inference is demonstrated by the fact that there is a reliable proscheme that can take any initial conjecture and whose output version sequence will always converge to $Th(\mathbf{M})$, the set of all true sentences in $\mathbf{M}$.

What this means is that, even if the initial conjecture is wrong, the inductive inference system will automatically revise it, making new generalizations from the observed facts, in such a way that the version sequence approaches the full truth about the domain being described.

We have also proved that GUINA is commutative, and this, together with convergence, means that it is a reliable proscheme that can be used practically, with finite sets of axioms, to axiomatize the knowledge of the domain.

Furthermore, if we start with no initial conjectured theory, GUINA will combine the observed facts with generalizations in such a way as to make, at every step, a consistent, independent version of the theory about $\mathbf{M}$. The limit of this process is a complete independent axiomatization of $Th(\mathbf{M})$.

The conclusion of this chapter shows that for an inductive inference system, if one can find a proscheme F such that for every scientific problem $\mathbf{M}$, the proscheme is always reliable, then the inductive process is rational.

Chapter 10

Workflows for Scientific Discovery

A principal thesis of this book is that mathematical logic is not only an abstract mathematical theory, but can also provide a practical framework for scientific research in the information society.

It shows us how to describe, analyze, and reason about knowledge in a way that can be, to some extent, 'mechanized'. In addition, the process of axiomatization, presented in the last half of this book, leads to a rational and computer-assisted workflow for the process of research. This workflow can also be used as a reliable high-level framework for the development of computer software and hardware.

The aim of this chapter is to explain this workflow for research and thus to make clear how to practically use the theories introduced in this book. Before doing this, we will review the fundamental theories of mathematical logic and axiomatization that we have presented in the previous nine chapters.

In Section 10.1, we explain the three language environments as contexts in which to study mathematics and natural science, with a few examples. In Section 10.2, we give the six basic principles that meta-language environments should obey. In Section 10.3, we review the core idea of axiomatization used in mathematical research. In Section 10.4, we summarize the main concepts and theorems of first-order languages, which we shall call the *theoretical framework of first-order languages*. On the basis of this framework, we finally describe in Section 10.5 a basic reliable workflow for research in informatics and natural science.

10.1 Three language environments

We talk about the theories of mathematical logic using three contexts, or language environments. As we have seen, it is important to be clear in which environment our discussion is taking place or else our reasoning can become paradoxical.

We have already clearly defined two of these environments, the object language and the model. The purpose of this and the next section is to clarify the third context that we use, the meta-language environment.

In the meta-language environment, we mainly use natural language to talk about theories. In this environment, we refer to and call on previously established theories of mathematics and natural science; we detail the data from observations and experiments, describe observed phenomena and make conjectures about universal principles.

For example, when defining a first-order language and its models, we use the concepts of sets, maps and their properties, which are all part of its meta-language environment.

For another example, consider Gödel's incompleteness theorem. The proof of this theorem not only involves the first-order language $\mathscr{A}$ but also uses its model **N**, and the proof uses reasoning methods, such as proof by contradiction and modus ponens. These reasoning methods are neither contained in the first-order language $\mathscr{A}$ nor used only by the model **N**. This indicates that the proof of Gödel's theorem is carried out in the meta-language environment of $\mathscr{A}$ and **N**.

Therefore, when we choose a domain of knowledge to study, we must define what first-order language can express its structure, what mathematical structures embody its truths and what meta-language environment is necessary in order to reason about the relation between language and models. Let us look at the following four examples to clarify this statement.

Example 10.1 ($\mathscr{A}$, N, and $\mathfrak{N}$). The elementary arithmetic language $\mathscr{A}$ is a first first-order language introduced in this book, its domain is $\mathbb{N}$, and its model is **N**.[1]

Object language. $\mathscr{A}$ is defined on the following sets: the set $\{0\}$ of a constant symbol, the set $\{S,+,\cdot\}$ of function symbols, and the set $\{<\}$ of a predicate symbol as well as the set of variable symbols, the set of logical connective symbols, the set of quantifier symbols, the set of the equality symbol, and the set of parentheses. The last five sets of symbols are the same for every first-order language. $\mathscr{A}$ defines two types of syntactic objects, i.e., terms and logical formulas, which both are symbol strings, generated according to their respective syntactic rules.

A formal theory may be defined for each first-order language. Formal theories are the fundamental objects of study for first-order languages. In fact, each first-order language is defined for some formal theory and its versions. The fundamental object of our study for $\mathscr{A}$ is the theory of elementary arithmetic Π, which consists of ten laws, which are described by sentences, and is a formal theory of $\mathscr{A}$.

Model. The model **N** of $\mathscr{A}$ is a pair $(\mathbb{N},I)$. $\mathbb{N}$ is a domain and it is a mathematical system over the set of natural numbers, which contains arithmetic operations, recursive functions, and *P*-procedures. s is the "plus 1" function over $\mathbb{N}$, i.e., $s(x) = x+1$. $+$ and $\cdot$ denote respectively the addition function and multiplication function over $\mathbb{N}$. $<$ is the "less than" relation over $\mathbb{N}$. The interpretation map $I:\ \mathscr{A} \to \mathbb{N}$ maps the special symbols of $\mathscr{A}$ onto the mathematical entities, functions and relations in $\mathbb{N}$:

$$I(0) = 0,\ \ I(S) = s,\ \ I(+) = +,\ \ I(\cdot) = \cdot,\ \ I(<) = <\,.$$

In the first equation of the above definition, note the distinction between the 0 on the left side of the equality above (which is a symbol of $\mathscr{A}$) and the 0 on the right side (which is the natural number 0). A similar distinction applies to all the other equations.

After the model is determined, every sentence is interpreted as a proposition in the model **N** and this proposition is either true or false in $\mathbb{N}$. For instance, every sentence

[1] As in Chapter 2, we call **N** the *structure* of $\mathscr{A}$ and $(\mathbf{N},\sigma)$ the *model* of $\mathscr{A}$, where σ is an assignment map. Since formal theory closures do not involve any free variable, we make no distinction between structures and models when discussing problems related to formal theories. For the sake of brevity, **N** is also called the *model* of $\mathscr{A}$.

in the formal theory Π is interpreted as an axiom about natural numbers and it is a true proposition in $\mathbb{N}$.

Meta-language environment. In defining $\mathscr{A}$ and the model **N**, the concepts that we have used about sets and maps, including the symbol $=$ used in defining the interpretation map, are all constituent parts of the meta-language environment of $\mathscr{A}$ and **N**. The explanations of the formal theory Π, the discussion of the theorems proved in this book and the comments about examples are also part of the meta-language environment. We denote this environment by $\mathfrak{N}$.

$\mathfrak{N}$ also includes the logical connectives "negation of ...," "... and ...," "... or ...," and "if ..., then ...", the quantifiers "for all ..." and "there exist ...", and the logical inference rules such as modus ponens and proof by contradiction. They are commonly used in all meta-language environments. The proofs of lemmas and theorems related to $\mathscr{A}$ and **N** are all mathematical proofs and they are also constituent parts of the meta-language environment $\mathfrak{N}$. Obviously, without the meta-language environment $\mathfrak{N}$, it would not be possible to study $\mathscr{A}$ and the model **N** and the authors would not be able to communicate with their readers.

We cannot define the meta-language environment $\mathfrak{N}$ by syntactic rules as we do with $\mathscr{A}$, but we know that a meta-language environment must obey certain basic principles such as the principle of excluded middle and those about the semantics of logical connectives. These basic principles are prerequisites for the study of first-order languages and their models. They are widely accepted by the academic community. The purpose of the next section is to present the basic principles that meta-language environments obey.

Example 10.2 (Newtonian physics). In Example 6.2, we discussed the evolution of physics. From an abstract point of view, we can use a sentence of a first-order language to describe the Galilean transformation. Let this first-order language be $\mathscr{M}$, which is the object language in this example and assume that Newtonian physics can be described by the formal theory Γ of $\mathscr{M}$. In the terminology of first-order languages, Γ may be called the *formal theory* that describes physics.

$$\Gamma = \{V, N_1, N_2, N_3, E\}$$

is the object of study about $\mathscr{M}$. The sentence

$$V = \forall x(B(x) \to A(x))$$

describes the Galilean transformation. In Landau's *Mechanics* [1960], laws about classical mechanics and their mathematical proofs are regarded as the domain $\mathbb{M}$ of $\mathscr{M}$, the predicate $B(x)$ is interpreted in $\mathbb{M}$ as "x is a rigid body," and the formula $A(x)$ is interpreted in $\mathbb{M}$ as "if the velocity of the rigid body x with respect to the coordinate system K is v and the velocity of the coordinate system K with respect to the coordinate system K' is w, then the velocity of x with respect to the coordinate system K' is $v+w$." The interpretation of the sentence $\forall x(B(x) \to A(x))$ in $\mathbb{M}$ is just the Galilean transformation. $\mathbb{M}$ together with the interpretation map forms a model **M** of $\mathscr{M}$.

The first part of the undergraduate physics textbook [Halliday, 2000] contains many examples and explanations, which clarify the model of mechanics set out by Landau and thus can be regarded as part of the meta-language environment of $\mathscr{M}$ and **M** denoted by $\mathfrak{M}$.

When performing mathematical reasoning in $\mathfrak{M}$, we use inference rules about logical connectives from the meta-language environment, which have the same meaning as in the meta-language environment of Example 10.1.

The concepts of first-order languages, their domains, and meta-language environments have been widely used in computer science. They have played a guiding role in the design and development of computer software.

Generally speaking, if the object language is a formal language, then the concepts and methods of study about first-order languages, their models, and meta-language environments presented in the first five chapters can all be generalized to the study of this object language. In what follows, we take the C language as an example to illustrate this.

Example 10.3 (The language C, the C compiler, and the C documentation). The C language is a formal language and is the object language in this example. C programs are the fundamental objects of our study of C. Let $\mathscr{C}$ denote the set composed of all C programs.

Let the compiler of C be I_C. I_C compiles each C program into a segment of code that is executable on a computer. Let $\mathbb{C}$ denote the set composed of all such segments of executable code, called the *domain of code*. The compiler I_C can be regarded as an interpretation map $I_C : \mathscr{C} \to \mathbb{C}$, because I_C maps each C program to an element of $\mathbb{C}$, i.e., a segment of code in $\mathbb{C}$. Following the terminology of first-order languages, the pair $(\mathbb{C}, I_C)$ can be regarded as a model of C.

The C manual, the documentation of the C compiler, and the comments in the C programs in $\mathscr{C}$ are part of the meta-language environment of C and its model $(\mathbb{C}, I_C)$. This meta-language environment is denoted as $\mathfrak{C}$.

In $\mathfrak{C}$, knowledge about the C language and its model is represented by propositions, containing all of the usual logical connectives and quantifiers. The inference rules for these logical connectives and quantifiers are all contained in $\mathfrak{C}$. They are the same as the inference rules used in the meta-language environment of first-order languages.

Note that the C language is not a first-order language, it is a formal language. However, the concepts of model and meta-language have exactly the same meaning as we defined for first-order languages. In fact, in computer science and software engineering research, the terminology of model is used extensively. Hereafter, as long as the object language is a formal language, distinguishing the object language, its model and meta-language is essential to research in the information society.

The concepts of an object language, its model, and meta-language environment are relative to the domain being studied. An object language in one situation can be a model, or even a meta-language, in another context. In this sense, the role of a language has dual nature. Let us illustrate this with the following example.

Example 10.4 (BASIC, BASIC interpreter, and the language C). BASIC is a programming language and it is a formal language in this example. For BASIC, the objects of study are BASIC programs. Let I_B be the interpreter of BASIC, implemented by using the C language. I_B can be regarded as an interpretation map that interprets each BASIC program as a C program. Let $\mathscr{C}$ be the set composed of all C programs. $\mathbf{B} = (\mathscr{C}, I_B)$ forms a model of the BASIC language and the C language itself forms part of the meta-language environment of the BASIC language and its model **B**.

The difference between this example and Example 10.3 is: in Example 10.3, C is the object language, but in this example it becomes part of the meta-language environment of the BASIC language. This example shows the relativity and duality of object languages, models, and meta-language environments.

After the above four examples have been discussed, the reader would ask what are the object language, model, and meta-language environment of this book. In fact, Definition 1.1 specifies the formal language $\mathscr{L}$ of this book. We have pointed out that each first-order language is defined for describing the knowledge of a specific domain and Definition 1.1 gives the general definition of first-order languages. Therefore, $\mathscr{L}$ can be considered as a representative of first-order languages. Both $\mathscr{A}$ in Example 10.1 and $\mathscr{M}$ in Example 10.2 are specialized first-order languages and can be considered as instances of $\mathscr{L}$. The model **M** of $\mathscr{L}$ is a pair $(\mathbb{M}, I)$. Definition 2.3 gives a general definition of models of first-order languages and $(\mathbb{M}, I)$ is a representative of the models of first-order languages. Both $(\mathbb{N}, I)$ in Example 10.1 and $(\mathbb{M}, I)$ in Example 10.2 are specialized models and can be considered as instances of **M**. In defining $\mathscr{L}$ and the model **M**, the concepts about sets and maps used in this book, including the $=$ symbol used in defining interpretation maps and explanations for the theory of natural numbers and the examples in this book, are a part of the meta-language environment of $\mathscr{L}$. We may use $\mathfrak{L}$ to denote this meta-language environment.

10.2 Basic principles of the meta-language environment

The meta-language environment cannot be defined using a similar method to that which we used to define first-order languages. However, the meta-language environment of first-order languages must obey certain basic principles. The purpose of this section is to introduce these principles.

1. Principle of environment

From the examples given in Section 10.1, we can abstract a fundamental principle, called the *principle of environment*:

Principle 10.1 (Principle of environment). *Each first-order language, as well as its model, is defined and explained in a meta-language environment and theorems related to this first-order language and its model are proved in this meta-language environment.*

In the previous examples, the language of elementary arithmetic $\mathscr{A}$ and its model **N** are defined and elaborated in the meta-language environment $\mathfrak{N}$. Theorems related to both of them, such as Gödel's theorems, are proved in $\mathfrak{N}$. The first-order language $\mathscr{M}$ and its model **M** about Newtonian physics are defined and elaborated in the meta-language environment $\mathfrak{M}$ and theorems related to them such as Kepler's three laws are proved in $\mathfrak{M}$. The language C and its model $(\mathbb{C}, I_C)$ are defined in the meta-language environment $\mathfrak{C}$ and C programs are interpreted and explained in the meta-language environment $\mathfrak{C}$.

2. Principle of excluded middle

In Chapter 2, we made a basic assumption on the domain of a first-order language, i.e., the principle of excluded middle. Namely, each proposition in a domain is either true or false and there is no other choice. This is also a basic principle that a meta-language environment must obey.

Principle 10.2 (Principle of excluded middle). *Each proposition in the meta-language environment of a first-order language is either true or false.*

The principle of the excluded middle is not a universal truth, it is an assumption like the axiom of parallels in plane geometry. Moreover, we only assume that it is true for the meta-language environment and the model.

Since the late 19th Century, logicians have divided into two camps. One branch of mathematical logic accepts the principle of excluded middle, and is called *classical logic*, while the other branch does not accept it and is called *intuitionistic logic*.

In this book we accept the principle because, without it, the method of proof by contradiction cannot be used and thus Gödel's theorems cannot be proved. In assuming this principle, we follow the mainstream of scientific research.

3. Principle of logical connectives

The logical connectives of any first-order language

$$\{ \neg, \wedge, \vee, \rightarrow, \leftrightarrow \}$$

are interpreted in the domain and meta-language environment of this language as "negation of ...," "... and ...," "... or ...," "if ..., then ..." and "... if and only if ..."

The semantics of logical connectives was given in Definition 2.7 of Chapter 2 using truth functions. Here it should be noted that Definition 2.7 is independent not only of the set of constant symbols, the set of function symbols, and the set of predicate symbols, but also of the domain of each first-order language. *Therefore, the semantics of the logical connectives is defined in the meta-language environment*.

According to the principle of excluded middle, each proposition in the meta-language environment of a first-order language is either true or false. Hence we can define the semantics of a proposition using the same truth table as in Definition 2.7 by replacing the logical connective symbols with the logical connectives in the meta-language environment:

Definition 10.1 (Semantics of logical connectives). Let the variables of the truth functions be X and Y, which denote the truth values of propositions in the meta-language environment. The negation of X is defined by the following truth table:

X	T	F
negation of X	F	T

The binary functions "X or Y," "X and Y," "if X, then Y," and "X if and only if Y" are defined by the following truth table:

X	Y	X or Y	X and Y	if X, then Y	X if and only if Y
T	T	T	T	T	T
T	F	T	F	F	F
F	T	T	F	T	F
F	F	F	F	T	T

The above definition gives the semantics of the logical connectives in the meta-language environment, from which we obtain the third basic principle:

Principle 10.3 (Principle of logical connectives). *In the meta-language environment of a first-order language, the semantics of the logical connectives is determined by Definition 10.1.*

According to the principle of logical connectives, the following corollary holds.

Corollary 10.1. *The logical connective symbols in first-order languages of classical mathematical logic, as well as the logical connectives in their models, correspond one-to-one to the logical connectives in the meta-language environment, and they have the same semantics.*

In Chapter 2, we pointed out that in natural language environments, logical connectives might be ambiguous. For example, the connective "or" might be exclusive or inclusive. The semantics of inclusive "or" is that given in Definition 10.1 and the semantics of exclusive "or" is: "X or Y" is true if one and only one of X and Y is true. Acceptance of Definition 10.1 specifies that "or" does not have exclusivity in the meta-language environment of first-order languages.

It should be pointed out that there are some formal languages whose logical connective symbols have different semantics to their corresponding logical connectives in the meta-language environment.

For instance, in three-valued logic the semantics of the logical connective symbols are defined by three-valued truth functions. However, theorems such as the soundness of inference rules for three-valued logic are proved in its meta-language environment, and such statements in this environment are either true or false and there cannot be a third choice. This shows that, in the meta-language environment of three-valued logic, the semantics of the logical connectives is two-valued and determined by Definition 10.1, even though the semantics of the object language has three truth values.

At the end of Chapter 3, we proved the derived rules consistent with the **G** system according to the semantics of the logical connective symbols. Based on the principle of logical connectives, we can also obtain derived rules about logical connectives.

Since the semantics of the logical connective symbols and that of the logical connectives are the same, these derived rules are the interpretations of the derived rules of the logical connective symbols in the meta-language environment. These derived rules are, proof by contradiction, proof by case analysis, modus ponens, etc. used in mathematical proof, so they are all valid in the meta-language environment.

4. Church-Turing thesis

In Chapter 4, we discussed the Church-Turing thesis, which is also a basic principle of the meta-language environment of first-order languages.

Principle 10.4 (Church-Turing thesis). *All acceptable definitions of computability are mutually equivalent.*

Having made the distinction between object languages, models, and the meta-language environment, the following example illustrates the real intention of the thesis.

ML is a functional programming language that makes it easy to solve problems by using recursive functions. Let the compiler of ML be I_{ML}, which interprets each recursive function defined by ML as a C program. Here we take the set of all halting C programs as the domain denoted by $\mathbb{C}$. The pair $(\mathbb{C}, I_{\mathrm{ML}})$ is a model of the formal language ML. In this case, we say that the language ML is *C-implementable*.

On the other hand, we say that the C language is *ML-implementable* if the C language is regarded as the object language and ML is used to implement an interpreter I_{C}, such that every halting C program uses the the corresponding ML function to do the same work, and the set $\mathbb{F}$ consisting of all such ML functions is taken as the domain. This illustrates the general principle that recursive functions and *P*-procedures are mutually implementable.

5. Principle of observability

The problems of natural science that can be described by first-order languages should all be related to observable phenomena:

Principle 10.5 (Principle of observability). *Experiments and observations can be made on natural phenomena and the results of experiments and observations can be described by digital data.*

The information era has seen the development of digital measuring instruments, which acquire information from natural phenomena and convert it into digital data. This gives us precise boundary conditions, so that natural phenomena can be modeled accurately. The principle means that we can abstract propositions from the acquired data, which is a pre-requisite for scientific research.

6. Principle of Occam's razor

The sixth principle of the meta-language environment of first-order languages is Occam's razor. This book has shown that there are two cases in which an axiom system must

change. One is that the axiom system meets a refutation by facts. The other is that experiments support a proposition that cannot be deduced from the axiom system. In either case, improvement of the axiom system should not exceed what is necessary. This is the principle of Occam's razor for meta-language environments.

Principle 10.6 (Principle of Occam's razor). *Every axiom system is improvable, but the improvement cannot exceed what is necessary.*

These six basic principles of the meta-language environment are the foundation for studying the theory of first-order languages.

10.3 Axiomatization

The use of axiomatization began with the ancient Greek mathematician Euclid. He collected together the current knowledge of geometry at the time and, in his *Elements*, established an axiom system for plane geometry. Euclid's axiom system was the first relatively complete axiom system in mathematics. This system uses propositions to describe geometric knowledge and uses logical inference rules to prove geometric propositions using only the axioms as premise.

Since then it has become widely used as a way of ensuring the soundness of a mathematical theory. Every branch of mathematics now has a foundation of basic axioms.

Generally speaking, axiomatization has four constituent parts: definition of concepts, statement of propositions, formation of axiom systems and proof of theorems.

(1) Definition of concepts. The knowledge of every domain contains some concepts, which may be divided into basic concepts and composite concepts. Basic concepts are undefined abstract objects and composite concepts are defined by means of some basic concepts or other composite concepts already defined in the theory. For example, in geometry, point, line and plane are basic concepts and triangle, rectangle and polygon are composite concepts.

(2) Statement of propositions. Domain knowledge consists of propositions. Propositions are constructed by combining basic concepts and composite concepts with logical connectives and quantifiers. Logical connectives include "negation of ...," "... and ...," "... or ...," and "if ..., then ..." and quantifiers include "for all ..." and "there exist ..." In propositions, logical relations between concepts are expressed by logical connectives.

(3) Establishment of axiom systems. Propositions may be divided into basic propositions and proved propositions. Basic propositions are usually called *axioms*, or *principles*, or *rules*. They are also called *postulates* in plane geometry.Axioms are those basic propositions that are in accordance with people's experience and intuition and are directly accepted without need of proof. For example, the propositions "only one line can be drawn through any two points" and "only one line can be drawn through any point not on a given line parallel to the given line" are both axioms in plane geometry.

(4) Proof of theorems. In domain knowledge, the truth of propositions other than axioms can be confirmed by mathematical proof. Proved propositions are called *theorems*. A

theorem may also be called a *lemma* or a *corollary* according to its role and importance in proving other propositions.

If we are given a proposition that is not an axiom, we can attempt to prove it by taking the axioms and all existing theorems as premise and then using logical inference rules on the connectives in the proposition to try to deduce it. If we succeed then it is a logical consequence of the axioms.

For every axiom system, one can ask whether it possesses the following five fundamental properties.

1) **Finiteness.** The axiom system contains only finitely many axioms.
2) **Consistency.** The axioms from the axiom system are not contradictory.
3) **Completeness.** For any proposition about the domain knowledge and its negation, one of them must be a theorem of the axiom system.
4) **Decidability.** There exists a computable procedure that can decide in a finite number of steps whether any proposition is a theorem of the axiom system.
5) **Independence.** Each axiom is not a logical consequence of the other axioms in the axiom system.

Axiomatization refers to the method that uses propositions to describe the knowledge of a domain, and

1. takes the formation of the axiom system as the primary objective and the axiom system itself as the premise and starting point for organizing the domain knowledge,
2. performs mathematical proving by using logical inference rules, and
3. establishes logical relations between propositions.

The advantage of axiomatization is that when used to analyze and organize domain knowledge, the method ensures that the reasoning is not paradoxical and gives rigorous mathematical proofs for propositions.

The advantages of axiomatization can only be fully seen when the domain knowledge has reached a high level of maturity. This means that rich data has been accumulated through extensive experiments, this data coincides with the basic concepts in the domain and it supports the propositions, which are used to form the axiom system for the domain.

Axiomatization also has limitations. We know that any axiom system containing arithmetic operations is incomplete and the consistency of the axiom system cannot be proved using the axiom system itself as premise. These are the interpretations of Gödel's theorems in the domain, although to be precise, Gödel's theorems can only be rigorously proved in the framework of first-order languages.

Also, we cannot, in general, establish an axiom system for a domain in one step. It usually requires extensive experiments and repeated verifications. Even after it is well established, the axiom system still needs to be continually verified in practice. This book has shown how to formalize the process of axiomatization.

Nowadays, in mathematics or natural science, the level of axiomatization of the knowledge in a domain has become a criterion for assessing the maturity of the theory. Axiomatization is a milestone in the advance of mankind's theory of knowledge.

10.4 Formal methods

This section is a review of the formal methods presented in this book. The first five chapters described the concepts and results of classical mathematical logic, which were developed and discovered in the last century.

In the last part of the book we studied how to formalize the process of axiomatization of formal theories. We showed that this was possible and introduced the concepts of version sequences and their limits, new axioms and refutation by facts, **R**-calculus and its reachability, soundness and completeness, and proschemes and their reliability.

Thus, the first part of the book analyzed the process of formal inference, while the last part examined the process of formal axiomatization. Together they are called the *theoretical framework of first-order languages*. This framework consists of the following 12 points:

(1) First-order language. Every first-order language is defined on eight sets of symbols. These sets of symbols are divided into two types: sets of object symbols and sets of logical symbols. The set $\mathscr{L}_c$ of constant symbols, the set V of variable symbols, the set $\mathscr{L}_f$ of function symbols, and the set $\mathscr{L}_P$ of predicate symbols are called *sets of object symbols*. The sets of logical symbols include the set C of logical connective symbols, the set Q of quantifier symbols, the set E of the equality symbol, and the set of parentheses. They are the same for every first-order language, while the sets of object symbols are chosen specially to describe different domains (see Section 1.1).

Two kinds of objects, terms and logical formulas, are defined in every first-order language. Each object is defined by syntactic rules, which specify how to combine the symbols in the language. Let us emphasize that these objects are just strings of symbols; they only have meaning after interpretation (see Sections 1.2 and 1.3).

(2) Domain, interpretation, and model. The structure $M = (\mathbb{M}, I)$ is composed of the domain $\mathbb{M}$ and the interpretation map I. The domain $\mathbb{M}$ is a mathematical system and it is the mathematical description of the domain knowledge. The interpretation map I is a one-to-one map from the first-order language to its domain. It interprets terms of the first-order language as constants, variables, and functions in the domain, predicates of the first-order language as relations and concepts, and sentences of the first-order language as propositions in the domain. The model $\mathbf{M} = (M, \sigma)$ is composed of the structure M and the assignment σ and can describe axiom systems in mathematics and natural science (see Sections 2.1 and 2.2).

(3) Formal inference system and formal proof. The formal inference system used in this book is the **G** system, which is composed of axioms, inference rules for logical connective symbols and quantifier symbols, and the cut rule. Every inference rule for a logical connective symbol is a rule of calculus for this symbol and it is interpreted as an inference

rule for the corresponding logical connective in the model. The cut rule is a rule for deleting logical formulas and it is interpreted in the model as: all theorems proved by using the cut rule can be proved directly by using only the inference rules for logical connective symbols and quantifier symbols (see Section 3.1).

The role of the **G** system is to produce formal proofs. The basic object of the **G** system is sequent $\Gamma \vdash A$, where Γ is called the *premise* and A the *formal consequence* of the sequent. A formal proof of a sequent has a tree structure. The root of the tree is the original sequent, each node of the tree is also a sequent, which is an instance of some formal inference rule in the **G** system, and the leaves of the tree are instances of the axiom sequent (see Section 3.2).

Since logical formulas occurring in a sequent are all symbol strings conforming to syntactic rules and the formal inference rules used in a formal proof are all rules of calculus for logical connective symbols and quantifier symbols, every process of formal proof is a process of symbolic calculus, from which a *P*-procedure for the provability of the sequent can be devised. In case the sequent is provable, the *P*-procedure will halt.

(4) Soundness and completeness of the G system. The soundness of the **G** system means that if $\Gamma \vdash A$ is provable, then $\Gamma \models A$, i.e., for any model **M**, so long as the interpretation of Γ in **M** is true, the interpretation of A in **M** is also true (see Section 3.3).

As the proof of $\Gamma \vdash A$ is accomplished by means of formal calculus, the soundness provides the following guarantee: so long as formal inference rules are correctly used in formal proof, if the interpretation of Γ in the domain is true, then so is the interpretation of A. This indicates that in formal proof there is no need to consider the way in which the formula is interpreted in a domain.

The completeness of the **G** system means that if $\Gamma \models A$, i.e., for any model **M**, $\mathbf{M} \models A$ so long as $\mathbf{M} \models \Gamma$, then $\Gamma \vdash A$ is provable. The guarantee provided by the completeness of the **G** system is that any logical consequence of the axiom system in the domain can be obtained by means of the method of formal proof with the **G** system (see Section 3.5).

(5) Formal theories. Formal theories are the basic objects of study for first-order languages. Each formal theory is a set of mutually consistent sentences. Every sentence contained in a formal theory is called a (nonlogical) *axiom* of this formal theory. A formal theory is interpreted as an axiom system in a domain. From this point of view, every first-order language is defined to describe some axiom system in a domain and the sets of its nonlogical symbols are all specially designed to express this system as a formal theory (see Section 4.1).

For example, the first-order language $\mathscr{A}$ is specially designed for the theory of elementary arithmetic Π, or in other words, it is defined for describing arithmetic operations on natural numbers.

A formal theory can be a system of calculus for constant symbols, function symbols, and predicate symbols occurring in it. For example, Π is a system of calculus for the function symbols S, $+$, and $\cdot$ and every sentence of Π is a rule of calculation for using these symbols.

(6) Consistency and completeness of formal theories. The consistency and completeness of formal theories can only be rigorously defined in first-order languages (see

Section 4.1). The most important results about formal theories in classical mathematical logic are the two theorems of Gödel. Namely, any finite formal theory of first-order languages containing the theory of elementary arithmetic Π is incomplete and its consistency cannot be proved by using any formal inference system (e.g., the **G** system) with the formal theory itself as premise (see Sections 5.4 and 5.5).

Since a formal theory of a first-order language is interpreted as an axiom system in the model and its consistency and completeness are interpreted as the consistency and completeness of the axiom system, Gödel's two theorems show the limitation of axiom systems that can be described by first-order languages. In other words, only with the aid of first-order languages can the consistency and completeness of axiom systems be rigorously defined but then, in this case, their consistency is unprovable and completeness is unattainable.

(7) New conjecture and refutation by facts. For any given model **M**, formal theory Γ, and formula A, if $\Gamma \vdash A$ is provable and $\mathbf{M} \models \neg A$ holds, then the model **M** with respect to $\neg A$ constitutes a refutation of Γ by facts. If $\mathbf{M} \models A$ holds and both $\Gamma \vdash A$ and $\Gamma \vdash \neg A$ are unprovable, then the formula A is a new axiom of Γ with respect to the model **M** (see Sections 7.2 and 7.3).

Here the model **M** is given. For Γ, neither its new conjectures nor its refutations by facts are logical consequences of Γ; they are sentences true in the model **M**. New conjectures and refutations by facts occur when discussing how to revise Γ with **M** as reference. In other words, refutations by facts and new conjectures are concepts occurring in the axiomatization process. Improvement or revision of a formal theory may generate new versions. We have to choose which model to use as reference for the revision of the formal theory, and this choice is guided by the data of experiments and observations.

(8) Revision calculus. Also called **R**-*calculus*, revision calculus is a system of calculus on **R**-configurations $\Delta \mid \Gamma$, in which Γ is a finite formula set and Δ is a formal theory composed of atomic sentences and negations of atomic sentences and is an **R**-refutation of Γ. **R**-calculus is composed of the **R**-axiom, **R**-logical connective symbol rules, **R**-quantifier symbol rules, and **R**-cut rules. The role of **R**-calculus is: with Δ as basis, use **R**-logical connective symbol and quantifier symbol rules as well as **R**-cut rules to delete those formulas that are inconsistent with Δ from Γ. Inconsistent **R**-configurations lead to scientific discovery, which can be automatically deduced by **R**-calculus. (see Section 7.4).

(9) Reachability, soundness, and completeness of R-calculus. The reachability of **R**-calculus means that for any given inconsistent **R**-configuration $\Delta \mid \Gamma$, any maximal subset of Γ that is consistent with Δ can be derived by using **R**-calculus. Namely, in applying **R**-calculus to delete formulas that are inconsistent with Δ from Γ, an **R**-termination will be reached after finitely many applications of **R**-calculus rules. If the **R**-termination is $\Delta \mid \Gamma'$, then Γ' is the maximal subset of Γ that is consistent with Δ.

The soundness of **R**-calculus means that for any given inconsistent **R**-configuration $\Delta \mid \Gamma$, when **R**-calculus is applied to delete formulas, that are inconsistent with Δ, from Γ. The **R**-termination that we obtain is $\Delta \mid \Gamma'$. If **M** is the model of Γ', then it is an ideal model of refutation by facts of Γ with respect to Δ.

The completeness of **R**-calculus means that for any model **M**, if Δ and Γ' are true in **M**, Γ is false, and Γ' is the maximal subset of Γ that is consistent with Δ, then Γ' can be formally deduced by using **R**-calculus on $\Delta \mid \Gamma$ (see Sections 7.6 and 7.7).

(10) Version sequences and their properties. In the process of axiomatization, every formal theory occurs in the form of a version Γ_n. When Γ_n is refuted by facts or encounters new conjectures, it will be revised and a new version Γ_{n+1} will be produced. The version sequence $\{\Gamma_n\}$ records the process of version evolution. The following three fundamental properties describe the properties of version evolution, which are also the characteristics of the evolution of the axiomatization process.

Convergence. The version sequence $\{\Gamma_n\}$ possesses convergence, i.e., not only the upper limit $\{\Gamma_n\}^*$ and lower limit $\{\Gamma_n\}_*$ of the version sequence are equal, but they are also equal to $Th(\mathbf{M})$. The convergence of the version sequence may be interpreted as: the limit of this sequence contains all true propositions about the domain under investigation (see Sections 6.2, 8.1, and 8.3).

Commutativity. The commutativity between the limit operation and the formal inference, i.e.,

$$\lim_{n\to\infty} Th(\Gamma_n) = Th(\lim_{n\to\infty} \Gamma_n).$$

In other words, the limit of formal theory closures is equal to the closure of the limit of formal theories. The commutativity indicates that to find the limit of formal theory closures, it suffices to find the limit of the formal theory sequence, and vice versa. If the initial formal theory in the version sequence is finite, then the infinite $\lim_{n\to\infty} Th(\Gamma_n)$ can be found by means of revising and expanding each finite Γ_n (see Section 8.4).[2]

Independence. We have shown that, if each version Γ_n is an independent formal theory and $\{\Gamma_n\}$ converges, then $\lim_{n\to\infty} \Gamma_n$ is also an independent formal theory (see Section 8.5).

The interpretation of the independence of a formal theory in the model is the independence of the axioms contained in the axiom system. The independence of axioms occupies an important position in the axiomatization process of the knowledge of mathematics and natural science. It is generally agreed that an axiom system is an ideal system only if it possesses independence.

Domains such as groups, rings, fields, classical mechanics, electromagnetics and quantum mechanics are all theories constructed from independent axioms and principles.

In practice, for ease of use and understanding, many axiom systems abandon the requirement of independence. For example, the **G** system with the cut rule added is not

[2]The number of statements contained in software systems is always finite, whereas the application domain of each software system is equivalent to a theory closure, which can be infinite. In updating each version of a software system, if the version sequence possesses commutativity between the limit operation and the formal calculus, then one only needs to update each software version and the validity of all applications can be guaranteed. In the software engineering community, the functions provided by a software system are called a *business logic*, while the applications of the software are called the *services* of a business logic. Within the context of first-order languages, the business logic can be implemented using systems of calculus on formal theories. The services of a business logic can be described by theory closures. Therefore, the commutativity of software version sequences is also called the *scalability* of software functions.

independent. In computer design and software development, most of the instruction sets and software systems are not made independent because this would entail longer calculation, whereas ease of use and efficiency are a priority for such systems.

(11) Proscheme. A proscheme is a generalization of the concept of *P*-procedure presented in Chapter 4. Firstly, it expands the conditions in the **if** statement and **while** statement of the *P*-procedure as follows: the conditions are both allowed to be boolean expressions and such undecidable relations as "$\Gamma \vdash A$ is provable" or "**consistent**(Γ, A) holds" (i.e., Γ and A are consistent). Secondly, the input to a proscheme is a sequence of sentences and the output of the proscheme is a sequence of formal theories. We can call the mathematical proof given by a carefully designed proscheme a kind of *constructive proof*. For example, the proofs of the Lindenbaum lemma and other related theorems in Chapters 8 and 9 can all be considered as constructive proofs (see Sections 6.3, 8.3–8.5, and 9.4–9.7).

Let P be a proscheme that takes Γ as initial formal theory, $\{A_n\}$ as input sequence of sentences, and $\{\Gamma_n\}$ as output sequence of versions. A proscheme P is said to be *reliable* if P is convergent and commutative. The proscheme P is said to be *ideal* if P is reliable and is independent. The proschemes `OPEN` and `GUINA` in Chapters 8 and 9 are both reliable. They can be modified to be ideal proschemes.

Consciously or unconsciously, we follow a certain method of research or use a certain strategy of development in proposing an axiom system for some domain. The methodology determines the quality of research and development. If this methodology can be described by using reliable proschemes, then the convergence ensures that it will eventually find all the true propositions of the domain. The commutativity guarantees that we can use a finite axiom system in each step of development. Lastly if the proscheme is ideal, the axioms in each version are mutually independent. This shows the significance of reliable and ideal proschemes.

(12) Meta-language environment. Every first-order language and its model are defined and explained in a meta-language environment. Theorems related to this first-order language and its model are proved in the meta-language environment. Every meta-language environment must obey the six basic principles given in Section 10.2 (see the introduction in Chapter 2).

The above 12 basic points form the *theoretical framework of first-order languages*. This framework is another important advance in the theory of knowledge. This is justified in the following points.

1. Descriptions of the concepts and problems studied in axiomatization are made rigorous. It is only after a first-order language, its model, and their meta-language environment are defined that the soundness and completeness of the inference rules for logical connectives and quantifiers can be strictly defined and proved. Moreover, only in the three language environments can properties such as consistency, completeness, decidability and independence be rigorously described, and so can Gödel's theorems about the incompleteness and consistency of formal theories be strictly proved.

2. Mathematical proofs of theorems in axiomatization are converted to symbolic calculus. As every formal theory is a symbol string satisfying the syntactic rules of first-order languages and the inference rules used in formal proofs are rules of calculus for logical connective symbols and quantifier symbols, one can design interactive computable procedures to prove formal consequences.

   Moreover, the soundness and completeness of formal inference systems ensure that, in performing symbolic calculus for formal proofs, there is no need to consider the interpretation of the formulas. Mathematical proofs are converted to routines in symbolic calculus.

   We have pointed out in the previous section that the advantage of axiomatization is to convert logical analysis on propositions to mathematical proofs. But, in general, finding a mathematical proof is a difficult art. Professional training and special insights are essential.

   On the other hand, for knowledge that can be described using first-order languages, mathematical proofs of theorems can be converted to routines in symbolic calculus. Thus, within the theoretical framework of first-order languages, finding a proof becomes a routine that can be assisted by interactive software tools. This frees human intelligence to focus on establishing axiom systems, proving their consistency and implementing efficient software tools.

3. The introduction of refutation by facts, revision calculus, version sequence and its limits, and proscheme makes it possible to strictly describe problems and prove theorems about axiomatization. The proposed **R**-calculus allows us, on the one hand, to formally describe the revision of axiom systems. It also allows us to convert the process of revision of axiom systems to a symbolic calculus, which can be assisted by interactive software systems.

   In the development of software systems, if a first-order language is used to describe their specifications and atomic sentences or negations of atomic sentences are used to describe testing samples, then the **R**-calculus system, together with the basic theorem of testing, form a theoretical framework for software testing.

4. The idea of proscheme makes it feasible to formally describe the methodologies of axiomatization. For example, the limits of version sequences can be used to analyze the eventual results of an evolutionary process. Formal descriptions of the reliability of methodologies, as well as methods for proving the reliability of proschemes, have been given by introducing the concepts of convergence, commutativity, and independence.

It should be pointed out that the theoretical framework of first-order languages is by no means a panacea for solving all problems in mathematics and natural science and it applies only to countably infinite objects. Moreover, the method of logical analysis based on this framework has the same limitations as Gödel showed for first-order languages.

The theoretical framework of first-order languages is only an advantage when axiomatization of a domain has reached a stage of maturity, i.e., the basic knowledge struc-

ture with the axiom system as its core has already been formed and extensive empirical models and observated data have been accumulated.

The concepts, methods, and models used in computer science, programming languages, software engineering, digitalized design and artificial intelligence are very similar to the theoretical framework of first-order languages. In fact, if the knowledge of a domain can be described by a formal language, then the theoretical framework of first-order languages can be generalized to that domain. In other words, the framework can be generalized to include formal languages and, when we do this, we can call this framework *formal methods*. In this case, we can view the theoretical framework of first-order languages as a typical example of formal methods.

If we impose restrictions on formal methods, prescribing that formal languages can only be programming languages, then formal methods are called *digital methods*. Digital methods have the advantage that the concepts and methods can all be implemented in a computer.

Owing to the rapid development of large-scale integrated circuits and the Internet, we now have access to extremely powerful computing resources. These resources are leading to a change in research methods in many branches of natural science, from axiomatization to digitalization. The resources and the applications developed on them are becoming the infrastructure indispensable for modern society. The theoretical framework of first-order languages can be viewed as the foundation for the construction of this information infrastructure.

10.5 Workflow of scientific research

In the previous section we completed our review of this book and showed how the concepts presented in it form a complete theoretical framework for first-order languages, or, more generally, of formal methods. In this section, we discuss how to use this knowledge to define a reliable workflow for scientific research.

The purpose of scientific discovery is to explain natural phenomena that have taken place and predict phenomena that have not been observed. Based on this dependable knowledge, engineering innovation creates new materials, manufactures machines and designs software to improve our quality of life. So, confirming that the process of scientific research is reliable will reinforce the foundation of our information society.

We have established in this book a paradigm for research that takes the following steps. Data is acquired from experiments and measurement. Patterns in this data can be formulated into propositions, which then can be generalized by induction to describe universal laws. A selection of these laws is chosen to make an axiom system for a domain, which is a mathematical system called a scientific theory, with which we model the real world.

For this model to be accepted, the logical consequences of the axioms should explain what we know to be true, but also they will inevitably predict phenomena we have not yet observed. If these predictions are confirmed by experiment, then this supports the theory but if they contradict the predictions, then the theory needs to be revised and we

need to make new conjectures.

This process is reliable if, as we repeat the above steps, the versions of the theory approach gradually to the full of truth of the domain.

Applying formal methods, one can clearly define activities such as induction, proof, interpretation, prediction, refutation and revision. We can then design interactive software systems to assist these hitherto manual processes and to give a workflow of scientific research for each specific domain. The following is the core content of this workflow.

1. **The Meta-language Environment $\mathfrak{L}$**

   (a) Choose a natural language as the meta-language environment $\mathfrak{L}$. This environment contains the domain of knowledge under investigation and the knowledge and theories that are already widely accepted. $\mathfrak{L}$ satisfies the six basic principles about meta-language environments given in Section 10.2.

   (b) Describe the experiments and observations and acquire data from their results.

   (c) Express the relations between the data by using propositions of $\mathfrak{L}$. Denote these propositions by
   $$\mathrm{A}_n = \{a_1, \ldots, a_k\}.$$

   (d) Formulate conjectures about universal laws on the basis of the data and phenomena. The set of conjectures is denoted by
   $$\mathrm{B}_n = \{b_1, \ldots, b_l\}.$$

   (e) In addition, we use the meta-language environment to: construct models, define the corresponding first-order languages, design proschemes, and prove properties of the first-order languages and their models.

2. **The Domains**

   (a) Introduce constants, functions and sets to describe persistent features of data, the relations between data, and classifications of data.

   (b) Formulate mathematical equations that connect the above features. The equations should be supported by the experiments. These equations and basic concepts constitute atomic propositions or negations of atomic propositions. They are denoted by
   $$\mathcal{A}_n = \{\alpha_1, \ldots, \alpha_s\}.$$

   (c) Use logical connectives and quantifiers to connect atomic propositions to form propositions. Those propositions that are in accordance with some observed phenomena are called *true propositions* denoted by β_j, and the propositions that are in accordance with all observed phenomena are called *universal principles*. The true propositions form a set
   $$\mathcal{B}_n = \{\beta_1, \ldots, \beta_t\}.$$

(d) Select from $\mathcal{B}_n$ those propositions that are most fundamental to form an axiom system denoted as $\mathcal{T}_n$. It should be noted that $\mathcal{T}_n$ contains no purely logical axioms. $\mathcal{T}_n$ can only contain propositions about the domain such as arithmetic, the theory of relativity and the theory of evolution.

(e) These constants, functions and propositions, $\mathcal{A}_n$, $\mathcal{B}_n$, and $\mathcal{T}_n$, form the basis of the domain which is denoted as $\mathbb{M}$. The domain $\mathbb{M}$ is a mathematical system, which, generally speaking, is not unique.

3. **The Object Language**

(a) On the basis of the constants, variables, functions, and atomic propositions of the domain, define a corresponding set of constant symbols, set of variable symbols, set of function symbols, and set of predicate symbols and thus define a corresponding first-order language $\mathscr{L}$.

(b) It should be noted that, for some domains, it may be more appropriate to define a formal language using a strict grammar. The workflow still applies.

(c) Define an interpretation map I, ensuring that every domain $\mathbb{M}$ is a model of $\mathscr{L}$, such that in the model the atomic sentences or negations of atomic sentences A_i of $\mathscr{L}$ are interpreted as the propositions α_i, the composite sentences B_j are interpreted as the propositions β_j, and the nth versions of formal theory Γ_n is interpreted as the axiom system $\mathcal{T}_n$.

4. **Formal Axiomatization** Note that it is this stage in the workflow that specifies what activities can be delegated to information technology.

(a) With $\Gamma = \Gamma_n$ and $\{A_1, \ldots, A_s\}$ as input, call a `GUINA`-like proscheme to generate the formal theory Γ_s. In executing the proscheme, there are two cases as follows.

(i) When it is required to prove that $\Gamma_i \vdash A_i$ holds, call the proof procedure *CP* in Section 3.2 and perform the proof with the aid of interactive software.

(ii) When we find a refutation by showing that $\Gamma_i \vdash \neg A_i$ holds, apply software tools based on the **R**-calculus to find maximal contractions of Γ_i with respect to A_i.

(b) With $\Gamma = \Gamma_s$ and $\{B_1, \ldots, B_t\}$ as input, call an `OPEN`-like proscheme to generate the formal theory Γ_t. There are three situations to distinguish:

(i) When we need to prove $\Gamma_j \vdash B_j$, we call the proof procedure *CP* in Section 3.2 and perform the proof with the aid of interactive software.

(ii) When we find a formal refutation by showing that $\Gamma_j \vdash \neg B_j$ for some j, then decompose $\neg B_j$ into atomic sentences and negations of atomic sentences $A_{j_1}, \ldots, A_{j_k}$, interpreted as the propositions $\alpha_{j_1}, \ldots, \alpha_{j_k}$ in the domain, and verify the truth of these atomic sentences in the meta-language environment $\mathfrak{N}$.

(iii) With Γ_j and $\{A_{j_1},\ldots,A_{j_k}\}$ as input, call a GUINA-like proscheme to generate the new version Γ_{j+1}.

This workflow shows how research can be done using the resources of the modern information society. It is reliable for the following reasons.

(1) Chapters 6 to 9 show that this workflow is convergent and commutative.

(2) For each version of the theory, the analysis of logical relations between propositions and proofs in the meta-language environment can be accomplished in first-order languages using interactive software based on the **G** system.

(3) When we find a refutation by facts, maximal contractions can be created and selected in first-order languages using interactive software based on the **R**-calculus.

(4) Mathematical calculations can also be assisted by efficient computer software designed specifically for this domain.

It should be pointed out that the workflow applies to all domain knowledge that can be described using formal languages and, for each specific problem, we can design proschemes, which have the same reliability as OPEN and GUINA but are far more efficient.

In summary, the framework of three language environments provides a clear way of determining those aspects of the research workflow can be partially automated. In particular, the design of experiments, the selection of axioms and the proof of significant consequences still need human intelligence. This aspect takes place in the meta-language environment and its purpose is to describe natural phenomena and incorporate all the widely accepted knowledge.

The result of this is a mathematical model of the domain, in which we can make definitions, do calculations and develop mathematical proofs. This is an environment where we can reason about knowledge using mathematics and has been, until now, the context for science.

However, this book has shown that a large part of the process of research can now be defined in the context of a formal language, which is an environment in which human and computer can interact. It is a digitalized virtual environment, in which reasoning can be assisted by software in a reliable way. The process of scientific discovery can be enhanced and implemented by proschemes, which are convergent, commutative and efficient. This ensures that scientific research will continually improve our knowledge and is a process that eventually approaches the truth.

Appendix 1

Sets and Maps

A collection of distinct objects is called a *set*. A set is usually denoted in boldface **A**, **B**, **M**, **N**, ... An individual in a set is called an *element* which is usually denoted as a, b, ... When a is an element of the set **A**, it is denoted as $a \in \mathbf{A}$ and reads as a belongs to **A**; when a is not an element of **A**, it is denoted as $a \notin \mathbf{A}$. A set containing no element is called the *empty set* and denoted as $\varnothing$. A set consisting of finitely many elements $a_1, a_2, \ldots, a_n$ is denoted as $\{a_1, a_2, \ldots, a_n\}$.

Definition A1.1 (Subset). If both **A** and **B** are sets and $a \in \mathbf{B}$ for all $a \in \mathbf{A}$, then **A** is called a *subset* of **B** and denoted as $\mathbf{A} \subseteq \mathbf{B}$. If there exists a $b \in \mathbf{B}$ such that $b \notin \mathbf{A}$, then **A** is called a *proper subset* of **B** and denoted as $\mathbf{A} \subset \mathbf{B}$.

Definition A1.2 (Equality). If both $\mathbf{A} \subseteq \mathbf{B}$ and $\mathbf{B} \subseteq \mathbf{A}$ hold for sets **A** and **B**, then we say that **A** and **B** are *equal* and denote it as $\mathbf{A} = \mathbf{B}$.

Definition A1.3 (Union). $\mathbf{A} \cup \mathbf{B}$ is called the *union* of the sets **A** and **B** if $x \in \mathbf{A} \cup \mathbf{B}$ holds if and only if $x \in \mathbf{A}$ or $x \in \mathbf{B}$ holds.

Definition A1.4 (Intersection). $\mathbf{A} \cap \mathbf{B}$ is called the *intersection* of the sets **A** and **B** if $x \in \mathbf{A} \cap \mathbf{B}$ holds if and only if both $x \in \mathbf{A}$ and $x \in \mathbf{B}$ hold. If $\mathbf{A} \cap \mathbf{B}$ is an empty set, then we say that the sets **A** and **B** *do not intersect*.

Definition A1.5 (Complement). $\mathbf{A} - \mathbf{B}$ is called the *complement* of the set **B** with respect to the set **A** if $a \in \mathbf{A} - \mathbf{B}$ holds if and only if $a \in \mathbf{A}$ but $a \notin \mathbf{B}$.

Definition A1.6 (Map). Let **A** and **B** be two sets. If there exists a correspondence φ such that for all $a \in \mathbf{A}$, there exists a unique $b \in \mathbf{B}$ corresponding to it, then φ is called a *map* from **A** to **B** and denoted as

$$\varphi : \mathbf{A} \to \mathbf{B}.$$

A is called the *domain* of φ and $\varphi(\mathbf{A})$ is called the *image* of **A** with respect to φ. The element a is called a *preimage* of b and b is called the *image* of a with respect to φ and denoted as $\varphi(a)$.

A map φ is called an *injection* if for two different elements $a \neq b$ of **A**, their images are also different, i.e., $a \neq b$ implies $\varphi(a) \neq \varphi(b)$.

A map φ is called a *surjection* or *onto* if every $b \in \mathbf{B}$ is an image of an element a in **A**, i.e., $b = \varphi(a)$.

If φ is both injective and surjective, then φ is called a *bijection*.

Definition A1.7 (Set $\mathbb{N}$ of natural numbers). The set consisting of all the natural numbers is called the *set of natural number* and denoted as $\mathbb{N}$, i.e.,

$$\mathbb{N}: \{0, 1, 2, \ldots, n, \ldots\}.$$

Definition A1.8 (Countable set). A set $\mathbf{A}$ is called a *countable set* if there exists a bijection $\varphi : \mathbb{N} \to \mathbf{A}$.

Example A1.1 (Set of even numbers). The set $\mathbf{E}$ of even numbers is a countable set since the following bijection can be constructed:

$$\begin{array}{ccccccc} 0, & 1, & 2, & 3, & \ldots & n, & \ldots \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ 0, & 2, & 4, & 6, & \ldots, & 2n, & \ldots \end{array}$$

In fact, $\varphi : \mathbb{N} \to \mathbf{E}$, $\varphi(n) = 2n$ suffices for the conclusion.

Example A1.2 (Set of proper fractions). The set consisting of all the rational numbers between 0 and 1 is countable. Since every rational number can be represented by a fraction $\frac{p}{q}$, we can enumerate all the rational numbers between 0 and 1 as follows. First, there is only one rational number with denominator 2, i.e., $\frac{1}{2}$. Then there are two rational numbers with denominator 3 and they are $\frac{1}{3}$ and $\frac{2}{3}$. There are also two rational numbers with denominator 4 and they are $\frac{1}{4}$ and $\frac{3}{4}$. The fraction $\frac{2}{4}$ is the same as $\frac{1}{2}$ and is skipped since it has been listed. In this way every $\frac{p}{q}$ is listed and hence the map is onto. Such kind of enumeration also ensures that different rational numbers have different preimages. Thus the map is injective. In conclusion, all the rational numbers can be enumerated as a countable sequence:

$$\frac{1}{2}, \frac{1}{3}, \frac{2}{3}, \frac{1}{4}, \frac{3}{4}, \ldots.$$

The fractions in this sequence are in one-to-one correspondence with the natural numbers.

Definition A1.9 (Characteristic function). For every set $\mathbf{A}$, there exists a function $\mathcal{X}_{\mathbf{A}} : \mathbf{A} \to \{0, 1\}$ satisfying

$$\mathcal{X}_{\mathbf{A}}(x) = \begin{cases} 1, & \text{if } x \in \mathbf{A}, \\ 0, & \text{if } x \notin \mathbf{A}. \end{cases}$$

$\mathcal{X}_{\mathbf{A}}$ is called the *characteristic function* of the set $\mathbf{A}$.

There is a one-to-one correspondence between sets and their characteristic functions. All set operations can be represented in terms of their characteristic functions.

Definition A1.10 (Union and intersection of a set sequence). Suppose that $\mathbf{A}_1, \ldots, \mathbf{A}_n, \ldots$ is a sequence of sets. Then

$$\bigcup_{i=1}^{\infty} \mathbf{A}_i$$

is a set, called the *union of the set sequence*, if $a \in \bigcup_{i=1}^{\infty} \mathbf{A}_i$ if and only if there exists some i such that $a \in \mathbf{A}_i$. The set

$$\bigcap_{i=1}^{\infty} \mathbf{A}_i$$

is also a set, called the *intersection of the set sequence*, if $a \in \bigcap_{i=1}^{\infty} \mathbf{A}_i$ if and only if $a \in \mathbf{A}_i$ holds for every $\mathbf{A}_i$.

The union and intersection operations of sets satisfy not only the commutative law and associative law but also the distributive law:

$$\begin{aligned}
\mathbf{A} \cup \mathbf{B} &= \mathbf{B} \cup \mathbf{A}, \\
\mathbf{A} \cup (\mathbf{B} \cup \mathbf{C}) &= (\mathbf{A} \cup \mathbf{B}) \cup \mathbf{C}, \\
\mathbf{A} \cap \mathbf{B} &= \mathbf{B} \cap \mathbf{A}, \\
\mathbf{A} \cap (\mathbf{B} \cap \mathbf{C}) &= (\mathbf{A} \cap \mathbf{B}) \cap \mathbf{C}, \\
\mathbf{A} \cap (\mathbf{B} \cup \mathbf{C}) &= (\mathbf{A} \cap \mathbf{B}) \cup (\mathbf{A} \cap \mathbf{C}), \\
\mathbf{A} \cup (\mathbf{B} \cap \mathbf{C}) &= (\mathbf{A} \cup \mathbf{B}) \cap (\mathbf{A} \cup \mathbf{C}).
\end{aligned}$$

The complement of a set satisfies the following three properties:

$$\begin{aligned}
(\mathbf{A} - \mathbf{B}) \cap \mathbf{B} &= \varnothing, \\
\mathbf{A} - (\mathbf{B} \cap \mathbf{C}) &= (\mathbf{A} - \mathbf{B}) \cup (\mathbf{A} - \mathbf{C}), \\
\mathbf{A} - (\mathbf{B} \cup \mathbf{C}) &= (\mathbf{A} - \mathbf{B}) \cap (\mathbf{A} - \mathbf{C}).
\end{aligned}$$

All the above equalities on sets can be directly verified by the definitions of the equality, union, intersection, and complement of sets.

Appendix 2

Substitution Lemma and Its Proof

Lemma A2.1 (Substitution lemma). *Let $\mathscr{L}$ be a first-order language with* $\mathbf{M}$ *and* σ *being its structure and assignment respectively. Suppose that t, t' and A are terms and formula of $\mathscr{L}$ respectively. Then the following two equalities hold:*

$$\begin{aligned}(t[t'/x])_{\mathbf{M}[\sigma]} &= t_{\mathbf{M}[\sigma[x:=t'_{\mathbf{M}[\sigma]}]]},\\ (A[t/x])_{\mathbf{M}[\sigma]} &= A_{\mathbf{M}[\sigma[x:=t_{\mathbf{M}[\sigma]}]]}.\end{aligned}$$

Proof. In what follows we give a structural inductive proof for the lemma. Let us begin with proving the first substitution equality through an induction on the structure of terms.

(1) t is a variable y. According to Definition 1.6 on formal substitutions in Chapter 1, we prove the equality in the following two cases: if $x \neq y$, then $y[t'/x] = y$ and thus

$$\begin{aligned}(y[t'/x])_{\mathbf{M}[\sigma]} &= y_{\mathbf{M}[\sigma]}\\ &= \sigma(y)\\ &= y_{\mathbf{M}[\sigma[x:=t'_{\mathbf{M}[\sigma]}]]}.\end{aligned}$$

The last step in the above holds since, according to the definition of $\sigma[x := t'_{\mathbf{M}[\sigma]}]$, the case $x \neq y$ indicates that it shares the same value as σ on the variable y.

If $x = y$, then $y[t'/x] = x[t'/x] = t'$ and thus

$$\begin{aligned}(y[t'/x])_{\mathbf{M}[\sigma]} &= (x[t'/x])_{\mathbf{M}[\sigma]} && \text{(because } x = y\text{)}\\ &= t'_{\mathbf{M}[\sigma]} && \text{(according to the formal substitution rule)}\\ &= x_{\mathbf{M}[\sigma[x:=t'_{\mathbf{M}[\sigma]}]]} && \text{(definition of the meaning of terms)}\\ &= y_{\mathbf{M}[\sigma[x:=t'_{\mathbf{M}[\sigma]}]]} && \text{(because } x = y\text{).}\end{aligned}$$

(2) If t is c, the proof is similar to (1).

(3) If t is $ft_1 \cdots t_n$, then

$$\begin{aligned}
(ft_1 \cdots t_n)[t'/x]_{\mathbf{M}[\sigma]} &= (ft_1[t'/x] \cdots t_n[t'/x])_{\mathbf{M}[\sigma]} \\
&\quad \text{(Definition 1.6 on substitutions)} \\
&= f_{\mathbf{M}}((t_1[t'/x])_{\mathbf{M}[\sigma]} \cdots (t'_n[t/x])_{\mathbf{M}[\sigma]}) \\
&\quad \text{(Definition 2.5 on the meaning of terms)} \\
&= f_{\mathbf{M}}((t_1)_{\mathbf{M}[\sigma[x:=t'_{\mathbf{M}[\sigma]}]]} \cdots (t_n)_{\mathbf{M}[\sigma[x:=t'_{\mathbf{M}[\sigma]}]]}) \\
&\quad \text{(structural induction hypothesis)} \\
&= (ft_1 \cdots t_n)_{\mathbf{M}[\sigma[x:=t_{\mathbf{M}[\sigma]}]]} \\
&\quad \text{(Definition 2.5 on the meaning of terms).}
\end{aligned}$$

Thus the first equality in the lemma, i.e., the substitution equality on terms, are proved.

In what follows we prove that the second equality in the lemma, i.e., the substitution equality on formulas, also holds. Let us make an inductive proof on the structure of the formula A. The proof examines five cases: $Pt_1 \cdots t_n$, $t_1 \doteq t_2$, $\neg B$, $B \vee C$, and $\exists xB$.

(1) If A is $Pt_1 \cdots t_n$, then

$$\begin{aligned}
(Pt_1 \cdots t_n)[t/x]_{\mathbf{M}[\sigma]} &= (Pt_1[t/x] \cdots t_n[t/x])_{\mathbf{M}[\sigma]} \\
&= P_{\mathbf{M}}((t_1[t/x])_{\mathbf{M}[\sigma]} \cdots (t_n[t/x])_{\mathbf{M}[\sigma]}) \\
&= P_{\mathbf{M}}((t_1)_{\mathbf{M}[\sigma[x:=t_{\mathbf{M}[\sigma]}]]} \cdots (t_n)_{\mathbf{M}[\sigma[x:=t_{\mathbf{M}[\sigma]}]]}) \\
&= (Pt_1 \cdots t_n)_{\mathbf{M}[\sigma[x:=t_{\mathbf{M}[\sigma]}]]}.
\end{aligned}$$

In this set of equalities, according to the formal substitution rule on predicates in Definition 1.7, the first equality holds. According to Definition 2.8 on the meaning of predicates, the second equality holds. According to the substitution equality on terms that has just been proved in this lemma, the third equality holds. According to the definition of the meaning of predicates, the last equality holds.

(2) If A is $t_1 \doteq t_2$, then

$$\begin{aligned}
((t_1 \doteq t_2)[t/x])_{\mathbf{M}[\sigma]} &= ((t_1[t/x]) \doteq (t_2[t/x]))_{\mathbf{M}[\sigma]} \\
&= \begin{cases} T & \text{if } (t_1[t/x])_{\mathbf{M}[\sigma]} = (t_2[t/x])_{\mathbf{M}[\sigma]}, \\ F & \text{otherwise} \end{cases} \\
&= \begin{cases} T & \text{if } (t_1)_{\mathbf{M}[\sigma[x:=t_{\mathbf{M}[\sigma]}]]} = (t_2)_{\mathbf{M}[\sigma[x:=t_{\mathbf{M}[\sigma]}]]}, \\ F & \text{otherwise} \end{cases} \\
&= (t_1 \doteq t_2)_{\mathbf{M}[\sigma[x:=t_{\mathbf{M}[\sigma]}]]}.
\end{aligned}$$

In this set of equalities, the first holds according to the formal substitution rule on $\doteq$ in Definition 1.7. According to the meaning of $\doteq$ in Definition 2.8, the second equality holds.

An application of the induction hypothesis to t_1 and t_2 indicates that the third equality holds. The validity of the last equality is proved by the meaning of $\doteq$ in Definition 2.8 and the assignment becomes $\sigma[x := t_{\mathbf{M}[\sigma]}]$.

(3) If A is $\neg B$, then

$$\begin{aligned}((\neg B)[t/x])_{\mathbf{M}[\sigma]} &= (\neg(B[t/x]))_{\mathbf{M}[\sigma]} && \text{(according to the substitution rule of } \neg)\\ &= H_{\neg}((B[t/x])_{\mathbf{M}[\sigma]}) && \text{(according to the meaning of } \neg)\\ &= H_{\neg}(B_{\mathbf{M}[\sigma[x:=t_{\mathbf{M}[\sigma]}]]}) && \text{(according to the induction hypothesis)}\\ &= (\neg B)_{\mathbf{M}[\sigma[x:=t_{\mathbf{M}[\sigma]}]]} && \text{(according to the meaning of } \neg).\end{aligned}$$

(4) If A is $B \vee C$, then

$$\begin{aligned}((B \vee C)[t/x])_{\mathbf{M}[\sigma]} &= ((B[t/x]) \vee (C[t/x]))_{\mathbf{M}[\sigma]}\\ &= H_{\vee}((B[t/x])_{\mathbf{M}[\sigma]}, (C[t/x])_{\mathbf{M}[\sigma]})\\ &= H_{\vee}(B_{\mathbf{M}[\sigma[x:=t_{\mathbf{M}[\sigma]}]]}, B_{\mathbf{M}[\sigma[x:=t_{\mathbf{M}[\sigma]}]]})\\ &= (B \vee C)_{\mathbf{M}[\sigma[x:=t_{\mathbf{M}[\sigma]}]]}.\end{aligned}$$

Similar to the last set of equalities, this set of equalities are according to, in sequence, the formal substitution rule of $\vee$, the meaning of $\vee$, the structural induction hypothesis and the meaning of $\vee$ on the formulas B and C respectively with the assignment being $\sigma[x := t_{\mathbf{M}[\sigma]}]$.

(5) A is $\exists yB$. According to the formal substitution rule, the proof is given in two cases.

(a) t is free in $\exists yB$ with respect to x, i.e., $y \neq x$, $y \notin FV(t)$ or $x \notin FV(B)$. Then

$$\begin{aligned}((\exists yB)[t/x])_{\mathbf{M}[\sigma]} &= (\exists y(B[t/x]))_{\mathbf{M}[\sigma]}\\ &\Leftrightarrow \text{there exists an } a \in M \text{ such that } (B[t/x])_{\mathbf{M}[\sigma[y:=a]]} = T \text{ holds}\\ &\Leftrightarrow \text{there exists an } a \in M \text{ such that } B_{\mathbf{M}[(\sigma[y:=a])[x:=t_{\mathbf{M}[\sigma[y:=a]]}]]} = T\\ &\Leftrightarrow \text{there exists an } a \in M \text{ such that } B_{\mathbf{M}[(\sigma[x:=t_{\mathbf{M}[\sigma]}])[y:=a]]} = T\\ &\Leftrightarrow (\exists yB)_{\mathbf{M}[\sigma[x:=t_{\mathbf{M}[\sigma]}]]}.\end{aligned}$$

Hereafter we shall use the equivalence symbol $\Leftrightarrow$, which is a symbol used in the meta-language, to denote "if and only if".

In the above set of equalities and equivalent relations, the first equality holds according to the formal substitution rule of $\exists$ in Definition 1.7. According to the meaning of $\exists$ in Definition 2.8, the first equivalent relation holds. Invoking the induction hypothesis on B leads to the second equivalent relation. Since $x \neq y$ and $y \notin FV(t)$, according to the definition on the assignment in Definition 2.4,

$$(\sigma[y := a])[x := t_{\mathbf{M}[\sigma[y:=a]]}] \;=\; (\sigma[x := t_{\mathbf{M}[\sigma]}])[y := a]$$

holds. Thus the third equivalent relation holds. The last equivalent relation is obtained from the meaning of $\exists$ with the assignment being $\sigma[x := t_{\mathbf{M}[\sigma]}]$.

(b) t is not free in $\exists yB$ with respect to x, i.e., $y \neq x$. But $y \in FV(t)$ and $x \in FV(B)$. In this case, according to (10) in Definition 1.7 on the substitution rule of $\exists$, in order to substitute $(\exists yB)[t/x]$ by $\exists zB[z/y][t/x]$ with z being an eigen-variable, we have

$$\begin{aligned}
((\exists yB)[t/x])_{\mathbf{M}[\sigma]} &= (\exists z(B[z/y][t/x]))_{\mathbf{M}[\sigma]} \\
&\Leftrightarrow \text{there exists an } a \in M \text{ such that } (B[z/y][t/x])_{\mathbf{M}[\sigma[z:=a]]} = T \\
&\Leftrightarrow \text{there exists an } a \in M \text{ such that } (B[z/y])_{\mathbf{M}[(\sigma[z:=a])[x:=t_{\mathbf{M}[\sigma[z:=a]]}]]} = T \\
&\Leftrightarrow \text{there exists an } a \in M \text{ such that } (B[z/y])_{\mathbf{M}[(\sigma[z:=a])[x:=t_{\mathbf{M}[\sigma]}]]} = T \\
&\Leftrightarrow \text{there exists an } a \in M \text{ such that } (B[z/y])_{\mathbf{M}[(\sigma[x:=t_{\mathbf{M}[\sigma]}])[z:=a]]} = T \\
&\Leftrightarrow (\exists zB[z/y])_{\mathbf{M}[\sigma[x:=t_{\mathbf{M}[\sigma]}]]} \\
&= (\exists yB)_{\mathbf{M}[\sigma[x:=t_{\mathbf{M}[\sigma]}]]}.
\end{aligned}$$

In the above equalities and equivalent relations, according to the formal substitution rule of $\exists$, the first equality holds. According to the definition of the meaning of $\exists$, the first equivalent relation holds. Invoking the structural induction hypothesis on $(B[z/y])$ leads to the second equivalent relation. Since z is an eigen-variable that does not appear in t, it is not a free variable of t. Thus

$$t_{\mathbf{M}[\sigma]} = t_{\mathbf{M}[\sigma[z:=a]]}$$

holds and the third equivalent relation holds. In the forth equivalent relation, z being an eigen-variable implies that $z \neq x$. Thus

$$(\sigma[z := a])[x := t_{\mathbf{M}[\sigma]}] = (\sigma[x := t_{\mathbf{M}[\sigma]}])[z := a]$$

holds and the forth equivalent relation holds. According to the definition of the meaning of $\exists$, the fifth equivalent relation holds. The last equality is obtained from the formal substitution rule on $\exists$. □

Appendix 3

Proof of the Representability Theorem

This appendix proves the representability theorem given in Chapter 4. We have pointed out before that the key for the proof is to find an $\mathscr{A}$ formula $A(x_1,x_2,x_3)$ to represent the corresponding P-procedure $F(x_1,x_2,x_3)$. In this way the proof of the representability theorem is naturally divided into two parts, i.e., constructing a formula and proving that it is the representation of the P-procedure $F(x_1,x_2,x_3)$ in Π. We have explained in Chapter 4 the idea for constructing the formula $A(x_1,x_2,x_3)$ using the method of structural inductions. More specifically, the respective representations of the P-procedure statements in Π are first described. Then the representation of the P-procedure itself in Π is defined via structural inductions. As in the notation in Chapter 4, we also use $\tau := \{x_1,x_2,x_3\}$ and $\tau' := \{y_1,y_2,y_3\}$ to denote the set of initial state variables and the set of terminating state variables respectively. In this way the statements of the P-procedure are represented by the $\mathscr{A}$ formulas whose free variables are the state variables $\{x_1,x_2,x_3\}$ and $\{y_1,y_2,y_3\}$. It is relatively easy to represent the assignment statement, **if** statement, sequential statement, and **call** statement as specified in Chapter 4. Nonetheless, it is not easy to represent the **while** statement as such a kind of formula. There we introduced a lemma proved by Gödel based on which the idea of constructing the representation of the **while** statement in Π is given. In this appendix we shall describe in detail the construction process of the representation of the **while** statement in Π, and prove the representability of the P-procedure body.

A3.1 Representation of the while statement in Π

According to the structural operational semantics of the **while** statement, it is easy to see that $\sigma_0 = \sigma$, $\sigma_l = \sigma'$ with l being the number of executions of the procedure body. The $(i+1)$th loop of the **while** statement is the execution of the procedure body α' in the state σ_i. The condition $0 < [x_1]_{\sigma_i}$ of the **while** statement holds and the state after the execution is σ_{i+1} with $0 \leqslant i < l$. Thus follows the conclusion of Lemma 4.6.

According to the discussions in Chapter 4, the meaning of the **while** statement is uniquely determined by its loop body execution state sequence. Lemma 4.6 shows that a state sequence satisfying the four conditions in the lemma is the loop body execution state sequence. In this way the representation of the **while** statement can be converted into a representation of the proposition "there exists a state sequence satisfying all four conditions of Lemma 4.6" in Π. In Chapter 4 we mentioned that the difficulty of representing the above proposition in Π lies in the representation of condition (4). We also briefly introduced the solution of Gödel, i.e., the representation of the loop body execution state

sequence using a matrix, which is further represented by a natural number. Every element of the matrix can be found through this natural number. According to the idea introduced in Chapter 4, we shall describe the representation of the **while** statement in detail step by step.

In what follows we strictly prove Lemma 4.7 given in Chapter 4, i.e., to construct the function β and natural number a. This proof refers to the proof in Section 6.4 of [Schoenfield, 1967].

Lemma A3.1 (Gödel). *There exists a function $\beta(x,y)$ defined on $\mathbb{N}$, which is representable in Π, such that for an arbitrary sequence $a_0, a_1, \ldots, a_{n-1}$ in $\mathbb{N}$, there exists a natural number a satisfying $\beta(a,i) = a_i$ and $\beta(a,i) \leqslant a-1$, where $i < n$.*

Proof. The key of the proof is to construct the natural number a and function β satisfying the conditions of the lemma. In the following proof we call a the *generator* of the sequence $a_0, a_1, \ldots, a_{n-1}$ and β the *generating function*. The process of constructing the generator a from the sequence $a_0, a_1, \ldots, a_{n-1}$ is as follows. First, from the perspective of programming, we need to match an element a_i in the sequence to another natural number b_i which amounts to the temporary storage address of a_i. Namely, we need to find an injection $OP : (a_i, i) \mapsto b_i$. In this way for different i, the temporary addresses b_i of a_i are different. Theoretically speaking, once the temporary address b_i is known, the corresponding element a_i and its subscript i are also known. We define the function OP as

$$OP(x,i) = (x+i)\cdot(x+i)+x+1$$

and can prove that it possesses the property

$$OP(x,i) = OP(y,j) \text{ if and only if } x = y, i = j,$$

i.e., OP is injective. Here $OP(a_i,i)$ can be regarded as the temporary address b_i of a_i. Since OP is composed of $+,\cdot$ only, evidently it is representable in Π and its representation is

$$f(x,i) := (x+i)\cdot(x+i)+Sx.$$

Next we need to find a method to construct the generator a of the sequence $a_0, a_1, \ldots, a_{n-1}$ through the temporary addresses of a_i. Namely, we need to find a method to construct the generator a through the temporary address sequence $OP(a_0,0), OP(a_1,1), \ldots, OP(a_{n-1}, n-1)$.

It is not difficult to prove the following conclusion. Suppose that c is an arbitrary natural number greater than 0 and $1 \leqslant g, h \leqslant c$. Let z be the least common multiple of $1, 2, \ldots, c$. If $g \neq h$, then $1+g\cdot z$ is coprime with $1+h\cdot z$. Thus $1+h\cdot z$ is divisible by $1+g\cdot z$ if and only if $g = h$.

Based on the temporary addresses $OP(a_i,i)$ of a_i, let us define

$$c := \max_{0\leqslant i<n}\{OP(a_i,i)+1\}.$$

We also define z being the least common multiple of $1, 2, \ldots, c$. Let the address of a_i be $AD(a_i,i) := 1+(OP(a_i,i)+1)\cdot z$. Since AD is also composed of $+$ and $\cdot$, $AD(x,i)$ is also

representable in Π. Its representation is

$$S((Sf(x,i))\cdot z).$$

Since OP is an injection, when $x < a_i$, $OP(x,i) \neq OP(a_i,i)$ and for $0 \leqslant j < n$, if $j \neq i$, then $OP(x,i) \neq OP(a_j,j)$. Thus when $x < a_i$, for every $0 \leqslant j < n$, $OP(x,i)+1 \neq OP(a_j,j)+1$. According to the conclusion in the last paragraph, for every $0 \leqslant j < n$, $AD(a_j,j)$ is not divisible by $AD(x,i)$ when $x < a_i$. This implies that a_i is the least natural number x such that

$$\prod_{j=0}^{n-1} AD(a_j,j) \text{ is divisible by } AD(x,i) \quad (\text{where } \prod \text{ denotes the product}).$$

Let the binary relation $Div(a,b)$ denote a being divisible by b. Then $Div(a,b)$ can be represented in Π by the formula

$$D(a,b) := \exists d((\neg(a < d)) \wedge a \doteq d \cdot b).$$

Thus $Div(a,b)$ is a representable relation in Π. Let $y := \prod_{j=0}^{n-1} AD(a_j,j)$. Then a_i is the least natural number x such that the relation

$$Div(y, AD(x,i)) \tag{A3.1}$$

holds. Thus for a given i, we can start from $x = 0$ to find a_i one by one by checking the divisibility of y by $AD(x,i)$. This shows that we can find all the elements of the sequence through y and z. If we define the generator of the sequence being $a := OP(y,z)$, then y and z are uniquely determined by a since OP is injective. As a result, we can find all the elements of the sequence $a_0, a_1, \ldots, a_{n-1}$ through a and the generating function β constructed in the following.

By formula (A3.1) and the above definition of a, a_i is the least natural number x such that both

$$a = OP(y,z) \text{ and } Div(y, AD(x,i)) \tag{A3.2}$$

hold. The constants y and z in the above formula are the values of the bound variables y and z that make the following proposition satisfiable:

there exist $y < a$ and $z < a$ such that $a = OP(y,z)$ and $Div(y, AD(x,i))$ hold.

Suppose that $\cdots x \cdots$ is a satisfiable proposition on $\mathbb{N}$ that is representable in Π. We define $\mu x(\cdots x \cdots)$ as

$$\mu x(\cdots x \cdots) := x,\ x \text{ is the least natural number such that the proposition } \cdots x \cdots \text{ holds.}$$

If the representation of the proposition $\cdots x \cdots$ in Π is $P(x)$, then the formula

$$P[y/x] \wedge \forall x(x < y \rightarrow \neg P(x))$$

represents the function $y = \mu x(\cdots x \cdots)$. Therefore $\mu x(\cdots x \cdots)$ is a representable function in Π. Let the proposition Q be:

$$x \leqslant a-1, \text{ and there exist } y < a \text{ and } z < a \text{ such that both } a = OP(y,z) \text{ and } Div(y, AD(x,i)) \text{ hold.}$$

Then the representation of Q in Π is

$$\neg(a-1 < x) \wedge \exists y \exists z (y < a \wedge z < a \wedge a \doteq f(y,z) \wedge D(y, S((Sf(x,i)) \cdot z))).$$

Summarizing the above discussions, we define

$$\beta(a,i) := \mu x(Q).$$

By (A3.2), $\beta(a,i) = a_i$. If we let $A(x,y,h)$ represent the formula

$$\neg(x-1 < h) \wedge \exists u \exists v (u < x \wedge v < x \wedge x \doteq f(u,v) \wedge D(u, S((Sf(h,y)) \cdot v))),$$

then the representation of the function $z = \beta(x,y)$ in Π is

$$B(x,y,z) := A(x,y,h)[z/h] \wedge \forall h (h < z \to \neg A(x,y,h)).$$

Thus β is a representable function in Π and we obtain the generating function β needed. □

Since $B(x,y,z)$ is the representation of the function $z = \beta(x,y)$ in Π, it is evident that the following lemma holds.

Lemma A3.2. *If $\beta(a,i) = a_i$, then $\Pi \vdash B[S^a 0, S^i 0, S^{a_i} 0]$ is provable; if $\beta(a,i) \neq a_i$, then $\Pi \vdash \neg B[S^a 0, S^i 0, S^{a_i} 0]$ is provable.*

It is only after the length of a sequence is specified that we can find all the elements of the sequence from its generator. Nonetheless the sequence generator obtained by the above method does not contain any information about the length of the sequence. This drawback can be overcome by adding the sequence length as the first element of the sequence. The generator of this new sequence obtained is the sequence number that will be introduced in the following definition.

Definition A3.1 (Sequence number). Suppose that $a_1, \ldots, a_n$ is a sequence on $\mathbb{N}$. The proposition Q is

$$\beta(x,0) = n, \text{ and } \beta(x,1) = a_1, \ldots, \text{ and } \beta(x,n) = a_n.$$

We call

$$Sq(a_1, \ldots, a_n) := \mu x(Q)$$

the *sequence number* of $a_1, \ldots, a_n$.

According to the above definition and Lemma 4.7 (Lemma A3.1 in this appendix), the sequence number is a generator that is irrelevant to the length of the sequence. From the perspective of programming, it amounts to a pointer to an array, i.e., the storage address of the array. We can compute the length of a sequence as well as every element of it through its sequence number and the β function.

Since the number of variables used in a **while** statement is always finite, for any **while** statement with k variables, we can assume that its $(i+1)$th loop body execution state is $\sigma_i := (x_1^i \mapsto m_1^i, \ldots, x_k^i \mapsto m_k^i)$. In this way the value of the variable x_j in σ_i is $[x_j]_{\sigma_i} = m_j^i$ with $1 \leqslant j \leqslant k$. Thus the values of the variable x_j in the sequence of states $\{\sigma_i\}_0^l$ also constitute a natural number sequence $\{m_j^0, m_j^1, \ldots, m_j^l\}$ with $1 \leqslant j \leqslant k$. The loop body execution state sequence can be represented by the following $(l+1) \times k$ natural number matrix $M[l+1][k]$:

$$M[l+1][k] := \begin{pmatrix} m_1^0 & m_2^0 & \ldots & m_k^0 \\ m_1^1 & m_2^1 & \ldots & m_k^1 \\ \vdots & \vdots & & \vdots \\ m_1^l & m_2^l & \ldots & m_k^l \end{pmatrix}. \tag{A3.3}$$

According to Definition A3.1, the idea of the sequence number is to generate the length of a sequence and all its elements by a generator. Thus we can use a generator to generate the numbers of rows and columns as well as all the elements of the matrix (A3.3). Specifically, we have the following definition.

Definition A3.2 (Matrix number). Suppose that $M[l+1][k]$ is the $(l+1) \times k$ matrix defined in (A3.3). The proposition Q is

$$\beta(x,0) = l+1, \text{ and } \beta(x,1) = k, \text{ and } \beta(x,2) = M[0][1], \ldots, \text{ and } \beta(x, i \cdot k + j + 1) = M[i][j], \ldots, \text{ and } \beta(x,(l+1) \cdot k + 1) = M[l][k].$$

We call the least natural number such that the proposition Q holds the *matrix number* of $M[l+1][k]$ and denote it as

$$Matrix(M[0][1], \ldots, M[i][j], \ldots, M[l][k]) := \mu x(Q).$$

For convenience, we use m to represent the matrix number of the previously defined matrix $M[l+1][k]$. By definition, the generating function γ of the $(l+1) \times k$ matrix is

$$\gamma(m,i,j) := \beta(m, i \cdot k + j + 1).$$

Obviously $\gamma(m,i,j) = M[i][j] = m_j^i$.

The role of this set of functions can be explained by using the idea of indirect addressing in programming. The natural number m can be regarded as the storage address of the matrix $M[l+1][k]$, i.e., the 2-dimensional array in the C language. The function γ can be regarded as the subscript operational symbol [ ] in the C language. It outputs the element on the ith row and jth column of the 2-dimensional array.

By Definition A3.2, the generator of the above matrix is m. Thus the loop body execution state sequence can be generated by the natural number m. The following lemma readily follows from Lemma 4.6.

Lemma A3.3. *Let m be a natural number and suppose $\sigma = (x_1 \mapsto m_1, \ldots, x_k \mapsto m_k)$ and $\sigma' = (y_1 \mapsto n_1, \ldots, y_k \mapsto n_k)$. Then m is the matrix number of the matrix corresponding to the loop body execution state sequence of the **while** statement* **while** $0 < x_1$ **do** α' *if and only if it satisfies the following four conditions.*

*L_1: $\beta(m,0) = l+1$, where l is the number of loops; $\beta(m,1) = k$, where k is the number of variables used in the **while** statement.*

L_2: $\gamma(m,0,j) = m_j$, where $1 \leqslant j \leqslant k$.

L_3: $\gamma(m,l,j) = n_j$, where $1 \leqslant j \leqslant k$, and the loop condition $0 < [x_1]_{\sigma'}$ does not hold.

L_4: For every $0 \leqslant i < l$, the initial state of the loop body α' for the $(i+1)$th loop is $\sigma_i = (x_1^i \mapsto \gamma(m,i,1), \ldots, x_k^i \mapsto \gamma(m,i,k))$, the loop condition $0 < [x_1]_{\sigma_i}$ holds, and the terminating state after the execution is $\sigma_{i+1} = (x_1^{i+1} \mapsto \gamma(m,i+1,1), \ldots, x_k^{i+1} \mapsto \gamma(m,i+1,k))$.

In this way the representation of the **while** statement in Π is converted into the representation of the proposition "there exists a natural number m such that the conditions L_1, L_2, L_3, and L_4 hold" in Π. Hence we need to find the representation $C(x,i,j,z)$ of the function $z = \gamma(x,i,j)$ in Π.

According to the representation of $z = \beta(x,y)$ in Π, the representation of the function $z = \gamma(x,i,j)$ in Π is defined as follows.

Definition A3.3 (Representation of $\gamma(x,i,j)$ in Π). Let k be a constant representing the number of variables used in a **while** statement. The representation $C(x,i,j,z)$ in Π of the function $z = \gamma(x,i,j)$ is defined as follows:

$$C(x,i,j,z) := B(x, i \cdot S^k 0 + j + S0, z).$$

Since we only consider the P-procedure with 3 variables, in the following discussions let us take $k = 3$. For simplicity, let $G(x,i,\tau)$ denote the formula

$$(C(x,i,S0,[x_1]_\tau) \wedge C(x,i,S^2 0,[x_2]_\tau) \wedge C(x,i,S^3 0,[x_3]_\tau)).$$

According to Lemma A3.2, the following lemma holds.

Lemma A3.4. *Let $\sigma := (x_1 \mapsto m_1, x_2 \mapsto m_2, x_3 \mapsto m_3)$ whose corresponding set of variables is $\tau = \{x_1, x_2, x_3\}$. If $\sigma = (x_1 \mapsto \gamma(m,i,1), x_2 \mapsto \gamma(m,i,2), x_3 \mapsto \gamma(m,i,3))$, then*

$$\Pi \vdash G(S^m 0, S^i 0, \tau)[S^{m_1} 0, S^{m_2} 0, S^{m_3} 0]$$

is provable. Otherwise

$$\Pi \vdash \neg G(S^m 0, S^i 0, \tau)[S^{m_1} 0, S^{m_2} 0, S^{m_3} 0]$$

is provable.

After the above preparations, now we can describe the representations in Π of the four conditions in Lemma A3.3.

(1) The representation of the condition L_1 in Π is

$$F_1(S^m0,l) := B(S^m0,0,Sl) \wedge B(S^m0,S0,S^30)$$

with l being the representation of the number of loops in Π. The meaning of this formula is that the length of the loop body execution state sequence represented by m is $l+1$ and the number of variables equals 3.

(2) The representation of the condition L_2 in Π is

$$F_2(S^m0,\tau) := G(S^m0,0,\tau).$$

The meaning of this formula is that the value of a variable in the first state of the loop body execution state sequence represented by m equals its value in the initial state of the **while** statement.

(3) The representation of the condition L_3 in Π is

$$F_3(S^m0,l,\tau') := G(S^m0,l,\tau') \wedge \neg(0 < [x_1]_{\tau'}).$$

The meaning of $G(S^m0,l,\tau')$ is that the value of a variable in the $(l+1)$th state of the loop body execution state sequence represented by m equals its value in the terminating state of the **while** statement. The meaning of the formula $\neg(0 < [x_1]_{\tau'})$ is: the condition of the **while** statement does not hold in the terminating state of the loop body.

(4) The representation of the condition L_4 in Π is

$$\begin{aligned} F_4(S^m0,l) := \forall j(j < l \to \exists u_1 \exists u_2 \exists u_3 \exists v_1 \exists v_2 \exists v_3 \\ (G(S^m0,j,\tau_u) \wedge G(S^m0,Sj,\tau_v) \wedge 0 < [x_1]_{\tau_u} \wedge T_{\alpha'}(\tau_u,\tau_v))). \end{aligned}$$

The meaning of the whole formula is that for every $j < l$, there exist a state σ_u corresponding to τ_u and a state σ_v corresponding to τ_v such that the value of a variable in σ_u equals its value in the jth state of the loop body execution state sequence represented by m; the value of the variable in σ_v equals its value in the $(j+1)$th state of the loop body execution state sequence represented by m. The condition of the **while** statement holds in σ_u. The formula obtained from the formula $T_{\alpha'}(\tau_u,\tau_v)$ by substituting its free variables by the values of the corresponding variables in σ_u and σ_v still holds.

In this way the representation $T_\alpha(\tau,\tau')$ of the **while** statement α in Π is

$$\exists w \exists l(F_1(w,l) \wedge F_2(w,\tau) \wedge F_3(w,l,\tau') \wedge F_4(w,l)).$$

In the following we introduce the concept of the characteristic number of the **while** statement.

Definition A3.4 (Characteristic number of the while statement). We call m the *characteristic number* of the **while** statement α if m satisfies the conditions L_1, L_2, L_3, and

L_4. In particular, suppose that the variables used in α are $x_1,\ldots,x_k$ and the loop body execution state sequence corresponding to α is $\{\sigma_i\}_0^l$. If

$$M[l+1][k] := \begin{pmatrix} [x_1]_{\sigma_0} & [x_2]_{\sigma_0} & \cdots & [x_k]_{\sigma_0} \\ [x_1]_{\sigma_1} & [x_2]_{\sigma_1} & \cdots & [x_k]_{\sigma_1} \\ \vdots & \vdots & & \vdots \\ [x_1]_{\sigma_l} & [x_2]_{\sigma_l} & \cdots & [x_k]_{\sigma_l} \end{pmatrix},$$

then $Matrix(M[0][1],\ldots,M[0][k],M[1][1],\ldots,M[l][k])$ is the minimal characteristic number of the **while** statement α.

A3.2 Representability of the *P*-procedure body

According to the above discussions, we can inductively define the representations of the *P*-procedure statements in Π as follows.

Definition A3.5 (Representations of the *P*-procedure statements in Π). The representation $T_\alpha(\tau,\tau')$ of the *P*-procedure statement α in Π is inductively defined as follows.

(1) If α is the assignment statement $x_3 := e$, then its representation $T_{x_3:=e}(\tau,\tau')$ in Π is defined as

$$(y_1 \doteq x_1) \wedge (y_2 \doteq x_2) \wedge (y_3 \doteq [e]_\tau).$$

(2) If α is the **if** statement **if** $0 < x_1$ **then** α_1 **else** α_2, and the representations of α_1 and α_2 in Π are $T_{\alpha_1}(\tau,\tau')$ and $T_{\alpha_2}(\tau,\tau')$ respectively, then its representation $T_{\textbf{if}\,0<x_1\,\textbf{then}\,\alpha_1\,\textbf{else}\,\alpha_2}(\tau,\tau')$ in Π is defined as

$$\mathrm{cond}(0 < [x_1]_\tau, T_{\alpha_1}(\tau,\tau'), T_{\alpha_2}(\tau,\tau')).$$

(3) If α is the sequential statement $\alpha_1;\alpha_2$, and the representations of α_1 and α_2 in Π are $T_{\alpha_1}(\tau,\tau_z)$ and $T_{\alpha_2}(\tau_z,\tau')$ respectively, then its representation $T_{\alpha_1;\alpha_2}(\tau,\tau')$ in Π is defined as

$$\exists z_1 \exists z_2 \exists z_3 (T_{\alpha_1}(\tau,\tau_z) \wedge T_{\alpha_2}(\tau_z,\tau')).$$

(4) If α is the **while** statement **while** $0 < x_1$ **do** α', and the representation of α' in Π is $T_{\alpha'}(\tau_u,\tau_v)$, then its representation $T_{\textbf{while}\ 0<x_1\ \textbf{do}\ \alpha'}(\tau,\tau')$ in Π is defined as

$$\exists w \exists l (F_1(w,l) \wedge F_2(w,\tau) \wedge F_3(w,l,\tau') \wedge F_4(w,l)).$$

(5) If α is the **call** statement $F(m_1,m_2,x_3)$, and the representation of its body α' in Π is $T_{\alpha'}(\tau_u,\tau_v)$, then its representation $T_{F(m_1,m_2,x_3)}(\tau,\tau')$ in Π is defined as

$$(y_1 \doteq x_1) \wedge (y_2 \doteq x_2) \wedge (\exists v_1 \exists v_2 (T_{\alpha'}(\tau_u,\tau_v)[S^{m_1}0/u_1, S^{m_2}0/u_2, x_3/u_3, y_3/v_3])).$$

We can prove that the following lemmas hold.

Lemma A3.5.

$$\frac{\Gamma \vdash \neg A, \Theta}{\Gamma \vdash \neg(A \wedge B), \Theta}$$

Proof.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma \vdash \neg A, \Theta \qquad \neg A \vdash \neg A, \neg B}{\Gamma \vdash \neg A, \neg B, \Theta}}{\Gamma, B \vdash \neg A, \Theta}}{\Gamma, A, B \vdash \Theta}}{\Gamma, A \wedge B \vdash \Theta}}{\Gamma \vdash \neg(A \wedge B), \Theta}$$

□

Lemma A3.6.

$$\frac{\Gamma \vdash \forall x \neg A, \Theta}{\Gamma \vdash \neg \exists x A, \Theta}$$

Proof.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma \vdash \forall x \neg A, \Theta}{\Gamma \vdash \neg A[y/x], \Theta}}{\Gamma, A[y/x] \vdash \Theta}}{\Gamma, \exists x A \vdash \Theta}}{\Gamma \vdash \neg \exists x A, \Theta}$$

□

In what follows we prove that the representability lemma of the procedure body holds.

Lemma A3.7 (Representability of the procedure body). *Let the procedure body be* α. *Suppose that the initial state is* $\sigma = (x_1 \mapsto m_1, x_2 \mapsto m_2, x_3 \mapsto m_3)$, *the terminating state of executing the procedure body is* $\sigma' = (y_1 \mapsto k_1, y_2 \mapsto k_2, y_3 \mapsto k_3)$, *and* $\sigma' = (y_1 \mapsto n_1, y_2 \mapsto n_2, y_3 \mapsto n_3)$.

(1) *If* $\sigma' = \sigma^t$, *i.e.,* $n_1 = k_1$, $n_2 = k_2$, *and* $n_3 = k_3$ *hold, then*

$$\Pi \vdash T_\alpha(\tau,\ \tau')[S^{m_1}0, S^{m_2}0, S^{m_3}0, S^{n_1}0, S^{n_2}0, S^{n_3}0] \text{ is provable.}$$

(2) *If* $\sigma' \neq \sigma^t$, *i.e.,* $n_1 \neq k_1$, $n_2 \neq k_2$, *or* $n_3 \neq k_3$ *holds, then*

$$\Pi \vdash \neg T_\alpha(\tau,\ \tau')[S^{m_1}0, S^{m_2}0, S^{m_3}0, S^{n_1}0, S^{n_2}0, S^{n_3}0] \text{ is provable.}$$

Proof. We make a structural induction on the procedure body α.

1. α is the assignment statement $x_3 := e$.

For the assignment statement $x_3 := e$, according to its operational semantics we have $k_1 = m_1$, $k_2 = m_2$, and $k_3 = [e]_\sigma$.

If $\sigma' = \sigma^t$, then $n_1 = m_1$, $n_2 = m_2$, and $n_3 = [e]_\sigma$. According to Lemma 4.3 in Chapter 4, $\Pi \vdash S^{n_1}0 \doteq S^{m_1}0$, $\Pi \vdash S^{n_2}0 \doteq S^{m_2}0$, and $\Pi \vdash S^{n_3}0 \doteq S^{[e]\sigma}0$ are all provable. Then Lemma 4.4 in Chapter 4 implies that $\Pi \vdash S^{[e]\sigma}0 \doteq Tr([e]_\sigma)$. Therefore

$$\Pi \vdash S^{n_1}0 \doteq S^{m_1}0 \wedge S^{n_2}0 \doteq S^{m_2}0 \wedge S^{n_3}0 \doteq Tr([e]_\sigma)$$

is provable, that is,

$$\Pi \vdash T_\alpha(\tau,\ \tau')[S^{m_1}0, S^{m_2}0, S^{m_3}0, S^{n_1}0, S^{n_2}0, S^{n_3}0]$$

is provable.

If $\sigma' \neq \sigma^t$, then $n_1 \neq m_1$, $n_2 \neq m_2$, or $n_3 \neq m_3$. According to Lemma 4.3 in Chapter 4, $\Pi \vdash \neg S^{n_1}0 \doteq S^{m_1}0$ is provable, or $\Pi \vdash \neg S^{n_2}0 \doteq S^{m_2}0$ is provable, or $\Pi \vdash \neg S^{n_3}0 \doteq S^{[e]\sigma}0$ is provable. Then Lemma 4.4 in Chapter 4 implies that $\Pi \vdash S^{[e]\sigma}0 \doteq Tr([e]_\sigma)$. Thus

$$\Pi \vdash \neg S^{n_1}0 \doteq S^{m_1}0 \vee \neg S^{n_2}0 \doteq S^{m_2}0 \vee \neg S^{n_3}0 \doteq Tr([e]_\sigma)$$

is provable, that is,

$$\Pi \vdash \neg(S^{n_1}0 \doteq S^{m_1}0 \wedge S^{n_2}0 \doteq S^{m_2}0 \wedge S^{n_3}0 \doteq Tr([e]_\sigma))$$

is provable, that is,

$$\Pi \vdash \neg T_\alpha(\tau,\tau')[S^{m_1}0, S^{m_2}0, S^{m_3}0, S^{n_1}0, S^{n_2}0, S^{n_3}0]$$

is provable.

2. α is the **if** statement **if** $0 < x_1$ **then** α_1 **else** α_2.

For the case of $m_1 > 0$, obviously $\Pi \vdash 0 < S^{m_1}0$ is provable. Namely, $\Pi \vdash (0 < [x_1]_\tau)[S^{m_1}0]$ is provable. Thus

$$\Pi, \neg(0 < [x_1]_\tau)[S^{m_1}0] \vdash T_{\alpha_2}(\tau,\tau')[S^{m_1}0, S^{m_2}0, S^{m_3}0, S^{n_1}0, S^{n_2}0, S^{n_3}0]$$

is provable as well. The meaning of the **if** statement implies that in this case α_1 is executed, i.e., α_1 is executed in σ to obtain the terminating state σ^t.

According to the induction hypothesis, α_1 satisfies the lemma. Thus when $\sigma' = \sigma^t$,

$$\Pi \vdash T_{\alpha_1}(\tau,\tau')[S^{m_1}0, S^{m_2}0, S^{m_3}0, S^{n_1}0, S^{n_2}0, S^{n_3}0]$$

is provable. Therefore

$$\Pi, (0 < [x_1]_\tau)[S^{m_1}0] \vdash T_{\alpha_1}(\tau,\tau')[S^{m_1}0, S^{m_2}0, S^{m_3}0, S^{n_1}0, S^{n_2}0, S^{n_3}0]$$

is also provable. The $\wedge$-*R* rule and Lemma A3.5 further indicate that

$$\Pi \vdash T_\alpha(\tau,\tau')[S^{m_1}0, S^{m_2}0, S^{m_3}0, S^{n_1}0, S^{n_2}0, S^{n_3}0]$$

is provable.

Similarly, when $\sigma' \neq \sigma^t$, according to the induction hypothesis,

$$\Pi \vdash \neg T_{\alpha_1}(\tau, \tau')[S^{m_1}0, S^{m_2}0, S^{m_3}0, S^{n_1}0, S^{n_2}0, S^{n_3}0]$$

is provable. Then $\Pi \vdash (0 < [x_1]_\tau)[S^{m_1}0]$ being provable together with the $\neg$ rule and $\wedge$-*R* rule indicate that

$$\Pi \vdash (\neg\neg(0 < [x_1]_\tau)[S^{m_1}0]) \wedge (\neg T_{\alpha_1}(\tau, \tau')[S^{m_1}0, S^{m_2}0, S^{m_3}0, S^{n_1}0, S^{n_2}0, S^{n_3}0])$$

is provable. Namely,

$$\Pi \vdash \neg((0 < [x_1]_\tau) \to T_{\alpha_1}(\tau, \tau'))[S^{m_1}0, S^{m_2}0, S^{m_3}0, S^{n_1}0, S^{n_2}0, S^{n_3}0]$$

is provable. Lemma A3.5 further indicates that this amounts to

$$\Pi \vdash \neg T_\alpha(\tau, \tau')[S^{m_1}0, S^{m_2}0, S^{m_3}0, S^{n_1}0, S^{n_2}0, S^{n_3}0]$$

being provable.

The case of $m_1 = 0$ can be similarly proved.

3. α is the sequential statement $\alpha_1;\ \alpha_2$.

Let the state obtained by executing α_1 in the initial state σ be denoted as $\sigma_z := (z_1 \mapsto s_1, z_2 \mapsto s_2, z_3 \mapsto s_3)$. Then by the meaning of the sequential statement, the state obtained by executing α_2 in the state σ_z is σ^t.

According to the induction hypothesis,

$$\Pi \vdash T_{\alpha_1}(\tau, \tau_z)[S^{m_1}0, S^{m_2}0, S^{m_3}0, S^{s_1}0, S^{s_2}0, S^{s_3}0]$$

is provable; if $\sigma' = \sigma^t$, then

$$\Pi \vdash T_{\alpha_2}(\tau_z, \tau')[S^{s_1}0, S^{s_2}0, S^{s_3}0, S^{n_1}0, S^{n_2}0, S^{n_3}0]$$

is provable. Then the $\wedge$-*R* rule and $\exists$-*R* rule indicate that

$$\Pi \vdash T_\alpha(\tau, \tau')[S^{m_1}0, S^{m_2}0, S^{m_3}0, S^{n_1}0, S^{n_2}0, S^{n_3}0]$$

is provable as well.

According to the induction hypothesis,

$$\Pi, \neg(z_1 \doteq S^{s_1}0 \wedge z_2 \doteq S^{s_2}0 \wedge z_3 \doteq S^{s_3}0) \vdash \neg T_{\alpha_1}(\tau, \tau_z)[S^{m_1}0, S^{m_2}0, S^{m_3}0]$$

is provable. If $\sigma' \neq \sigma^t$, then by the induction hypothesis,

$$\Pi, z_1 \doteq S^{s_1}0 \wedge z_2 \doteq S^{s_2}0 \wedge z_3 \doteq S^{s_3}0 \vdash \neg T_{\alpha_2}(\tau_z, \tau')[S^{n_1}0, S^{n_2}0, S^{n_3}0]$$

is provable. Lemma A3.5 and the rule of proof by case analysis imply that

$$\Pi \vdash \neg(T_{\alpha_1}(\tau, \tau_z) \wedge T_{\alpha_2}(\tau_z, \tau'))[S^{m_1}0, S^{m_2}0, S^{m_3}0, S^{n_1}0, S^{n_2}0, S^{n_3}0]$$

is provable. Then the $\neg$ rule and $\exists$-L rule indicate that

$$\Pi \vdash \neg\exists z_1\exists z_2\exists z_3(T_{\alpha_1}(\tau,\tau_z)\wedge T_{\alpha_2}(\tau_z,\tau'))[S^{m_1}0,S^{m_2}0,S^{m_3}0,S^{n_1}0,S^{n_2}0,S^{n_3}0]$$

is provable. Namely,

$$\Pi \vdash \neg T_\alpha(\tau,\tau')[S^{m_1}0,S^{m_2}0,S^{m_3}0,S^{n_1}0,S^{n_2}0,S^{n_3}0]$$

is provable.

4. α is the **while** statement **while** $0<x_1$ **do** α'.

If $\sigma'=\sigma^t$, then let m_0 be the minimal characteristic number of the **while** statement α and l_0 be the number of loops. Lemmas A3.3, A3.2, and A3.4 indicate that

$$\Pi \vdash (F_1(S^{m_0}0,S^{l_0}0)\wedge F_2(S^{m_0}0,\tau)\wedge F_3(S^{m_0}0,S^{l_0}0,\tau')\wedge F_4(S^{m_0}0,S^{l_0}0))$$
$$[S^{m_1}0,S^{m_2}0,S^{m_3}0,S^{n_1}0,S^{n_2}0,S^{n_3}0]$$

is provable. According to the $\exists$-R rule, this amounts to

$$\Pi \vdash T_\alpha(\tau,\tau')[S^{m_1}0,S^{m_2}0,S^{m_3}0,S^{n_1}0,S^{n_2}0,S^{n_3}0]$$

being provable.

If $\sigma'\neq\sigma^t$, then the proof can be divided into the following two cases.

(1) If m is the characteristic number of the **while** statement, then the discussion can be further divided into two cases as follows.

i) If l_0 is the number of loops of the **while** statement, then according to the operational semantics of the **while** statement, $\sigma^t=(y_1\mapsto\gamma(m,l_0,1),y_2\mapsto\gamma(m,l_0,2),y_3\mapsto\gamma(m,l_0,3))$ holds. By Lemma A3.4, the sequent

$$\Pi \vdash \neg G(S^m0,S^{l_0}0,\tau')[S^{n_1}0,S^{n_2}0,S^{n_3}0]$$

is provable. Then Lemma A3.5 indicates that the sequent

$$\Pi \vdash \neg F_3(S^m0,S^{l_0}0,\tau')[S^{n_1}0,S^{n_2}0,S^{n_3}0]$$

is provable. Lemma A3.5 implies that the sequent

$$\Pi \vdash \neg(F_1(S^m0,S^{l_0}0)\wedge F_2(S^m0,\tau)\wedge F_3(S^m0,S^{l_0}0,\tau')\wedge F_4(S^m0,S^{l_0}0))$$
$$[S^{m_1}0,S^{m_2}0,S^{m_3}0,S^{n_1}0,S^{n_2}0,S^{n_3}0]$$

is also provable.

ii) If l_0 is not the number of loops of the **while** statement, then $l_0+1\neq\beta(m,0)$ holds. Lemma A3.2 implies that the sequent

$$\Pi \vdash \neg B(S^m0,0,S^{l_0+1}0)$$

is provable. By the $\vee$-R rule,

$$\Pi \vdash \neg B(S^m0,0,S^{l_0+1}0)\vee\neg B(S^m0,S0,S^30)$$

is provable. Then according to the $\neg$ rule and $\wedge$-L rule, the sequent

$$\Pi \vdash \neg F_1(S^m 0, S^{l_0} 0)$$

is provable. Lemma A3.5 implies that the sequent

$$\Pi \vdash \neg(F_1(S^m 0, S^{l_0} 0) \wedge F_2(S^m 0, \tau) \wedge F_3(S^m 0, S^{l_0} 0, \tau') \wedge F_4(S^m 0, S^{l_0} 0)) \\ [S^{m_1} 0, S^{m_2} 0, S^{m_3} 0, S^{n_1} 0, S^{n_2} 0, S^{n_3} 0]$$

is also provable.

(2) If m is not the characteristic number of the **while** statement, then the four conditions of Lemma A3.3 cannot be satisfied simultaneously. Thus the proof may be divided into the following four cases.

(a) The condition L_1 is not satisfied, i.e., $\beta(m,0) \neq l_0 + 1$ or $\beta(m,1) \neq 3$ holds. According to Lemma A3.2, the sequent

$$\Pi \vdash \neg B(S^m 0, 0, S^{l_0+1} 0) \vee \neg B(S^m 0, S0, S^3 0)$$

is provable. Then the $\neg$ rule and $\wedge$-L rule indicate that the sequent

$$\Pi \vdash \neg F_1(S^m 0, S^{l_0} 0)$$

is provable. By Lemma A3.5, the sequent

$$\Pi \vdash \neg(F_1(S^m 0, S^{l_0} 0) \wedge F_2(S^m 0, \tau) \wedge F_3(S^m 0, S^{l_0} 0, \tau') \\ \wedge F_4(S^m 0, S^{l_0} 0))[S^{m_1} 0, S^{m_2} 0, S^{m_3} 0, S^{n_1} 0, S^{n_2} 0, S^{n_3} 0]$$

is provable as well.

(b) The condition L_2 is not satisfied, i.e., $\sigma \neq (x_1 \mapsto \gamma(m,0,1),\ x_2 \mapsto \gamma(m,0,2),\ x_3 \mapsto \gamma(m,0,3))$. According to Lemma A3.4, the sequent

$$\Pi \vdash \neg G(S^m 0, 0, \tau)[S^{m_1} 0, S^{m_2} 0, S^{m_3} 0]$$

is provable. Lemma A3.5 further implies that this amounts to

$$\Pi \vdash \neg F_2(S^m 0, \tau)[S^{m_1} 0, S^{m_2} 0, S^{m_3} 0]$$

being provable. By Lemma A3.5, the sequent

$$\Pi \vdash \neg(F_1(S^m 0, S^{l_0} 0) \wedge F_2(S^m 0, \tau) \wedge F_3(S^m 0, S^{l_0} 0, \tau') \\ \wedge F_4(S^m 0, S^{l_0} 0))[S^{m_1} 0, S^{m_2} 0, S^{m_3} 0, S^{n_1} 0, S^{n_2} 0, S^{n_3} 0]$$

is also provable.

(c) The condition L_3 is not satisfied, i.e., $\sigma' \neq (y_1 \mapsto \gamma(m,l_0,1),\ y_2 \mapsto \gamma(m,l_0,2),\ y_3 \mapsto \gamma(m,l_0,3))$ or $0 < [x_1]_{\sigma'}$ holds. According to Lemma A3.4, the sequent

$$\Pi \vdash \neg G(S^m 0, S_0^l 0, \tau')[S^{n_1} 0, S^{n_2} 0, S^{n_3} 0] \vee \neg\neg(0 < [x_1]'_\tau)[S^{n_1} 0]$$

is provable. Then the $\neg$ rule and $\wedge$-L rule indicate that the sequent

$$\Pi \vdash \neg F_3(S^m 0, S_0^l 0, \tau')[S^{n_1}0, S^{n_2}0, S^{n_3}0]$$

is provable. By Lemma A3.5, the sequent

$$\Pi \vdash \neg(F_1(S^m 0, S^{l_0}0) \wedge F_2(S^m 0, \tau) \wedge F_3(S^m 0, S^{l_0}0, \tau') \\ \wedge F_4(S^m 0, S^{l_0}0))[S^{m_1}0, S^{m_2}0, S^{m_3}0, S^{n_1}0, S^{n_2}0, S^{n_3}0]$$

is provable as well.

(d) The condition L_4 is not satisfied. Suppose $\sigma_u := (u_1 \mapsto s_1, u_2 \mapsto s_2, u_3 \mapsto s_3)$ and $\sigma_v := (v_1 \mapsto t_1, v_2 \mapsto t_2, v_3 \mapsto t_3)$. Then there exists a $j_0 < l_0$ such that at least one of the following four cases holds.

i) The initial state $\sigma_u \neq (u_1 \mapsto \gamma(m, j_0, 1), u_2 \mapsto \gamma(m, j_0, 2), u_3 \mapsto \gamma(m, j_0, 3))$. According to Lemma A3.4, the sequent

$$\Pi \vdash \neg G(S^m 0, S^{j_0}0, \tau_u)[S^{s_1}0, S^{s_2}0, S^{s_3}0]$$

is provable. Then by Lemma A3.5, the sequent

$$\Pi \vdash \neg(G(S^m 0, S^{j_0}0, \tau_u) \wedge G(S^m 0, S^{j_0+1}0, \tau_v) \wedge (0 < [x_1]_{\tau_u}) \\ \wedge T_{\alpha'}(\tau_u, \tau_v))[S^{s_1}0, S^{s_2}0, S^{s_3}0, S^{t_1}0, S^{t_2}0, S^{t_3}0]$$

is provable.

ii) $0 < [x_1]_{\sigma_u}$ does not hold. Then

$$\Pi \vdash (\neg 0 < [x_1]_{\tau_u})[S^{s_1}0/u_1]$$

is provable. According to Lemma A3.5, the sequent

$$\Pi \vdash \neg(G(S^m 0, S^{j_0}0, \tau_u) \wedge G(S^m 0, S^{j_0+1}0, \tau_v) \wedge (0 < [x_1]_{\tau_u}) \\ \wedge T_{\alpha'}(\tau_u, \tau_v))[S^{s_1}0, S^{s_2}0, S^{s_3}0, S^{t_1}0, S^{t_2}0, S^{t_3}0]$$

is provable.

iii) The terminating state $\sigma_v \neq (v_1 \mapsto \gamma(m, j_0+1, 1),\ v_2 \mapsto \gamma(m, j_0+1, 2),\ v_3 \mapsto \gamma(m, j_0+1, 3))$. According to Lemma A3.4, the sequent

$$\Pi \vdash \neg G(S^m 0, S^{j_0+1}0, \tau_v)[S^{t_1}0, S^{t_2}0, S^{t_3}0]$$

is provable. Then by Lemma A3.5, the sequent

$$\Pi \vdash \neg(G(S^m 0, S^{j_0}0, \tau_u) \wedge G(S^m 0, S^{j_0+1}0, \tau_v) \wedge (0 < [x_1]_{\tau_u}) \\ \wedge T_{\alpha'}(\tau_u, \tau_v))[S^{s_1}0, S^{s_2}0, S^{s_3}0, S^{t_1}0, S^{t_2}0, S^{t_3}0]$$

is provable.

iv) The initial state $\sigma_u = (u_1 \mapsto \gamma(m, j_0, 1), u_2 \mapsto \gamma(m, j_0, 2), u_3 \mapsto \gamma(m, j_0, 3))$ and the terminating state $\sigma_v = (v_1 \mapsto \gamma(m, j_0+1, 1), v_2 \mapsto \gamma(m, j_0+1, 2), v_3 \mapsto \gamma(m, j_0+1, 3))$. Nonetheless, the terminating state obtained by executing the loop body α' under the initial state σ_u is not σ_v. Then according to the induction hypothesis,

$$\Pi \vdash \neg T_{\alpha'}(\tau_u, \tau_v)[S^{a_3}0, S^{s_1}0, S^{s_2}0, S^{s_3}0, S^{t_1}0, S^{t_2}0, S^{t_3}0]$$

is provable. Lemma A3.5 indicates that the sequent

$$\Pi \vdash \neg(G(S^m 0, S^{j_0}0, \tau_u) \wedge G(S^m 0, S^{j_0+1}0, \tau_v) \wedge (0 < [x_1]_{\tau_u}) \wedge T_{\alpha'}(\tau_u, \tau_v))[S^{s_1}0, S^{s_2}0, S^{s_3}0, S^{t_1}0, S^{t_2}0, S^{t_3}0]$$

is provable.

Summarizing the above four cases, we know that the sequent

$$\Pi \vdash \forall u_1 \forall u_2 \forall u_3 \forall v_1 \forall v_2 \forall v_3 \neg(G(S^m 0, S^{j_0}0, \tau_u) \wedge G(S^m 0, S^{j_0+1}0, \tau_v) \wedge (0 < [x_1]_{\tau_u}) \wedge T_{\alpha'}(\tau_u, \tau_v))$$

is provable. Then according to Lemma A3.6, the sequent

$$\Pi \vdash \neg\exists u_1 \exists u_2 \exists u_3 \exists v_1 \exists v_2 \exists v_3 (G(S^m 0, S^{j_0}0, \tau_u) \wedge G(S^m 0, S^{j_0+1}0, \tau_v) \wedge (0 < [x_1]_{\tau_u}) \wedge T_{\alpha'}(\tau_u, \tau_v))$$

is provable. As $j_0 < l_0$,

$$\Pi \vdash S^{j_0}0 < S^{l_0}0$$

is provable. The $\rightarrow$-*L* rule, $\neg$ rule, and $\exists$-*R* rule imply that

$$\Pi \vdash \exists j \neg(j < S^{l_0}0 \rightarrow \exists u_1 \exists u_2 \exists u_3 \exists v_1 \exists v_2 \exists v_3 (G(S^m 0, S^j 0, \tau_u) \wedge G(S^m 0, S^{j+1}0, \tau_v) \wedge (0 < [x_1]_{\tau_u}) \wedge T_{\alpha'}(\tau_u, \tau_v)))$$

is provable. According to Lemma A3.6,

$$\Pi \vdash \neg F_4(S^m 0, S^{l_0}0)$$

is provable. Lemma A3.5 implies that the sequent

$$\Pi \vdash \neg(F_1(S^m 0, S^{l_0}0) \wedge F_2(S^m 0, \tau) \wedge F_3(S^m 0, S^{l_0}0, \tau') \wedge F_4(S^m 0, S^{l_0}0))[S^{m_1}0, S^{m_2}0, S^{m_3}0, S^{n_1}0, S^{n_2}0, S^{n_3}0]$$

is also provable.

Summarizing the two cases (1) and (2), we have

$$\Pi \vdash \forall w \forall l \neg(F_1(w, l) \wedge F_2(w, \tau) \wedge F_3(w, l, \tau') \wedge F_4(w, l))[S^{m_1}0, S^{m_2}0, S^{m_3}0, S^{n_1}0, S^{n_2}0, S^{n_3}0]$$

being provable. By Lemma A3.6, this amounts to

$$\Pi \vdash \neg T_\alpha(\tau,\tau')[S^{m_1}0, S^{m_2}0, S^{m_3}0, S^{n_1}0, S^{n_2}0, S^{n_3}0]$$

being provable.

5. α is the **call** statement $F(m_1, m_2, x_3)$.

Here the real parameters of F are m_1 and m_2 of the initial state. Let us denote the procedure body of the **call** statement as α'. According to the induction hypothesis, α' satisfies this lemma. The meaning of the **call** statement indicates that $k_1 = m_1$, $k_2 = m_2$, and a terminating state $\sigma_v = (v_1 \mapsto t_1, v_2 \mapsto t_2, v_3 \mapsto t_3)$ with $t_3 = k_3$ is obtained by executing α' in the state $\sigma_u = (u_1 \mapsto m_1, u_2 \mapsto m_2, u_3 \mapsto 0)$. Thus both $\Pi \vdash S^{m_1}0 \doteq S^{k_1}0$ and $\Pi \vdash S^{m_2}0 \doteq S^{k_2}0$ are provable. According to the induction hypothesis, if $n_3 = k_3$, then

$$\Pi \vdash (\exists v_1 \exists v_2 T_{\alpha'}(\tau_u, \tau_v)[S^{m_1}0/u_1, S^{m_2}0/u_2, x_3/u_3, y_3/v_3])[0/x_3, S^{n_3}0/y_3]$$

is provable; if $n_3 \neq k_3$, then

$$\Pi \vdash \neg(\exists v_1 \exists v_2 T_{\alpha'}(\tau_u, \tau_v)[S^{m_1}0/u_1, S^{m_2}0/u_2, x_3/u_3, y_3/v_3])[0/x_3, S^{n_3}0/y_3]$$

is provable. If $\sigma' = \sigma^t$, then $n_i = k_i$ holds for $i = 1, 2, 3$. Thus

$$\Pi \vdash T_\alpha(\tau,\tau')[S^{m_1}0, S^{m_2}0, S^{m_3}0, S^{n_1}0, S^{n_2}0, S^{n_3}0]$$

is also provable. If $\sigma' \neq \sigma^t$, then at least one of $n_i \neq k_i$ holds with $i = 1, 2, 3$. Therefore

$$\Pi \vdash \neg T_\alpha(\tau,\tau')[S^{m_1}0, S^{m_2}0, S^{m_3}0, S^{n_1}0, S^{n_2}0, S^{n_3}0]$$

is provable.

By the method of structural inductions, the lemma is proved. □

Bibliography

[AGM, 1985] C.E. Alchourrón, R. Gärdenfors and D. Makinson, On the Logic of Theory Change: Partial Meet Contraction and Revision Functions, *The Journal of Symbolic Logic* **50** No. 2 (1985), 510–530.

[Enderton, 1972] H.B. Enderton, *A Methematical Introduction to Logic*, Academic Press, New York, 1972.

[McKeon, 1941] *The Basic Works of Aristotle*, edited by R.P. McKeon, p. 198, edition of 1941.

[Backus, 1959] J.W. Backus, The Syntax and Semantics of the Proposed International Algebraic Language of Zürich ACM-GAMM Conference, in: *Proceedings of the International Conference on Information Processing*, pp. 125–131, 1959.

[Blum et al, 1997] L. Blum, F. Cucker, M. Shub and S. Smale, *Complexity and Real Computation*, Springer, New York, 1997.

[Burgess, 1977] J.P. Burgess, *Forcing, Handbook of Mathematical Logic* (Ed. J. Barwise), North-Holland Publishing Company, Amsterdam, pp. 403–452, 1977.

[Church, 1941] A. Church, *The Calculi of Lambda-Conversion*, Princeton University Press, Princeton, NJ, USA, 1941.

[Cohen, 1966] P.J. Cohen, *Set Theory and the Continuum Hypothesis*, Benjamin, Inc., New York, 1966.

[Darwin 1859] C. Darwin, *On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life* (1st ed.), John Murray, London, 1859.

[Darwin 1979] C. Darwin, *The Journal of a Voyage*, in: H.M.S "Beagle", Genesis Publications Ltd., 1979.

[Davis, 1958] M. Davis, *Computability and Unsolvability*, McGraw-Hill Book Company, Inc., New York, 1958.

[Dijkstra, 1976] E.W. Dijkstra, *A Discipline of Programming*, Prentice-Hall PTR, Upper Saddle River, NJ, USA, 1976.

[Ebbinghaus et al, 1994] H.D. Ebbinghaus, J. Flum and W. Thomas, *Mathematical Logic* (2nd Edition), Springer, New York, 1994.

[Einstein, 1921] A. Einstein, *Relativity: The Special & The General Theory* (translated by R.W. Lawson), Methuen & CO. LTD., London, 1921.

[Flew, 1979] A. Flew, *A Dictionary of Philosophy*, Pan Books Ltd., London, 1979.

[Galilei, 1632] G. Galilei, *Dialogo Sopra i due massimi Sistemi del mondo, tolemaico e copernico*, Italy, 1632.

[Gallier, 1986] J.H. Gallier, *Logic for Computer Science: Foundations of Automatic Theorem Proving*, Harper & Row, New York, 1986.

[Gärdenfors, 1988] P. Gärdenfors, *Knowledge in Flux: Modeling the Dynamics of Epistemic States*, Bradford Books, The MIT Press, Cambridge, Massachusetts, 1988.

[Garey and Johnson, 1979] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Company, San Francisco, 1979.

[Gentzen, 1969] G. Gentzen, Investigations into Logical Deduction, in: *The Collected Papers of Gerhard Gentzen* (Ed. M.E. Szabo), North-Holland, Amsterdam, pp. 68–131, 1969.

[Halliday et al, 2000] D. Halliday, R. Resnick and J. Walker, *Fundamentals of Physics* (7th Edition), John Wiley & Sons, Inc., 2000.

[Hilbert, 1899] D. Hilbert, *Grundlagen der Geometrie*, B.G. Teubner, Leipzig, 1899.

[Hilbert, 1925] D. Hilbert, On the Infinite, *Mathematische Annalen* **95** (1925), 161–190.

[Hoare, 1969] C.A.R. Hoare, An Axiomatic Basis for Computer Programming, *Communications of the ACM* **12** No. 10 (1969), 576–580.

[Hopcroft et al, 2006] J.E. Hopcroft, R. Motwani and J.D. Ullman, *Introduction to Automata Theory, Languages and Computation* (3rd edition), Addison-Wesley, 2006.

[Landin, 1964] P.J. Landin, The Mechanical Evaluation of Expressions, *The Computer Journal* **6** No. 4 (1964), 308–320.

[Landau and Lifshitz, 1960] L.D. Landau and E.M. Lifshitz, *Mechanics*, Addison-Wesley, Reading, Massachusetts, 1960.

[Li, 1982] W. Li, An Operational Semantics for Ada Multi-tasking and Exception Handling, in: *Proceedings of AdaTec Conference*, ACM Press, New York, pp. 138–151, 1982.

[Li, 1983] W. Li, *An Operational Approach to Semantics and Translation for Concurrent Progamming Languages*, Ph.D thesis, CST-20-83, University of Edinburgh, 1983.

[Li, 1992] W. Li, An Open Logical System (in Chinese), *Science in China, Ser. A* **22** (1992), 1103–1113.

[Li, 1993] W. Li, A Theory of Requirement Capture and its Applications, in: *TAPSOFT '93: Theory and Practice of Software Development*, LNCS 668, Springer, Berlin, Heidelberg, pp. 406–420, 1993.

[Li, 1994] W. Li, *A Logical Framework for Evolution of Specifications, Programming Languages and Systems — ESOP '94*, LNCS 788, Springer, Berlin, Heidelberg, pp. 394–408, 1994.

[Li, 2000] W. Li, *A Computational Framework for Convergent Agents*, IDEAL '2000, pp. 295–300, 2000.

[Li et al, 2001] W. Li, S. Ma, Y. Sui and K. Xu, *A Logical Framework for Convergent Infinite Computations*, CoRR cs.LO/0105020, 2001.

[Li and Ma, 2004] W. Li and S. Ma, Limits of Theory Sequences over Algebraically Closed Fields and Applications, *Discrete Applied Mathematics* **136** No. 1 (2004), 23–43.

[Li, 2007] W. Li, R-Calculus: An Inference System for Belief Revision, *The Computer Journal* **50** No. 4 (2007), 378–390.

[Milner, 1980] R. Milner, *A Calculus of Communicating Systems*, LNCS 92, Springer, Berlin, Heidelberg, 1980.

[Mo, 1993] S. Mo, *Analysis of Inductive Logic* (in Chinese), Special Issue on Studies in Logic, Philosophical Researches, Supplement, 1993.

[Plotkin, 1981] G.D. Plotkin, *A Structural Approach to Operational Semantics*, DAIMI FN-19, Computer Science Department, Aarhus University, Denmark, 1981.

[Popper, 1959] K. Popper, *The Logic of Scientific Discovery*, Basic Books, Inc., New York, 1959.

[Reiter, 1980] R. Reiter, A Logic for Default Reasoning, *Artificial Intelligence* **13** No. 1–2 (1980), 81–132.

[Robinson, 1965] J.A. Robinson, A Machine-Oriented Logic Based on the Resolution Principle, *Journal of the Association for Computing Machinery* **12** No. 1 (1965), 23–41.

[Shoenfield, 1967] J.R, Schoenfield, *Mathematical Logic*, Addison-Wesley, Reading, Massachusetts, 1967.

[Smullyan, 1968] R.M. Smullyan, *First-order Logic*, Springer, New York, 1968.

[Turing, 1936] A. Turing, On Computable Numbers, With an Application to the Entscheidungsproblem, in: *Proceedings of the London Mathematical Society, Ser. 2* **42** (1936), 230–265.

[Wang, 1987] S. Wang, *Fundamentals of Model Theory* (in Chinese), Science Press, Beijing, 1987.

Index

CP, 56
$GF(X)$, 103
$GN(X)$, 103
P-computability, 80
P-kernel, 76
P-kernel language, 101
P-procedures, 76
T-rules, 6
G inference system, 49
G rule schema, 51
G rules, 50
G system, 49
N-expansion, 143
R-¬ derived rule, 151
R-axiom, 148
R-calculus, 146, 148, 221
R-complete, 164
R-configuration, 147
R-contraction, 147
R-cut rule-I, 150
R-inference tree, 152
R-proof tree, 152
R-provable, 152
R-reachable, 160
R-refutation, 147
R-sound, 164
R-termination, 153
R-transition, 148
R-transition sequence, 153
i-th version, 171
n-ary function, 35
n-ary relation, 35
call statement, 77
if statement, 77
while statement, 77
`GUINA`, 190
`OPEN`, 169, 174
`OPEN`$^+$, 182

Acceptable, 193
    contraction, 173
    relation, 193
Antecedent, 50
Assignment, 24
    statement, 77
Atomic
    formulas, 8
    proposition, 35
    statement, 77
Axiom, 50, 72, 217, 220
    system, 46
Axiomatization process for physics, 119

Backus normal form, 6
Basic
    instance, 190
    sentence, 191
    theorem of testing, 165
Bijection, 229
Bound, 12
    variable, 8
Business logic, 222

C-implementable, 216
Call
    by name, 89
    by value, 78, 89
Characteristic
    function, 230
    number, 243
Church-Turing thesis, 81, 216
Classical physics, 120
Closed term, 10
Closure, 72
Code of the procedure, 107
Commutativity, 183
Compactness, 61
Complement, 229

Complete, 49, 73
    inductive sequence, 200
    sequence of basic sentences, 191
    set of basic sentences, 191
Completeness, 63
Composite
    formulas, 8
    statement, 77
Computable function, 79
Computational result, 78
Conclusion, 15, 50
Configuration, 84
Consistent, 62
Constant
    sequence, 124
    symbols, 2
Constructive
    knowledge, 71
    proof, 223
Convergence, 183
Corollary, 218
Countable set, 230
Counterexample, 57

Decidable, 101
    relation, 79
    set, 101
Decreasing sequence, 122
Default
    conclusion, 131
    expansion, 132
    expansion sequence, 132
    operator, 131
    premise, 131
    value, 130
Digital methods, 225
Domain, 22, 229
    of code, 212

E-type version, 171
Element, 229
Elementary arithmetic, 74
    language, 5
Empty set, 229
Enumerate, 102
Equal, 229
Equality symbol, 5
Equivalent, 32
Execution sequence, 91
Execution state, 91

First class transition, 84
First-order language, 1
Fixed point, 99, 104
Forcing sequence, 136
Formal
    consequence, 54, 69, 220
    methods, 225
    proof, 53, 54
    refutation, 145
    theory, 71–73
Formula variables, 51
Free, 12
    variable, 10
Function symbols, 2

Gödel
    coding, 13
    number, 13, 99, 104
    set, 107
    term, 13, 15, 99
Gödel's
    consistency theorem, 111
    incompleteness theorem, 109
Galilean physics, 120
Generating function, 93, 238
Generator, 93, 238
Ground term, 10, 33, 191

Halting procedure, 79
Herbrand
    domain, 33
    model, 33, 36
    structure, 36
    term, 33
    universe, 33
Hintikka set, 33

I-type version, 197

Ideal, 184, 223
  model of refutation by facts, 144
  proscheme, 184
  refutation model, 144
  research methodology, 182
Image, 229
Implementation problem, 72
Implementational knowledge, 71
Inconsistent **R**-configuration, 147
Increasing sequence, 122
Independence, 183
Independent, 73
  theory, 73
Induction
  basis, 16
  hypothesis, 16
  rule, 188
Inductive
  consequence, 188, 194
  process, 197
  sequence, 197
  version, 197
Inference tree, 53
Initial
  conjecture, 197
  state, 87
  theory, 171
Injection, 229
Input parameter, 76
Instance, 190
  addition rule, 194
  of the inference rule, 52
Interpretation, 22
Intersection, 229
  of the set sequence, 231

Lemma, 218
Limit, 123
  of sequence, 123
Lindenbaum
  extension, 63
  procedure, 63
Logical
  connectives, 3
  consequence, 31, 45, 69
  formula, 8
  symbols, 2, 3
Loop body, 78
Loop invariant, 91
Lower limit, 123

Map, 229
Matrix number, 241
Maximal
  consistent set, 62
  contraction, 144, 145
Meta-language environment, 76
Microsoft Solution Framework, 119
ML-implementable, 216
Model, 24, 72, 210
  of **R**-refutation, 147
  of refutation by facts, 144, 147

N-type version, 120, 171, 197
Necessary antecedent, 140
  set, 140
Negative instance, 191
New
  axiom, 120, 143
  conjecture, 139, 143
Newtonian physics, 120
Non-acceptable, 193
Non-halting procedure, 79
Non-monotonic sequence, 122
Normal default rule, 131

Onto, 229
Operation, 127
Operational calculus, 86
Order, 100
Ordinal number, 13
Output
  parameter, 77
  sequence, 127
  version sequence, 176

Positive instance, 191
Post-condition, 88

Postulate, 217
Pre-condition, 88
Predicate symbols, 2
Prefix representation, 7
Preimage, 229
Premise, 15, 50, 220
Prerequisite, 131
Principal formula, 51
Principle, 217
    of environment, 213
    of excluded middle, 22, 214
    of logical connectives, 215
    of observability, 216
    of Occam's razor, 217
Printing statement, 77
Proof
    theory, 49
    tree, 54
Proper subset, 229
Proposition variable, 99
Proscheme, 119, 127
Provable, 46, 54, 152, 153

Quantifiers, 3

R-type version, 121, 171, 197
Rank, 18
Recursive functions, 80
Recursively enumerable set, 102
Refutation by facts, 139
Reliable, 184, 223
    proscheme, 184
Represent, 96
Representable, 95, 96
Representation, 81, 95
    problem, 72
Resolution procedure, 129
Resolvent, 128
    relation, 128
Revision
    consequence, 194
    rule, 194
Rule, 217

Satisfiability, 30
Satisfiable, 30
Scalability, 222
Scientific problem, 172
Second class transition, 85
Self-referential, 98
Semantic conclusion, 31
Sentence, 10
Sequence, 122
    number, 240
    of formal theories, 122
Sequent, 46, 50
Sequential statement, 77
Service, 222
Set, 229
    of natural number, 230
    of object symbols, 219
    of truth values, 26
Side formulas, 51
Sound, 49
Soundness, 57
Special theory of relativity, 121
Specificational knowledge, 71
State, 83
    variable, 87
Statement, 77
Structural
    induction, 16
Structural operational semantics, 86
Structure, 22, 210
Subset, 229
Substitution calculus, 42
Succedent, 50
Successor function, 74
Surjection, 229
Symbol string, 100
Symbols, 2
    about the domain knowledge, 2

T-condition, 134
T-generic Set, 135
T-generic set, 135
Tautology, 30
Term, 6
    domain, 33

Terminating state, 87
Theorem, 217
Theoretical framework of first-order languages, 209, 219, 223
Theory of elementary arithmetic, 74
Transition, 84
True propositions, 226

Union, 229
    of the set sequence, 231
Universal
    induction rule, 194
    inductive version, 197
    principles, 226
Unprovable, 54, 152
Upper limit, 123

Valid, 30, 31, 57
Validity, 30
Value, 77
Variable, 3
    symbols, 3
Variable set
    of the initial state, 87
    of the terminating state, 87
Version, 171
    sequence, 171