Lecture Notes in Artificial Intelligence 1867

Subseries of Lecture Notes in Computer Science
Edited by J. G. Carbonell and J. Siekmann

Lecture Notes in Computer Science
Edited by G.Goos, J. Hartmanis, and J. van Leeuwen

Berlin
Heidelberg
New York
Barcelona
Hong Kong
London
Milan
Paris
Singapore
Tokyo

Bernhard Ganter Guy W. Mineau (Eds.)

Conceptual Structures: Logical, Linguistic, and Computational Issues

8th International Conference
on Conceptual Structures, ICCS 2000
Darmstadt, Germany, August 14-18, 2000
Proceedings

Series Editors

Jaime G. Carbonell, Carnegie Mellon University, Pittsburgh, PA, USA
Jörg Siekmann, University of Saarland, Saarbrücken, Germany

Volume Editors

Bernhard Ganter
Technische Universität Dresden
Fakultät Mathematik und Naturwissenschaften, Institut für Algebra
01062 Dresden, Deutschland
E-mail: ganter@math.tu-dresden.de

Guy W. Mineau
Université Laval, Faculté des Sciences et Génie, Dépt. d'Informatique
Québec, Canada G1K 7P4
E-mail: mineau@ift.ulaval.ca

Cataloging-in-Publication Data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Conceptual structures : logic, linguistic, and computational issues ;
proceedings / 8th International Conference on Conceptual Structures,
ICCS 2000, Darmstadt, Germany, August 14 - 18, 2000. Bernhard Ganter ;
Guy W. Mineau (ed.). - Berlin ; Heidelberg ; New York ; Barcelona ;
Hong Kong ; London ; Milan ; Paris ; Singapore ; Tokyo : Springer, 2000
(Lecture notes in computer science ; Vol. 1867 : Lecture notes in
artificial intelligence)
ISBN 3-540-67859-X

CR Subject Classification (1998): I.2, G.2.2, F.4.1, F.2.1

ISBN 3-540-67859-X Springer-Verlag Berlin Heidelberg New York

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 any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer-Verlag. Violations are liable for prosecution under the German Copyright Law.

Springer-Verlag Berlin Heidelberg New York
a member of BertelsmannSpringer Science+Business Media GmbH
© Springer-Verlag Berlin Heidelberg 2000
Printed in Germany

Typesetting: Camera-ready by author, data conversion by Steingräber Satztechnik GmbH, Heidelberg
Printed on acid-free paper SPIN: 10722280 06/3142 5 4 3 2 1 0

Preface

Computer scientists create models of a perceived reality. Through AI techniques, these models aim at providing the basic support for emulating cognitive behavior such as reasoning and learning, which is one of the main goals of the AI research effort. Such computer models are formed through the interaction of various acquisition and inference mechanisms: perception, concept learning, conceptual clustering, hypothesis testing, probabilistic inference, etc., and are represented using different paradigms tightly linked to the processes that use them. Among these paradigms let us cite: biological models (neural nets, genetic programming), logic-based models (first-order logic, modal logic, rule-based systems), virtual reality models (object systems, agent systems), probabilistic models (Bayesian nets, fuzzy logic), linguistic models (conceptual dependency graphs, language-based representations), etc.

One of the strengths of the Conceptual Graph (CG) theory is its versatility in terms of the representation paradigms under which it falls. It can be viewed and therefore used, under different representation paradigms, which makes it a popular choice for a wealth of applications. Its full coupling with different cognitive processes lead to the opening of the field toward related research communities such as the Description Logic, Formal Concept Analysis, and Computational Linguistic communities. We now see more and more research results from one community enrich the other, laying the foundations of common philosophical grounds from which a successful synergy can emerge.

ICCS 2000 embodies this spirit of research collaboration. It presents a set of papers that we believe, by their exposure, will benefit the whole community. For instance, the technical program proposes tracks on Conceptual Ontologies, Language, Formal Concept Analysis, Computational Aspects of Conceptual Structures, and Formal Semantics, with some papers on pragmatism and human related aspects of computing. Never before was the program of ICCS formed by so heterogeneously rooted theories of knowledge representation and use. We hope that this swirl of ideas will benefit you as much as it already has benefited us while putting together this program.

June 2000

Guy W. Mineau
Bernhard Ganter
Program Chairs, ICCS 2000

Organization

The International Conference on Conceptual Structures (ICCS) is the annual conference and principal research forum in the theory and practice of conceptual structures. Previous ICCS conferences have been held at the Université Laval (Quebec City, 1993), the University of Maryland (1994), the University of California (Santa Cruz, 1995), Sydney (1996), the University of Washington (Seattle, 1997), Montpellier (1998), and at Virginia Tech (Blacksburg, 1999).

ICCS 2000, the 8th International Conference on Conceptual Structures, was organized by the Department of Mathematics, University of Technology, Darmstadt, Germany.

General Chair

Rudolf Wille	Technische Universität Darmstadt, Germany

Program Chairs

Bernhard Ganter	Technische Universität Dresden, Germany
Guy W. Mineau	Université Laval, Québec, Canada

Conference Chair

Gerd Stumme	Universität Karlsruhe, Germany

Editorial Board

Michel Chein (France)
Harry Delugach (USA)
Peter Eklund (Australia)
John Esch (USA)
Brian Garner (Australia)
Roger Hartley (USA)
Bernard Moulin (Canada)
Heather Pfeiffer (USA)
Uta Priss (USA)
John Sowa (USA)
Bill Tepfenhart (USA)

Ontology Track Chair: Dieter Fensel (The Netherlands)

Program Committee

Galia Angelova (Bulgaria)
Jean-François Baget (France)
Tru Cao (UK)
Dan Corbett (Australia)
Judy Dick (Canada)
Norman Foo (Australia)
David Genest (France)
Olivier Gerbé (Canada)
Robert Godin (Canada)
Adil Kabbaj (Morocco)
Mary Keeler (USA)
Adalbert Kerber (Germany)
Sergei Kuznetsov (Germany)
Pavel Kocura (UK)
Lotfi Lakhal (France)
Wilfried Lex (Germany)
Dickson Lukose (USA)
Graham Mann (Australia)
Philippe Martin (Australia)
Robert Meersman (Belgium)
Peter Øhrstrøm (Danmark)
Silke Pollandt (Germany)
Richard Raban (Australia)
Anne-Marie Rassinoux (Switzerland)
Myriam Ribière (USA)
Ulrike Sattler (Germany)
Finnegan Southey (Canada)
Gerd Stumme (Germany)
Thanwadee Thanitsukkarn (Thailand)
Kristina Toutanova (USA)
Petko Valtchev (France)
Michel Wermelinger (Portugal)

Further Reviewers

Robert W. Burch (USA)
Richard Cole (Australia)
Winfried Gödert (Germany)
Bernd Groh (Australia)
Michel Habib (France)
Ollivier Haemmerlé (France)
John E. Heaton (UK)
Fritz Lehmann (USA)
Wolfgang Lenski (Germany)
François Modave (Australia)
Ralf Molitor (Germany)
Susanne Prediger (Germany)
Daniel Rochowiak (USA)
Eric Salvat (France)
Gregor Snelting (Germany)
Thomas Tilley (Australia)
Stephan Tobies (Germany)
Rudolf Wille (Germany)

Table of Contents

Invited lectures are marked with an asterisk

Concepts and Language

The Role of Conceptual Structure in Human Evolution * 1
Keith Devlin

Concepts in Linguistics – Concepts in Natural Language * 13
Gisela Harras

Patterns, Schemata, and Types –
Author Support through Formalized Experience 27
Felix H. Gatzemeier

Conventions and Notations for Knowledge Representation and Retrieval .. 41
Philippe Martin

Conceptual Ontology

Ontology, Metadata, and Semiotics * 55
John F. Sowa

Pragmatically Yours, ... 82
Mary Keeler

Conceptual Modeling for Distributed Ontology Environments * 100
Deborah L. McGuinness

Discovery of Class Relations in Exception Structured Knowledge Bases ... 113
Hendra Suryanto, Paul Compton

Conceptual Graphs: Perspectives

CGs Applications: Where Are We 7 Years after the First ICCS ? * 127
Michel Chein, David Genest

The Engineering of a CG-Based System: Fundamental Issues * 140
Guy W. Mineau

Conceptual Graphs, Metamodeling, and Notation of Concepts 157
Olivier Gerbé, Guy W. Mineau, Rudolf K. Keller

Knowledge Representation and Reasonings
Based on Graph Homomorphism * 172
Marie-Laure Mugnier

User Modeling Using Conceptual Graphs for Intelligent Agents 193
James F. Baldwin, Trevor P. Martin, Aimilia Tzanavari

Towards a Unified Querying System of Both Structured and Semi-structured Imprecise Data Using Fuzzy View 207
Patrice Buche, Ollivier Haemmerlé

Formal Semantics of Conceptual Structures

The Extensional Semantics of the Conceptual Graph Formalism 221
Guy W. Mineau

Semantics of Attribute Relations in Conceptual Graphs 235
Pavel Kocura

Nested Concept Graphs and Triadic Power Context Families* 249
Susanne Prediger

Negations in Simple Concept Graphs 263
Frithjof Dau

Extending the CG Model by Simulations 277
Jean-François Baget

Contextual Logic and Formal Concept Analysis

Building and Structuring Description Logic Knowledge Bases Using Least Common Subsumers and Concept Analysis 292
Franz Baader, Ralf Molitor

On the Contextual Logic of Ordinal Data 306
Silke Pollandt, Rudolf Wille

Boolean Concept Logic 317
Rudolf Wille

Lattices of Triadic Concept Graphs 332
Bernd Groh, Rudolf Wille

Formalizing Hypotheses with Concepts 342
Bernhard Ganter, Sergei O. Kuznetsov

Generalized Formal Concept Analysis 357
Laurent Chaudron, Nicolas Maille

A Logical Generalization of Formal Concept Analysis 371
Sébastien Ferré, Olivier Ridoux

On the Treatment of Incomplete Knowledge in Formal Concept Analysis .. 385
Peter Burmeister, Richard Holzer

Conceptual Structures in Practice

Logic-Based Networks: Concept Graphs and Conceptual Structures * 399
Peter W. Eklund

Conceptual Knowledge Discovery and Data Analysis 421
Joachim Hereth, Gerd Stumme, Rudolf Wille, Uta Wille

CEM – A Conceptual Email Manager 438
Richard Cole, Gerd Stumme

A Contextual-Logic Extension of TOSCANA 453
Peter Eklund, Bernd Groh, Gerd Stumme, Rudolf Wille

A Conceptual Graph Model for W3C Resource Description Framework ... 468
Olivier Corby, Rose Dieng, Cédric Hébert

Computational Aspects of Conceptual Structures

Computing with Conceptual Structures * 483
Bernhard Ganter

Symmetry and the Computation of Conceptual Structures * 496
Robert Levinson

An Introduction to SNePS 3 * 510
Stuart C. Shapiro

Composition Norm Dynamics Calculation with Conceptual Graphs 525
Aldo de Moor

From PROLOG++ to PROLOG+CG:
A CG Object-Oriented Logic Programming Language 540
Adil Kabbaj, Martin Janta-Polczynski

A Cost-Bounded Algorithm to Control Events Generalization 555
Gaël de Chalendar, Brigitte Grau, Olivier Ferret

Author Index 569

The Role of Conceptual Structure in Human Evolution

Keith Devlin

Center for the Study of Language and Information, Stanford University,
Stanford, CA 94305, USA,
and School of Science, Saint Mary's College of California,
Moraga, CA 94575, USA
devlin@csli.stanford.edu

Abstract. We generally think of conceptual structures (and their mathematical representations: conceptual graphs) as a technical framework providing a sound basis for research and design work in knowledge representation and related areas of computer science. In this article, I suggest that conceptual structure predates computer science by some three-million years. In particular, I argue that the human brain (or rather, the brain of our hominid ancestors) acquired conceptual structure long before it acquired language, and that the acquisition of conceptual structure was the key cognitive development that led to the emergence of contemporary humans. Moreover, human language was the result of the addition of grammatical structure to an already developed conceptual structure.

1 Introduction

This article is somewhat different from the usual fayre in the conceptual structures community, in that I step back from the field and view conceptual structure as a feature of the human mind. I suggest that it is a feature that arose naturally during the course of human evolution, and indeed, played a pivotal role in the acquisition of language.

The thesis I present arose during the course of an investigation I was carrying out having a quite different purpose: to try to explain how the human brain acquired the ability for abstract, mathematical thought. The result of that project is described fully in Devlin (2000), and parts of this paper are adapted from that account.

I hope that members of the conceptual structures community will find what I have to say of interest. Whether the considerations I outline might lead to any advances within conceptual structures research I cannot say. That is not my purpose. Rather, having been asked by the organizers of the conference to address the broader philosophical issues surrounding conceptual structures, I felt there would be interest in the kinds of considerations I present here, where we take a view of conceptual structures very different from the one with which we are familiar.

Like any evolutionary account of the development of a cognitive capacity, my thesis is subject to debate in practically every particular. I believe my account to be coherent, and consistent with what evidence is available concerning human evolution. Personally, I also find it plausible. But there are plenty of competing theories of human language acquisition. A more comprehensive approach, that tries to consider

B. Ganter and G.W. Mineau (Eds.): ICCS 2000, LNAI 1867, pp. 1-12, 2000.
© Springer-Verlag Berlin Heidelberg 2000

and compare all possible interpretations of the evidence, would be far too long for this volume. Instead, I have adopted the approach — not uncommon in contemporary writing on evolution — of focusing on one particular thread.

2 The Puzzle of Human Brain Size

The human brain is one of nature's oddities. It is nine times larger than is normal for a mammal of our body size, and thirty times larger than the brain of a dinosaur of the same body size. Its actual size varies between 1,000 and 2,000 cubic centimeters, with the vast majority between 1,400 and 1,500 cubic centimeters. It reached its present size about half a million years ago, following a growth period that lasted about three-and-a-half million years. In terms of relative brain size, our closest rivals are the dolphins and porpoises, and after them the non-human primates — apes, chimpanzees, and monkeys. But all lag well behind us in the brain-to-body-weight ratio.

In addition to its highly unusual size, the human brain is an incredible energy hog. It makes up less than 2% of the body's overall mass, yet uses about 20% of its energy. The size and complexity of our brain also carries a cost in terms of childcare. Because much of the development of the human brain takes place between birth and two years of age, a young human is very dependent on the care of others, and has to be looked after fairly extensively for several years.

From an evolutionary perspective, the human brain is, then, somewhat of a puzzle. Natural selection is a hard taskmaster. A particular organ will only develop if it offers a definite survival advantage. Given the enormous, and quite atypical, cost of the human brain, both in terms of energy use and the need for extensive and prolonged parenting, the benefits conferred by the organ must be highly significant. What are those benefits? Or rather, what were they during the three-and-a-half million year timespan over which the human brain developed to its present size? Among the suggestions that have been proposed as possible answers to this question are: language, planning, tool use, logical thinking, color vision, accurate throwing, and body cooling. I shall argue that none of these really works, and that the most likely answer is the acquisition of conceptual structure. Part of my argument will be to show that the acquisition of conceptual structure is a necessary prerequisite for the acquisition of language and logical thinking ability.

One caution is perhaps in order at this point. The survival advantage that drives a particular evolutionary change is not necessarily the primary advantage the organism subsequently gains from the change. Natural selection often finds a novel use for a feature that evolved for some quite different purpose — a process evolutionary biologists refer to as *exaptation*. For example, the wings of birds developed from the cooling fins of certain dinosaurs. Such examples demonstrate why the suggestion that the brain grew in size because of its cooling effect is not quite as fanciful as might first appear. The brain is, after all, a highly effective organ for lowering body temperature, which is why wearing a hat on a cold day has such a major effect on how cold we feel. It is just possible — although for reasons that will emerge in this paper, highly unlikely — that the present use of the human brain for advanced thinking is merely an exaptation of a cooling device. After all, if birds could think, we could imagine them reasoning along the lines: „Wings are so perfectly designed for flying,

and so crucial to our lives as birds, how could they possibly have started out as cooling devices?“

3 The Advantages of a Large Brain

Many authorities have speculated about the main driving force behind hominid brain growth. See, for example, Bickerton (1990, 1995), Gibson & Ingold (1993), Hurford, Studdert-Kennedy & Knight (1998), and Dunbar (1997). My purpose here is not to survey that extensive and fiercely disputed field of scholarship. Instead, I shall simply indicate why I think some of the more commonly advanced suggestions are not sufficient. My purpose in doing so is to set the scene for what I believe is the most likely explanation.

In looking for the survival advantages that drove the enormous hominid brain growth for three-and-a-half million years, I am obviously not going to consider the advantages to the individual conferred by high intelligence in modern society — even if we assume that high intelligence requires a large brain.[1] Complex societies are a very recent development that came long after the human brain reached its present size and complexity.

Several authorities have suggested that the use of tools led to brain growth. (See Gibson & Ingold 1993.) But when you consider that by the end of the *Erectus* period, the brain size of the majority of *Erectus* members was already in the lower half of the modern human range, while their collection of tools was still relatively meager and crude, tool use can surely have been at most a contributing factor to brain growth. The real explosion in tool use came after the brain reached its present size.

Likewise, although language uses large portions of the human brain, the acquisition of language cannot have been the main driving force for brain growth. Language (i.e., full language, with syntax) was acquired at most 200,000 years ago, after the human brain had reached its present size. Still, given the huge advantage to the species conferred by language, it is tempting to argue that brain growth could have been driven by some kind of precursor to language. Fine. But what was that precursor (or those precursors)?

Color vision is another cognitive capacity that has been considered as a possible factor leading to large brains. It offers a number of survival advantages, including spotting colored fruit on bushes and trees, detecting potential predators, and seeing greater attractiveness in a potential mate. Moreover, the timing may be about right: color vision could have developed about three or four million years ago. However, other contemporary primate besides humans have color vision, and their brains are nothing like the size of ours, so it’s hard to see how the acquisition of color vision

[1] Within the size range of the human brain there is no obvious correlation between brain size and intelligence. Some very intelligent people have brains around 1,000 cubic centimeters, and others who show no signs of what we would call high intelligence possess 2,000 cubic centimeter giants. The Neanderthals, who died out about 35,000 years ago, are popularly supposed to have been of low intelligence, and quite likely were without language, yet they had slightly larger brains than ours, with most falling in the range 1,500 to 1,750 cubic centimeters.

could have played more than a comparatively minor role in driving the growth of the hominid brain.

William Calvin (1993) has suggested that accurate throwing could be the main force driving brain growth. Personally, I am not convinced that throwing requires a brain anything like the size of the human brain, and so I suspect that it played at most a small, contributory part in driving brain growth.

4 The Path to Thinking

Despite the many surprising examples of exaptation that exist, my own view is that brain growth was driven by precisely the thing you would expect when you look at the purposes to which we put our brains today. Namely, our ancestors developed larger brains primarily to enable them to think (and hence to act smarter, including formulating and following plans) and secondarily to communicate better. (I classify communication as secondary because communication of any idea presupposes a brain that can first generate that idea.) My task is to explain what „thinking" consisted of during the brain-growth period and to what end our ancestors put their improved communicative abilities.

In this connection, it is important not to be seduced by the popular present-day conception that the mind is a computing machine, which thinks by following a progression of discrete logical steps. As I argued in my book *Goodbye Descartes* (Devlin 1997), the assumption that all mental processes can be captured by logical rules is false. Moreover, it is precisely the falsity of that assumption that explains the failure of the many attempts to program digital computers to recognize scenes, handle natural language, and exhibit artificial intelligence. At the end of Devlin (1997), I proposed an alternative view of the human mind as a device for recognizing patterns — visual patterns, aural patterns, linguistic patterns, patterns of activities, patterns of behavior, logical patterns, and many others. Those patterns may be present in the world, or they may be imposed by the human mind as an integral part of its view of the world. (Some might argue that only patterns of the second kind are possible. My thesis is independent of that issue.)

Some of the patterns we recognize can be described using language (including specialized languages such as musical notation for describing certain aural patterns and mathematical notation for describing various mathematical patterns). Other patterns seem to defy all explicit description. For example, we generally have little difficulty recognizing a friend or relative we have not seen for several decades. In all probability, all details of the individual's face will have changed — indeed, our first reaction is to be surprised at how different they appear. And yet we recognize them nevertheless. Despite the many individual differences in their appearance, we know it is *the same face*. Recognition of a face seems to involve high level patterns that survive any number of individual changes.

In Devlin (1997) I also suggested that human expertise results not from learning to follow rules well but from acquiring the ability to recognize (and possibly create) a great many patterns and react accordingly. Rules are useful for acquiring a new skill. Expertise comes about when the brain has adapted to that new skill — when it has learned to recognize the crucial patterns — types of object, circumstances, or contextual situation — and make the appropriate response automatically and

effortlessly. At that stage, rules are no longer needed. Driving a car is an obvious example. Practically any human being can, and many do, become expert in driving a car. When we start learning to drive, we follow rules explicitly — and it shows! With a little practice, we become more fluent. We still follow rules, not explicitly but efficiently. Problems may arise when we meet conditions we are not used to: we sometimes revert to complete beginner status, having to think explicitly what to do. Eventually, after we have been driving for some time, we become so attuned to the various patterns and responses involved in driving that we can usually drive without giving the matter any attention, and our response when faced with an unexpected situation is instinctive and immediate. We no longer need the rules that guided the learning process.

Of course, human evolution can hardly have been guided by the development of the ability to drive a car. And yet we easily become highly skilled at driving. What evolution has provided us with is the ability to recognize new patterns and develop behavioral responses to those patterns. I shall build on this observation in order to develop an explanation of the growth of the hominid brain. But before I embark on that task, I need to introduce the linguists' notion of protolanguage.

5 Language and Protolanguage

Language is much more than a system of communication. Communication systems are generally mixtures of sounds, gestures, facial expressions, skin coloration, and bodily movements that enable creatures to inform fellow creatures of their current emotions, current needs, current desires, pending action, impending danger, and the location of food supplies. Many species of animals have communication systems, often of great sophistication.

According to the thesis I am advancing in this article, our pre-human, hominid ancestors developed a particularly expressive communication system that linguists refer to as *protolanguage*. A typical protolanguage utterance consists either of a single, referring word or a pair of words that express the fact that a particular object has a certain property. For example: „Danger!" „Leopard!" „Me hungry", „Mammoth dead", or „Give me!" Such a communication system is adequate for conveying basic predication information (i.e., information no more complex than „object X has property P") about any entity in the immediate environment or any object for which there is a referring word. No grammatical structure is required, other than the juxtaposition of two words. Word order is not important: „Hungry mother" and „mother hungry" mean the same, as do „Come mammoth" and „Mammoth come." The expressive power of protolanguage can be increased only by enlarging the vocabulary of referring words. The linguistic utterances of children less than two to three years of age are restricted to protolanguage. We adults often revert to protolanguage communication when we find ourselves in a foreign country where we do not speak the language, using a translation dictionary to determine the words we need but having no knowledge of how to combine those words to form grammatical sentences.

Unlike protolanguage, a *language* has a combinatorial structure (i.e., *syntax* or *grammar*) that allows the expression of more complicated ideas. The rules of syntax tell you how to put words together to form meaningful expressions that convey

complex ideas. Because of this structure, language may convey ideas about structured circumstances and events outside the immediate environment.

One obvious distinguishing feature of language is the presence of words that do not refer to anything in the physical world. Whereas the words of protolanguage all refer to something specific in the world — such as an object, a location, an action, or an emotional state — languages have a great many signs whose purpose is within language itself. English words such as *if, then, because, unless, and, or,* and *every* have a syntactic function — their role is to provide the means of combining simple ideas into more complicated expressions.

There is evidence that chimpanzees, monkeys, and apes can be taught protolanguage, although the instruction period generally takes several years and is limited to at most a few hundred „words." (The „words" are usually pictures or symbols drawn on cards or presented on a touch-sensitive computer screen.) Some of the most intriguing research on the use of protolanguage by apes is that of Sue Savage-Rumbaugh with her bonobo ape Kanzi. She has described the degree to which Kanzi has acquired protolanguage — using picture cards to communicate — in a number of books and articles, most recently Savage-Rumbaugh, Shanker, and Taylor (1998).

It is my belief that the three-and-a-half million year period of hominid brain growth was accompanied by — and, in a sense I'll make precise later, was driven by — a steady increase in protolanguage. During this growth period, our ancestors acquired more and more referring words, enabling them to (think and) communicate about ever more things and events. Then, between two-hundred thousand and seventy-five thousand years ago, our *Homo sapiens* ancestors acquired syntax. Exactly how the transition from protolanguage to language occurred is a matter of some considerable debate in the linguistic community. As I shall now show, there is good reason to believe that the change took place rapidly, rather than incrementally over a long period.

First, when you look closely at grammatical structure — in particular the universal grammar that is shared by all human languages[2] — it is clear that, with the exception of some specialized constructions such as the passive tense, which can be „added later," grammatical structure is an all-or-nothing thing. You simply can't have „half a grammar." (The seeming complexity of grammar and the ability to generate sentences of greater and greater length comes from the fact that a few basic rules can be *iterated* indefinitely.)

Second, if languages had evolved gradually, they would surely have evolved at different rates. In that case, we should see contemporary languages of different grammatical complexity. But we don't. Among all the world's languages, not one has a simpler structure intermediate between protolanguage and full language.

Third, the contemporary human brain goes from protolanguage to full, grammatical language in a single step. This is exactly how children acquire language. For the first two or three years of life, they acquire more and more words, which they use one or two at a time in the form of protolanguage. Then, suddenly, they progress to full language. A similarly rapid transformation occurs with the emergence of creoles — new languages built on the vocabularies from two or more other languages. For example, during the nineteenth century laborers from around the world migrated to Hawaii to work on the sugar plantations. The original immigrants communicated with

[2] See, for example, Pinker (1994) or Devlin (1997).

one another using a protolanguage which took its words from the different languages of the new immigrants. The resulting protolanguage was what linguists refer to as a *pidgin*. The immigrants' children, however, spontaneously spoke a grammatical language — a *creole* — that resulted from the addition of grammatical structure (syntax) to the vocabulary of the pidgin. (See Bickerton 1995.)

What occasioned the emergence of language in our ancestors — the step from protolanguage to syntactic language? It seems unlikely that the primary cause was an urge/desire for richer communication. Language arose when the only form of communication was face-to-face. Yet, practically all face-to-face, day-to-day communication about the here-and-now can be achieved using simple combinations of two or three content words, together with gestures. Thus, a sufficiently rich protolanguage is all you require. You don't need syntax — the grammatical structure of language.

But if language did not emerge solely or primarily to facilitate better communication, what additional survival benefits did the acquisition of syntax confer? I suggest that the benefits of full language came only when human social life had grown sufficiently complex to allow coordinated planning. For example, a well-developed protolanguage would enable one *Homo erectus* to say to another something like „dead mammoth" and point toward the north, or to say the *Erectus* equivalent of „fruit" while gesturing westward. And off they could trot in search of food. But protolanguage cannot convey a complex idea such as „There is a dead mammoth there [gestures to the north], but yesterday there were tigers in the vicinity, so if we get there and find they are still around, we should be prepared to move on over there [gestures to the west] where there is plenty of great fruit with little danger." For that kind of communication, you need the syntactic structure of language. But such communication is only necessary when the creatures doing the communicating can generate such complex thoughts.

Thus, I believe that language did not evolve *primarily* to facilitate greater communication. Rather, it arose, almost by chance, as a by-product of our ancestors acquiring the ability for an ever richer *understanding* of the world in which they found themselves — both the physical environment and their increasingly complex social world. The key development, I suggest, was „off-line thinking" — the capacity to reason in an abstract, „what if?" fashion. (Off-line thinking is to be distinguished from on-line thinking — sometime called stimulus–response activity — which is occasioned by sensory inputs from the environment and results in an appropriate responsive action. Neither the idea of off-line thinking nor the name is original with me. I discuss the concept at length in Devlin 2000, showing that the capacity for off-line thinking is in fact equivalent to having syntax.)

6 Brain Types

Here is my theory as to what drove the growth in the human brain. Our hominid ancestors were easily outclassed by other species in size, strength, and speed. They did not have the obvious advantages of lethally sharp claws or large, powerful jaws with long, razor-sharp teeth. Nor did they have thick skins or indeed any other anatomical defense against being ripped apart and eaten. They were masters of no particular habitat and survived only by remaining highly adaptable, moving from

place to place and from terrain to terrain in search of whatever food they could find — fruit, nuts, leaves, eggs, and occasionally meat from animals. Success in this lifestyle depended on a rich view of the world — the ability to recognize a large collection of patterns. The greater and the richer our ancestors' understanding of the world, the greater their chances of survival — by foreseeing, and then staying away from, danger and by figuring out where the next meal would come from and acting accordingly.

An obvious instance where it would have been advantageous to recognize a larger collection of patterns would be in tracking. An injured rhinoceros might still fend off an *Erectus* band who happened to come across it. But if the band could track the animal over several days, taking advantage of clues such as footprints, broken branches, and animal droppings, then they could follow the rhino at a safe distance until it was sufficiently weakened that they could move in for the kill. The greater the number of patterns the hunters could differentiate, the more clues they could take advantage of, and the better they would be able to track their prey. The human ability to track prey, often over great distance for several days, is not shared by apes or chimpanzees, so it could well have been one of the factors that drove early hominid brain growth. If our ancestors could also exchange information about the hunt using some form of utterances, they could coordinate their efforts, and so increase their chances of success.

In other words, our ancestors were highly adaptable, nomadic scavengers who lived by their wits. Survival depended on their being smarter than any other species, and most likely by being able to communicate much finer information than any other species. They developed an intense curiosity about their environment and about all other species, a desire — indeed, an instinct — to *understand* and *explain* (at least to themselves) whatever they saw. This survival feature, unique to humans (at least in extent), is something we recognize today, even in young babies.

Of course, a great deal of what we nowadays think of as „acting smart" involves trains of thought that presuppose syntax — what I referred to earlier as „off-line" thought. But syntax emerged at the end of the three-and-a-half million year period of brain growth? What exactly did acting smart amount to before then? Let's start out by looking at some simpler examples.

The sunflower constantly turns its head to face the sun. Since it requires the sun's rays to live and grow, its behavior could be said to be „acting smart" from „the sunflower's perspective." The sunflower's turning is somewhat analogous to our heading for the refrigerator when we are hungry. But most of us would be very reluctant to talk about a sunflower even having a perspective, let alone describing its movements as „acting smart."

There are simple water-based bacteria which will move toward nutrients in the water and will swim away from regions that contain chemicals poisonous to them. Although this is a fairly simple, low level, chemical response, it serves the survival needs of the bacterium well enough. Still, it is not what we generally mean by „acting smart."

Stomphia coccinea is a species of sea anemone whose ocean environment typically contains eleven types of starfish, only two of which prey on the anemone. If one of the nine non-threatening species of starfish happens to brush against a *Stomphia*, the anemone does not react. If one of the two predator starfish touches a *Stomphia*, however, the anemone recoils immediately. As with the bacteria, this is purely an automatic response to a particular chemical or collection of chemicals found in the

two predator species but not in the other nine. But the *Stomphia's* ability to distinguish between two categories of starfish — dangerous and not — is a cognitive ability that helps to keep it alive. Are we fully justified in maintaining that it is not „acting smart"?

Moving on to creatures with brains, when rooks see an animal coming too near their nest, they will pick up stones and drop them on the invader to drive the threat away. Is that „acting smart"? A leaf cutting ant that encounters an opening too narrow for the leaf it is carrying will maneuver the leaf until it can pass through the opening. Does that qualify as „acting smart"? How about the octopus which figured out how to unscrew a mason jar in order to get to food inside? Acting smart or not?

What all these examples have in common is that the organism or creature in question produces an action that is appropriate for a given situation. There is no need for a „thought process" mediating between a stimulus from the environment and the production of the response; it can be quite automatic. The key in terms of ensuring the organism's continued survival is that there be a link between a *type* of stimulus and a *type* of responsive action. To make this observation a bit more precise, we need to analyze the notion of a „type" (of object, environment, action, etc.).[3]

Type recognition is the key to life. The bacterium in water that moves toward dissolved nutrients and away from poisonous water can be said to „recognize" the types *nutritious* and *poisonous* („good" and „bad" if you like). The *Stomphia* anemone is able to distinguish between two kinds of starfish: *dangerous* and *non threatening*. (In human scientific terms, the *Stomphia's* single „dangerous" category contains two types of starfish, its single „non threatening" category contains nine. Eleven types for humans, two for *Stomphia*. What constitutes a type is very much in the eye of the beholder.) These two examples illustrate that it does not take a high intellect in order to „recognize" a type. Indeed, it does not even require a brain. All that is required is that the creature modify its behavior in a systematic way according to the type.

The cognitive requirement for the conscious acquisition of types is the ability to recognize similarities and differences: to realize that some things are similar — they are of the same *type* — and other pairs of things are different — they are not of the same *type*. But type-driven activity does not require cognition. For example, a simple thermostat can „recognize" the types *warm* and *cold* — by which I mean that it reacts appropriately to the two types. Of course, one important difference between a thermostat and an organism is that the types a living creature recognizes *matter* to it. Even for an organism as simple as a bacterium, the nutritious/poisonous distinction is a matter of life and death. But nothing matters to an inanimate type-recognizer.

In addition to the types *nutritious* and *poisonous*, which apply to ingestible substances, most animals recognize the types *friend* and *foe*, which apply to other animals. Adequate recognition of these types is again often a matter of life and death. In modern society, medical doctors are people who have learned to recognize a great many types dealing with conditions of the human body: the type of having a cold, the type of having influenza, the type of being HIV positive, the type of having a high cholesterol level, the type of being overweight, blood type, and so on. Indeed, much of the training doctors have to complete prior to being allowed to practice involves learning to recognize a great many such types, and learning to link each type of medical condition to a suitable *type* of treatment.

---

[3] A detailed analysis of types is provided in Devlin (1991).

One evolutionary path that many creatures have followed — including ourselves — is to increase the number of types they recognize and respond to. Such species „progress“ by successive generations responding better to various types than did their ancestors, and by differentiating *new* types (as the environment changes), which their ancestors did not differentiate.

Much of the time, such evolutionary developments comprise the acquisition (by a new generation) of new automatic stimulus–response links. These require no conscious effort or even cognitive activity of any kind. Humans, however, often do make a conscious effort to increase our repertoire of types. This can be at the level of a particular individual, such as a trainee doctor, or it can be at the species level, as in medical research. Much medical research amounts to increasing the number of types of body conditions that can be recognized, and expanding the collection of types of treatment that can be effectively applied. This may involve refining existing types of splitting them into subtypes. Or we may discover that what were once thought to be separate types are really subtypes of the same type — a new common thread is observed.

Clearly, distinguishing types is the very essence of life, or at least of staying alive. Type recognition is so important that in many animals, large parts of one particular organ have evolved to handle types — to recognize types and generate responses of appropriate types. That organ is the brain. Simple brains will recognize one or more types and in each case produce a bodily response of the appropriate type. Bigger brains have a greater repertoire of types, both types they can recognize and response types they can generate.

7 Conceptual Structure

I suggest that the massive growth of the hominid brain was driven largely by the acquisition of more and more types, giving our ancestors an increasingly rich view of the world (i.e., allowing them to differentiate more and more regularities in the world).

Some time between 75,000 and 200,000 years ago, after our ancestors' brain had reached its present size, it underwent a structural change (for what else could it have been?) that gave us syntax. I have already indicated that this change occurred rapidly. Since evolutionary changes are usually minor and incremental, what that implies, presumably, is that a certain threshold level of brain complexity was reached, so that a *structural* change that was in itself minor resulted in a major *functional* change in the brain's representational, analytic, and communicative capacities. Is there anything else, in addition to a rich collection of types (equivalently, a large protolanguage vocabulary) that would have been necessary for such a threshold change to take place?

I suggest there is: A complex *structure* that connects types together. Without that, I believe, the jump from protolanguage to language would have been far too great to have occurred in a short space of time and be the result of a minor structural change in the brain. Given a sufficiently rich repertoire of types and an associated structural network, however, it is possible to posit simple mechanisms whereby a single, small structural change in the brain could have given rise to syntax. One such, which I

describe in full in Devlin (2000), is that the brain exaptated the already existing neural circuits for generating syllables from phonemes.

Let me give you some examples of the kind of type-structure that is necessary for the emergence of syntax. We recognize the type of all people. We also recognize various *sub-types*: male humans and female humans, children and adults, people with college degrees farmers, the types of Americans, English, Germans, Mexicans, and so forth.

In addition to some types being subtypes (or refinements) of others, types form a complex, interrelated web. For instance, there are male, adult, Mexicans and female, American farmers with college degrees. We make implicit use of this web whenever we communicate with one another; it is part of the way we understand the world.

I believe that the development of such a type structure was a key step toward language. It may well be where our ancestral line parted company with the other primates in terms of cognitive development. Certainly, no evidence indicates that any nonhuman animals that have been taught protolanguage have such a structure. They may have a few types that apply to things in the world. But there is no evidence to suggest that they are aware of a framework of connections between those categories.

In contrast, I suggest that in order for *Homo erectus's* enormous brain to have provided sufficient survival advantage to outweigh the cost of supporting it, the acquisition of a rich collection of types must have been accompanied by the acquisition of a type *structure* — that is, their world view comprised not only types but connections between them.

Of course, types connected by structural relations in this manner is exactly what we mean by conceptual structure (and represent mathematically by conceptual graphs). In other words, I conclude that the principal driving force behind the three-and-a-half million year growth of the human brain, leading to the acquisition of language between 75,000 and 200,000 years ago, was the acquisition and development of conceptual structure in the pre-human hominid brain. Moreover, human language resulted from the addition of syntax on top of the conceptual structure that had already been acquired.

References and Further Reading

Bickerton, D. (1990). *Language and Species*, Chicago, IL: University of Chicago Press.

Bickerton, D. (1995). *Language and Human Behavior*, Seattle, WA: University of Washington Press.

Bickerton, D. (1998). Catastrophic Evolution: The Case for a Single Step from Protolanguage to Full Human Language, in Hurford, J. Studdert-Kennedy, M & Knight, C. (1998), pp.341–358.

Calvin, W. (1993). The unitary hypothesis: A common neural circuitry for novel manipulations, language, plan-ahead, and throwing?, in Gibson & Ingold (1993), pp.230–250.

Devlin, K. (1991). *Logic and Information*, Cambridge, UK: Cambridge University Press.

Devlin, K. (1997). *Goodbye Descartes: The End of Logic and the Search for a New Cosmology of the Mind*, New York, NY: John Wiley and Sons.

Devlin, K. (2000). *The Math Gene: How Mathematical Ability Evolved and Why Numbers Are Like Gossip*, New York, NY: Basic Books.

Dunbar, R. (1997). *Grooming, Gossip, and the Evolution of Language*, Cambridge, MA: Harvard University Press.

Gibson, K. R. & Ingold, T. (eds). (1993). *Tools, Language, and Cognition in Human Evolution*, Cambridge, UK: Cambridge University Press.

Hurford, J. Studdert-Kennedy, M & Knight, C. (eds) (1998). *Approaches to the Evolution of Language*, Cambridge, UK: Cambridge University Press.

Lieberman, P. (1992). *The Biology and Evolution of Language*, Cambridge, MA: Harvard University Press.

Pinker, S. (1994). *The Language Instinct: How the Mind Creates Language*, New York, NY: William Morrow.

Savage-Rumbaugh, S., Shankar, S., & Taylor, T. (1998). *Apes, Language, and the Human Mind*, New York, NY: Oxford University Press.

Tobias, P. V. (1971). *The Brain in Hominid Evolution*, New York, NY: Columbia University Press.

Concepts in Linguistics – Concepts in Natural Language

Gisela Harras

Institut für deutsche Sprache
Mannheim

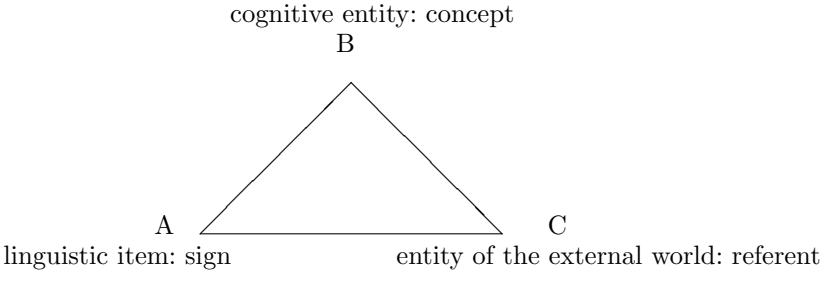
Abstract. This paper deals with different views of lexical semantics. The focus is on the relationship between lexical expressions and conceptual components. First the assumptions about lexicalization and decompositionality of concepts shared by the most semanticists are presented, followed by a discussion of the differences between two-level-semants and one-level-semantics. The final part is concentrated on the interpretation of conceptual components in situations of communication.

1 The Classical View

Following the classical scholastic view linguistic signs are related to two types of entitites:

(a) the type of cognitive entitites, concepts
(b) the type of entities of the external world

There is a direct relation between signs and concepts, between concepts and entities of the external world, and there is an indirect relation between signs and entities of the external world being mediated by concepts. These relations have been represented by the well known semiotic triangel (first in Ogden and Richards, 1953; cf. Lyons, 1977, 96.):



This representation is in accordance with the traditional analysis of signification as expressed in the famous scholastic maxim:

B. Ganter and G.W. Mineau (Eds.): ICCS 2000, LNAI 1867, pp. 13–26, 2000.
© Springer-Verlag Berlin Heidelberg 2000

"Voces significant res mediantibus conceptibus"
(Words signify things by means of mediating concepts)

As Lyons has pointed out there is considerable disagreement about the details of the triadic analysis of signification (cf. Lyons, 1977, 99):

- should A be defined as a physical or a mental entity?
- what is the psychological or ontological status of B?
- is C something that is referred to by uttering the sign?
  (If this would be so, how can signs while they are not used signify something?)
- is C the totality of things that might be referred to by uttering the sign?
- is C some typical or ideal representative of this class?

In modern semantics there are two ways of answering those questions:

(1) the post Saussurean view: A and B are both psychological (or mental) entitites. They constitute the linguistic sign als having two aspects: the aspect of the *signifiant*, *image accoustique*, i.e. the phonological form, and the aspect of the *signifié*, the concept. The meaning of a linguistic sign is then, following Saussure, composed of the intrinsic relation between *signifiant* and *signifié* and the meaning relations the sign holds to all the other signs of a given la nguage, the *valeur*.This Saussurean conception has been modified by modern semanticists in the following way: Concepts (which are not clearly defined by Saussure) are considered as abstract and collective entities in contrast to individual mental images, ideas or thoughts. They are relatively stable (in contrast to most psychological views) and highly structured. The principles of structuring concepts are part of the human cognitive endowment, they are innate. Linguistic expressions encode concepts as their semantic content cut out of the conceptual pool which is universal, i.e. independent of any existing language. Because of this twofold relationship of linguistic expressions and concepts as semantic content and universal conceptual structure this kind of semantics has also been called **two-level-semantics**. It has been worked out by Bierwisch, Lang, Wunderlich, Schwarz.
(2) A second issue in modern semantics is the view that linguistic forms are immediately related to concepts without any intermediate level of semantic content. It has been worked out by Jackendoff, Lakoff, Fauconnier, Langacker.

In spite of the difference between the two kinds of semantics there are some common assumptions concerning the relation between linguistic expressions and concepts:

First of all, it is a common assumption that there are linguistic expressions which don't encode any concept at all, as for instance pronouns, interjections, the single words of an idiom or a formula like *good bye* or *hello*.

A further common assumption is that some concepts have no corresponding word, and can be encoded only by a phrase. Speakers of French don't have a

word for the concept expressed by engl. *sibling* or german *Geschwister*, but may nonetheless have the concept of "sibling" characterized as child of same parents, and object of many beliefs and expectations, a concept which has *frère* and *soeur* as subcategories. We don't have a word for "wheeled vehicle" or "bad person". The question if l exical gaps are generally or in a certain language accidental or systematically distributed is an open question. There seems to be some evidence that lexical gaps are generally high-leveled in the hierarchy of concepts and constitute generic terms for a range of lower leveled concepts and words.(cf. Fellbaum 1996.)

The third common assumption concerns the overall phenomenon that words can - in an actual use - encode a whole range of concepts. Suppose Mary says to Peter:

(1) Open the bottle

In most situations, she would be understood as asking him to uncork or uncap the bottle depending on the properties of the referent of the direct object: thus opening a corked bottle means uncorking it, and so on. Uncorking a bottle may be the standard way of opening it, but another way is to saw off the bottom, and on some occasion, this might be what Mary was aking Peter to do. Or, suppose Mary says to Peter:

(2) Open the washing machine

In most situations, she will probably be asking him to open the lid of the machine. However, if Peter is a plumber, she might be asking him to unscrew the back; in other situations, she might be asking him to blow the maschine open, or whatever.

The general point is that expressions like *open* can be used to convey indefinitely many concepts. It is impossible for all of these to be listed in a lexicon. Nor can they be generated by taking only the linguistic context, particulary the direct object, into account. It seems reasonable to conclude that a word like *open* is often used to convey a concept that is encoded neither by the word itself nor by the verb phrase *open X*. (Cf. for similar examples: Searle 1980; Pinkal 1995; Pustejovsk y 1995; Sperber/Wilson 1998.) The common claim is that lexical meaning is flexible, but not structureless, and from this follows that lexical meanings can be assigned a description of its structure in terms of features or meaning postulates.

The claim of decompositionality of the conceptual content of words is far from being without problems: Johnson-Laird (1987) tested the human faculty of giving definitions of word meanings by non experts. He choose four levels of semantic complexity, and predicted that semantically complex verbs, such as *watch* and *lend*, would be easier to define than the semantically simplest verb, such as *see* and *own*. This prediction was confirmed: It was easy for the subjects to break down the meani ng of a complex verb into simpler components for which there are corresponding words, but it was hard for the subjects to find such components for a simple verb.

Some semanticists therefore assume that there is a set of semantic primitives that cannot be further analyzed, such as "cause", "bring about", "vision", "human", "thing", "place" and so on. They are the basic features for building up more complex word meanings. (Cf. already Katz/Fodor 1963; Wierzbicka 1972; Bierwisch 1992; Lang 1994.) But even if we admit that there is only a subset of word meanings which can be analyzed in terms of features there are some problems left: The problems concern the organizatio n of the features as well as their logical status. The features "animal" and "concrete object" are analytical implications of the meanings of *cat* and *car* respectively, they have the status of meaning postulates unter the condition that the words are used in a standard non-marked context. Meaning postulates may vary from language to language (cf. Schwarze, 1987.) The German preposition *auf* analytically implies "in contact with (x,y)" whereas the french preposition sur does not analytically imply "in contact with (x,y)", cf.:

(3) L'avion plane sur la ville
*Das Flugzeug schwebt auf der Stadt

Another group of word meanings can well be described by components, but they have another organization and logical status than meaning postulates. This is the case with words like *elephant, tiger, lemon, water* and so on, so called natural kind terms. Most speakers of English are able to say what an elephant is, they have seen an elephant in the zoo, or a picture of one, and they know something about the nature of the animal. Yet the term is a theoretical one (cf. Johnson-Laird 1987, 203.) It designate s a set of creatures within our categorization of animals. Our knowledge of such matters is far from complete. We don't know for certain what the essentials of elephanthood, tigerhood or cathood actually are. These words notoriously give rise to the problem of delimiting what should be said in a dictionary and what should be said in an encyclopedia, because it is doubtful whether there are any necessary and sufficient conditions for defining them. (cf. Putnam 1975; Lutzeier 1985; Schwarze 1987; Harras 1991.) If someone tells me that he saw an elephant in the zoo, then I will interpret her utterance to mean that she saw a large four-legged mammal with tusks and a trunk. But these characteristics are not essential, and they are not mere inductions, since to check them inductively presupposes some independent method for first identifying elephants; in fact they are part of our "theory" of elephants (cf. Johnson-Laird 1987; Putnam 1975.) which tells us that the stereo- or prototypical member of the class has each of these attributes. The lexical meaning of *elephant* must therefore be represented by a schema of the stereo- or prototypical animal, a mental model with a set of default values, that is, specific values for variables that can be assumed in the absence of information to the contrary. Default values have a special status concerning their contribution to the truth conditions of the sentence where the word occurs. Necessary components of a word's meaning support valid inferences, default values hold onl y in the case that nothing is asserted to the contrary. Stereotypes as the content of the meaning of natural kind words may vary from culture to culture, they are

not necessarily dependent on a certain language. (cf. Putnam 1975; Schwarze 1987.) The same is true of words for artefacts like *knife, hammer, plate, table* and so on.

After we have known the common assumptions of most of modern semantic theories let's now have a look at their differences. I will first discuss the so called two-level-semantics.

2 Two Conceptual Domains: The Two-Level-Semantics

Concepts appear within this theory in two domains:

- in the domain of semantic form as the conceptual content of a lexical expression
- in the domain of conceptual structure in terms of which the actual interpretation of a given linguistic expression is specified.

The domain of semantic form is related to the language-dependent representation of a conceptual structure, the conceptual structure is related to the universal representation of encyclopedic background knowledge, contextual information and situational conditions. The semantic form of a lexical expression constitutes its core meaning, that is, the context-free meaning as stored in long term memory. The domain of conceptual structure is needed for the interpretation of a given lexical expression in a certain context and situation. The focus of two-level-semantics is upon the representational aspect of meaning as well as on the dynamic procedural aspect of information processing. This kind of semantics is therefore claimed to be a part of cognitive science and the cognitive information processing system does not necessarily have to be a human being.

The distinction between semantic form and conceptual structure is mainly motivated by the overall phenomenon of the underdetermination of linguistic expressions. Well known examples are the following (cf. Bierwisch/Schreuder 1992; Schwarz 1992):

(1) John left the institute an hour ago
(2) John left the institute a year ago

Linguistically, we know that *John* is a proper name by means of which we may refer to male persons identified by this name. This is the information represented in the semantic form of *John*. Conceptually, we have a specific knowledge about each person named John we happen to know. This knowledge has nothing to do with the knowledge of English.

In (1) *the institute* most likely refers to a building and *leave* is interpreted as a change of place, while in (2) *the institute* refers to an institution and *leave* is interpreted as a change of profession. The different time intervals in (1) and (2) bring in different background knowledge not contained in any of the expressions in (1) and (2).

Another case of linguistic underdetermination is the following:

(3) The office is closed

If (3) is uttered by a servant of the office the office door may be open, while uttered in another situation the door may be locked. In all these cases the interpretation of a given utterance is first based on the semantic form of the expressions which activates the relevant background knowledge of conceptual structures. The semantic form is only a part of the final interpretation, or more precisely: the conceptual interpretation of a given semantic form would have to contain this semantic form as a proper substructure, or at least the weaker condition must hold: the semantic form has to be embeddable into the conceptual interpretation, with embedding to be conceived as the relation of a partial model to a more complete model the partial model is compatible with. (cf. Bierwisch/Schreuder 1992; Kamp/Reyle 1994.)

The strucure of a full lexical entry contains four kinds of lexical information (cf. Bierwisch/Lang 1987; Bierwisch/Schreuder 1993; Lang 1994):

- the phonetic form represented by phonological features
- the grammatical features, syntactic category, finiteness
- the semantic form represented by categorized variables x,y,P, formal components such as $\subset$, $\subseteq$, $\rightarrow$, $\leftrightarrow$ and material components such as CAUSE, BECOME, PLACE.
- the argument structure of verbs and some nouns.

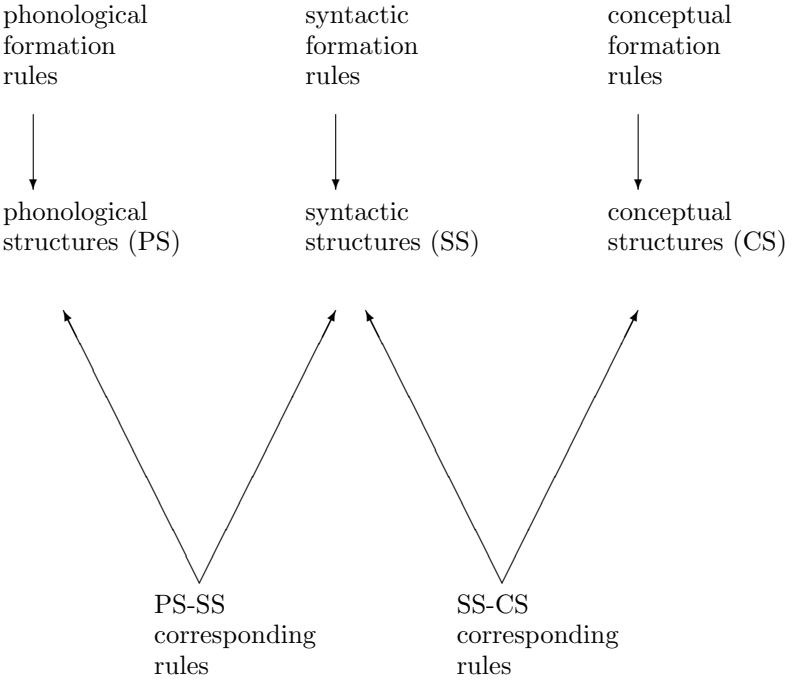
The natural domain of two-level-semantics are simple and complex verbs, dimension adjectives and prepositions (cf. Wunderlich, 1997). Their semantic form is much more structured than the semantic form of nouns, especially words for natural kinds or artefacts like *cat, camel, lemon* or *chair, knife* and *car*. The conceptual analysis of these words may be infinite and the object of all kinds of contingent knowledge. The semantic form of words for natural kinds is poor; it contains only the info rmation NATURAL KIND, ANIMAL; their conceptual structure may be extremely rich. The semantic form of artefacts contains the information of specific functions that allows for inferences about the relation between their referents and human activities. In accordance to this assumption the representations of the semantic form of *cat* and *camel* on the one side and *chair* and *knife* on the other are the same in all relevant aspects. The differences between cats and camels and between chairs a nd knifes are captured in the conceptual structure. This strategy implies a problematic division of labour, that is: linguistics is responsible for the formal reduction, while psychology, philosophy or whatever world science is responsible for the material instanciation. However, besides this very general problem, there are at least two serious problems left, one of which concerns rather methodological difficulties, whereas the other relates to more fundamental questions about information processing:

- the first problem concerns the distinction between assigning semantic and conceptual information to a given lexical expression. As Lang (1994, 28) has pointed out, the fact that the conceptual structure is inaccessible to direct observation may leed to a thoughtless overgeneralization of semantic properties of a given language. This may be true even if semanticists respect

the caveat principle of cross-linguistic and intermodal comparing. (cf. Lang 1994; Meyer 1992; Kriffka/Wenger 1990; Dölling 19 92.)
– the second problem concerns the question of how human beings are capable to convey and understand the relevant information of a given linguistic expression, if most of their utterances are semantically underdetermined. I will turn to this problem in the last part of my paper.

3 One Domain of Concepts: One-Level-Semantics (Jackendoff)

The fundamental assumption of Jackendoff (1983; 1990; 1997.) is that the grammatical structure of natural languages offers an important new source of evidence for (the theory of) cognition. The grammatical structure of a natural language is regarded as a triple, consisting of phonological structures, syntactic structures and conceptual structures. Phonological, syntactic and conceptual structures are determined by phonological, syntactic and conceptual formation rules. The three structures are connected by corresponding rules (cf. Jackendoff 1997, 39.):



Phonological and syntactic structures are modules of their own, whereas conceptual structures must be linked to all the other sensory modalities. In contrast to two-level-semantics Jackendoff (and others like Fauconnier 1985; Lakoff 1987;

Langacker 1988; Pustejovsky 1995.) does not postulate a level of semantic form as an interface between grammar and concepts: "...there is not a form of mental representation devoted to a strictly semantic level of word meanings, distinct from the level at which linguistic and nonlinguistic information are compatible. This means that if, as is often claimed, a distinction exists between dictionary and encyclopedia lexical information, it is not a distinction of level; these kinds of information are cut from the same cloth." (Jackendoff 1983, 110.)

This position has at least two consequences:

- if semantics takes into account that lexical meanings are highly context dependent then a one-level conception has to match all (possible) contexts or phrase structures to a given lexical item.
- There must be some rules on the level of the conceptual structure allowing for re-interpretations of a lexical expression in a certain context and situation.

The first claim is fulfilled in so far as the rules of corresponding syntactic and conceptual structures are applied to syntactic structures of different complexity: as V^0 (the simple verb), as V^1 (verb + direct object), V^2 (verb + direct object + indirect object), V^3 (verb + direct object + indirect object + prepositional object) and so on. So we get different (complex) lexical entries for *give, give something, give something to someone* or *give something to someone on some occasion.* (cf. Jackendoff 1990) The aim of Jackendoff (and other one-level-semanticists) obviously is not to build up a genuine lexicon of a given language, but rather to show the above mentioned quality of natural language as a source of evidence of human cognition.

The second claim is fulfilled by introducing preference rules for stereo- or prototype concepts such as BIRD, FRUIT or VEGETABLE. Preference rules are like to default values: They mark the best example in a subcategory of a higher category, so as for instance ROBIN as the best example of the category BIRD.

Preference rules are not only applied to nouns constituting words for natural kinds or artefacts, they are also applied to verbal concepts (cf. Jackendoff 1983, 150):

(1) I must have looked at that a dozen times, but I never saw it
(2) I must have seen that a dozen times, but I never noticed it

(1) and (2) raise a serious problem if we assume that *see* has a unified single meaning: The meaning of *see* in (1) is used to deny its meaning in (2). "x $sees_1$ y" means something like 'x's gaze goes to y'. In this sense of see_1 the direct object may alternate with prepositonal phrases:

(3) John saw into the kitchen
(4) John saw under the chair

(4) may mean that John's gaze terminated at a certain point under the chair or that his gaze passed under the chair to a point beyond. So see_1 is very similar to a verb of motion:

(5) John saw the flying saucer from his living room

In (1) is asserted that I never became aware of the object; so we get see_2 with the meaning 'y comes to x's visual awareness'. It is precisely this awareness that is not necessary for the assertion of *see* in (2). A sentence like

(6) John sees Mary

is not in the same way ambiguous as *We went to the bank* : The speaker does not have one reading or the other in mind. (6) appears to intend both that John's gaze went to Mary and that Mary enters John's awareness, but the presence of both 'x's gaze goes to y' and 'y comes to x's awareness' is not a necessary condition for the use of the verb *see*, they are stereotypical devices, captured by preference rules.

The one-level-semantics of Jackendoff and others obviously does not give rise to the problem of the distinction between what is a language-dependent concept and what is a language-independent concept, a problem two-level-semantics is very much concerned with. However, the other problem of how people are capable of conveying and understanding the relevant meaning in semantically underdetermined utterances is left. I will turn to this problem next.

4 Meaning, Understanding and Human Communication

Communication seems to be an extremely ambiguous expression in English. In all cases of its occurrence it has obviously something to do with human interpretation: we may interpret all kinds of images, signs, natural states and behaviour of persons as something conveying meaning. This can be illustrated by the following example (cf. Carston 1999): Imagine observing a scene in which a man lowers himself, head and arms first, down into a hole in the ground while another man holes onto his legs, swiveling his eyes leftwards in our direction and jerking his head quite violenty from left to right. Very few observers will represent this scene to themselves as I have described it and leave it at that; most of us will try to find some plausible beliefs, desires and/or intentions that we can attribute to these two men, some set of mental states which will explain their behaviour. We'll take the head movement of the second man to be, not some involuntary tic he developed upon seeing us, but rather a movement desig ned to make it evident to us that he wants our intention and has something to tell us. We might even infer what the intended message is, something like "I want you to help me" perhaps.

The second man's behaviour, his swiveling his eyes and jerking his head, counts for us as an act of communication under the following conditions:

- the man's behaviour is intentional, it is an action;
- the intention is directed towards us
- the intention contains a message, so that: the man intended us to recognize his intention to communicate to us that he wants some help.

These conditions of human communication have been worked out in detail by Grice (Grice 1989). Following him, an act of communication can be defined als follows: A speaker S communicates something to a hearer H, iff

(i) S intends H to react in a certain way r
(ii) S intends H to recognize (i)
(iii) S intends H to react in the way r by means of recognition of (i)

Communication is thus regarded as a means of influencing people by using signs in order to bring about a certain reaction of people presupposing that the recognition of the intention of bringing about the reaction will be a sufficient reason for people to react in the intended way. An important point is that influencing people counts only as a case of communication, if it is intended to be recognized. In a given situation an imperative "go downstairs" may not be very different from a kick. Both kinds of inf luencing may bring about the same reaction of an adressee, that is: to go downstairs, but only the imperative is a case of communication in the Gricean sense (cf. Keller 1995).

The Gricean conception of communication is not necessarily restricted to linguistic communication. What are then the special conditions of linguistic communication? First of all we would say that the content of the act of communication, the utterance, has to be understood in the intended way. Understanding the speaker's utterance is necessary for the hearer to react in the intended way. If someone says to me:

(1) Could you pass me the salt

I first have to understand that the speaker uttered a request in order to react in the intended way, that is: to pass him the salt.

The definition of Gricean communication seems to suggest that the hearer has the role of a pure recipient of speaker's intention. But, following Grice, communication demands not only intentions on the side of the speaker but also cooperation on both sides. Communication is a cooperative enterprise governed by a general cooperative principle and some more specific maxims. The cooperative principle is about the presupposed appropriateness of a given conversation, i.e. a case of linguistic communication: "Make your conversational contribution such as is required, at the stage at which it occurs, by the accepted purpose of direction of the talk exchange in which you are engaged." (Grice 1989, 26.) The specific maxims concern the quantity of information (be informative), the quality of information (be sincere), the relevance of information (be relevant) and the modality of information (be clear).

The general cooperative principle and the maxims of communication seem to be trivial on the first account. But Grice himself and legions of linguists consider this principle and the maxims to be a powerful instrument for explaining how people convey and understand relevant information. Let's take a very innocent example: Someone saying

(2) The kettle is black

thereby means that the outside of the iron kettle is covered with dark brown grease stains (example from Travis 1997). The addressee who can see the kettle would certainly not say that (2) is false, even if he has evidence that, taken (2) literally, it is actually false! In interpreting what the speaker intends him to

understand, namely that the kettle is now darker than it has been some time before, the hearer must rely on the following assumptions:

- S and I are in the same situation of communication
- S has communicated to me that p (the kettle is black)
- p is not true in the situation we are in
- S is respecting the general principle of cooperation
- S is respecting the maxim of quality, i.e. he does not want to tell me something false
- given the situation we are in and the fact that S is cooperative, S intends me to unterstand that q (the kettle is now darker than it has been some time before).

This is the rough mechanism of what Grice has called a conversational implicature in contrast to implications which are due to the semantics of the expressions in a given utterance. It is quite clear that implicatures rely on different kinds of information:

- the information provided by the content of the speaker's utterance, i.e. what is linguistically said;
- the information provided by the special situation speaker and hearer are in;
- the information provided by the situation type of communicating.

These kinds of information have to be completed by a fourth one for cases in which the state asserted by the speaker's utterance is not - as in our example - a part of the situation speaker and hearer are in. This kind of information is provided by all kinds of background knowledge about how things are or have to be. Finally - the most crucial condition - the knowledge constituting all these types of information - or at least a part of it - has to be available for both speaker and hearer, i.e. it has to be common knowledge. Most of the traditional definitions have to struggle with an infinite regress by explaining common knowledge in terms of individual knowledge, like:

A knows X
B knows X
A knows that B knows X
B knows that A knows X
A knows that B knows that A knows X
B knows that A knows that B knows X
A knows that B knows that A knows that B knows X
B knows that A knows that B knows that A knows X
and so on: ad infinitum

It is quite obvious that, when people make use of common knowledge, they do not pursue an infinite regress of this kind. The analysis in terms of invidual knowledge is not at all appropriate to give a notion of real common knowledge.

Devlin (1997) has proposed an explanation in terms of situation theory: The act of two persons A and B having common knowledge of a fact, event or circumstance X constitutes a situation s. The situation s is a common knowledge situation for A and B to have common knowledge of X, iff:

(i) s is a situation supporting the individual knowledge of A and B:
s | ≪A knows X≫ ∧ ≪B knows X≫
(ii) both A and B know that s, the situation they are in, is one of common knowledge:
s | ≪A knows X≫ ∧ ≪B knows X≫ ∧ ≪A knows p≫ ∧ ≪B knows p≫

where p is the very proposition about A's and B's knowing p (cf. Devlin 1997, 255.). The analysis is circular and selfreferential. Its advantage consists in the principal garantee that everything in the common knowledge situation available for A is to the same extent available for B and vice versa. Consequently, with regard to the Gricean framework, the speaker ist not more privileged in meaning something than his addressee is in understanding what the speaker meant.

My purpose of this part of my paper actually is to give an account of how underdetermined utterances or parts of them can be interpreted by a hearer in accordance to the speaker's meaning. Till now, we have only considered the case of the black kettle, a case where the utterance had to be re-interpreted. What's about the *open-, office-* or *institute*-cases mentioned in part 1 and 2 of my paper? In these cases the hearer has to determine what concept of all possible concepts is the one meant by th e speaker. Does he really have to work out the speaker's meaning by a Gricean implicature, as it seems plausible for the black kettle case (as well as for other cases like metaphorical uses, rhetorical questions, tautologies and irony)? Sure, if we assume that we have stored all the possible concepts for *open* in our mental lexicon and that we have clear devices for accessing them, the Gricean mechanism would be unnecessary. But this is not the case: the possible concepts for *open* (and legions of other words like *cut, write, fly, eat, begin, end* or *compute*) are indefinite, and we don't have singular situations in mind as pointers to linguistic expressions: what we have stored in long term memory are situation types related to types of words, if any.

What special kind of implicature would be efficient to determine the conceptual content of *open* in a given context and situation? A speaker utters

(3) The plumber opened the washing machine yesterday, but he couldn't find any defect

meaning that the plumber unscrewed the machine. Are the conceptual components of the utterance sufficient for the hearer to understand that the plumber unscrewed the machine? Surely not, the plumber could just have opened the lid of the machine and looked into it or he could have done both, opened the lid and unscrewed the machine. So, the hearer has the choice between 'unscrewing the machine", 'opening the lid' and 'opening the lid and unscrewing the machine'. Is it our knowledge about the world that determines meaning in context? If so, what kind of re al, normal or expected world should this be? I think there

is no way out of relying only on linguistic context and individual background knowledge. We first have to account for the hearer's presumption of the speaker's cooperative acting, especially his being relevant. Assuming that the speaker intends to communicate relevant information, the hearer's background knowledge presupposed by him to be the speaker's and the hearer's common knowledge is activated, and this allows him to recognize the speaker's int ended meaning, that the plumber has unscrewed the washing machine.

Though this is a very weak implicature, it clearly illustrates that even in rather unspectacular cases of interpretation, pragmatic aspects related to acting, play a fundamental role. The lexicon of a language, conceptual knowledge and communicative acting are inseperately interrelated within human cognitive systems unlike artificial ones.

References

1. Bach, K. (1999): The Myth of Conventional Implicature. In: Linguistics and Philsophy 22, 327-366.
2. Bach, K. (1999): The Semantics-Pragmatics Distinction. What It is and Why It Matters. In: Turner, K. (ed.): The Semantics/Pragmatics Interface from Different Points of View. Oxford/Amsterdam etc.: Elsevier, 65-84.
3. Berg, J. (1991): The relevant relevance. In: Journal of Pragmatics 16, 411-425.
4. Bierwisch, M. & E. Lang (1989): Dimensional Adjectives: Grammatical structure and conceptual interpretation. Heidelberg: Springer.
5. Bierwisch, M. & R. Schreuder (1992): From concepts to lexical items. In: Cognition 42, 23-60.
6. Carston, R. (1999): The Semantics/Pragmatics Distinction: A View from Relevance Theory. In: Turner, K. (ed): The Semantics/Pragmatics Interface from Different Points of View. Oxford/Amsterdam etc.: Elsevier, 85-126.
7. Devlin, K. (1997): Goodbye, Descartes. The End of Logic and the Search for a New Cosmology of the Mind. New York etc.: Wiley.
8. Dölling, J.(1992): Flexible Interpretation durch Sortenverschiebung. In: Zimmermann, I. & A. Strigin (ed.): Fügungspotenzen. Berlin: Akademie-Verlag, 23-62.
9. Fauconnier, G. (1985): Mental Spaces: Aspects of Meaning Constructions in Natural Language. Cambridge/Mass.: MIT Press.
10. Fellbaum, Ch. (1996): WordNet: Ein semantisches Netz als Bedeutungstheorie. In: Grabowski, J./Harras, G. & Th. Herrmann (eds.): Bedeutung - Konzepte - Bedeutungskonzepte. Opladen: Westdeutscher Verlag, 211-230.
11. Grice, H.P. (1989): Studies in the Way of Words.Cambridge/Mass.: Harvard University Press.
12. Harras, G. (1991): Zugänge zu Wortbedeutungen. In: Harras, G./Haß, U. & G. Strauß (eds.): Wortbedeutungen und ihre Darstellung im Wörterbuch. Berlin/New York: de Gruyter, 3-96.
13. Jackendoff, R (1997): The Architecture of the Language Faculty. Cambridge/Mass.: MIT Press.
14. Jackendoff, R. (1983): Semantic and Cognition. Cambridge/Mass.: MIT Press
15. Jackendoff, R. (1990): Semantic Structures. Cambridge/Mass.: MIT Press.
16. Johnson-Laird, P.N: (1987): The mental representation of the meaning of words. In: Cognition 25, 189-211.

17. Kamp, H. & M. Reyle (1993): From Discourse to Logic. Dordrecht: Kluwer.
18. Katz, J. & J. Fodor (1963): The Structure of a Semantic Theory. In: Language 39, 170-210.
19. Keller, R. (1995): Zeichentheorie. Tübingen/Basel: Francke UTB.
20. Krifka, M. & K. Wenger (1990): Gradierung und Dimension. Eine Diskussion von "Grammatische und konzeptuelle Aspekte von Dimensionsadjektiven". In: Linguistische Berichte 130, 478-504.
21. Lakoff, G. (1987): Women, Fire and Dangerous Things. What Categories Reveal about the Mind. Chicago: University of Chicago Press.
22. Lang, E. (1994): Semantische vs. konzeptuelle Struktur. Unterscheidung und Überschneidung. In: Schwarz, M. (ed): Kognitive Semantik/Cognitive Semantics. Tübingen: Narr, 25-40.
23. Langacker, R. (1988): A View of Linguistic Semantics. In: Rudzka-Ostyn, B. (ed): Topics in Cognitive Linguistics. Amsterdam: Benjamins, 49-90.
24. Lutzeier, P. (1985): Linguistische Semantik. Stuttgart: Sammlung Metzler.
25. Lyons, J. (1977): Semantics. Vol 1. Cambridge: Cambridge University Press.
26. Meyer, R. (1994): Probleme von Zwei-Ebenen-Semantiken. In: Kognitionswissenschaft 4, 32-46.
27. Pinkal, M. (1995): Logic and Lexcon. Dordrecht: Kluwer.
28. Pustejovsky, J. (1995): The Generative Lexikon. Cambridge/Mass.: MIT Press.
29. Putnam, H. (1975): The Meaning of Meaning. In: Gunderson, K. (ed): Language, Mind, and Knowledge. Minneapolis: University of Minnesota Press, 131-193.
30. Schwarz, M. (1992): Kognitive Semantiktheorie und neuropsychologische Realität. Repräsentationale und prozedurale Aspekte der semantischen Kompetenz. Tübingen: Niemeyer.
31. Schwarze, Ch. (1987): Text Understanding and Lexical Knowledge. In: Verschueren, J. & M. Bertucelli-Papi (eds.): The Pragmatic Perspective. Amsterdam: Benjamins, 587-612.
32. Searle, J. (1980): Literal Meaning. In: Searle, J.: Expression and Meaning. Cambridge: Cambridge University Press, 117-136.
33. Sperber, D. & D. Wilson (1995): Relevance. Communication and Cognition. Second edition. London: Blackwell.
34. Sperber, D. & D. Wilson (1998): The Mapping between the mental and the public Lexicon. In: Carruthers, P. & J. Boucher (eds.): Thought and Language. Cambrdge: Combridge University Press, 184-200.
35. Travis, C. (1997): Pragmatics. A Companion to the Philosophy of Language. In: Hale, B. & C. Wright (eds.): Pragmatics. Oxford: Blackwell, 87-107.
36. Wierzbicka, A. (1972): Semantic Primitives. Frankfurt/M.: Athenäum.
37. Wunderlich, D. (1997): Cause and the Structure of Verbs. In: Linguistic Inquiery 28, 27-68.
38. Ziv, Y. (1988): On the Rationality of 'Relevance' and the Relevance of 'Rationality'. In: Journal of Pragmatics 12, 535-545.

Patterns, Schemata, and Types — Author Support through Formalized Experience

Felix H. Gatzemeier*

Lehrstuhl für Informatik III, RWTH Aachen
fxg@i3.informatik.rwth-aachen.de

Abstract. Conceptual authoring support provides tools to help authors construct and organize their document on the conceptual level. As computer-based tools are purely formal entities, they cannot handle natural language itself. Instead, they provide the author with directions and examples that (if adopted) remain linked to the text. This paper discusses several levels of such directions: A *Pattern* describes a solution for a common problem, here a combination of audience and topic. It may point to several *Schemata*, which may be expanded in the document structure graph, leaving the author with more specific graph structures to expand and text gaps to fill in. A *Type Definition* is finally a restriction on the possible document structures the author is allowed to build. Several examples of such patterns, schemata and types are presented.
These levels of support are being implemented in an authoring support environment called CHASID. It extends conventional authoring applications, currently ToolBook. The graph transformation aspects are implemented as an executable PROGRES specification.

1 What Conceptual Authoring Support Is Supposed to Be Good for

Most readers of this text will have been confronted with the task of writing larger complex texts. A Master's Thesis, as a once-in-a-lifetime task, is accepted as consuming a lot of time. However, when articles, proposals, reports or even books must be written on tight schedules, more efficient techniques have to be adopted. The reader, on the other hand, has come to expect more from the structure of a text. It is annoying to page through an electronic PostScript or even PDF document to trace cross-references. Ideally, the document should not only have *one* linear reading path, but provide just the necessary information needed to satisfy the reader's interests.

These observations are not fundamentally new. However, solutions to these problems are rare. Writing courses help in formulating ideas and in maintaining

* This work has been funded by the Deutsche Forschungsgemeinschaft (DFG) in its "Schwerpunktprogramm V3D2" (Verteilte Verarbeitung und Vermittlung digitaler Dokumente, Distributed Processing and Exchange of Digital Documents), http://www.cg.cs.tu-bs.de/dfgspp.VVVDD.

B. Ganter and G.W. Mineau (Eds.): ICCS 2000, LNAI 1867, pp. 27–40, 2000.
© Springer-Verlag Berlin Heidelberg 2000

an overview of the text as a whole. There is also advice on reading, addressing the extraction of the crucial context (1), or simply increasing the speed of the sentence recognition process. As in other application areas, usage of automated help is a viable alternative to education: not everybody who needs the skills has the time to attain them. A less fatalistic argument is that the computer is better at avoiding the stupid mistakes humans tend to make when having too much on their mind and too little time.

Computers have proven useful in capturing and typesetting text. However, the support currently available in practice and research remains very basic. Word processors and text editors allow manipulation of characters, maintaining a unified layout and up-to-date cross-references. This does not address the *horror vacui* writers traditionally face: the empty sheet has been replaced by an empty screen. To alleviate this, we propose a system of patterns that map problems to solution strategies and schemata that provide a structure of topics to be addressed and their relationships.

The structure of this article follows the PROBLEM/SOLUTION PRESENTATION SCHEMA as shown in Figure 7. The "Problems" section (2) looks closer at the writing and reading problems and identifies the ones most interesting here. The "Solution" is spread across three sections: the first (Section 3) describes the basis of the solution, namely authoring using conceptual graph structures. Concrete functionality is presented in Section 4, with the implementation details deferred to Section 5. The concepts presented here are related to other work in the obligatory "Related Work" section (6). Finally, conclusions and plans (Section 7) recapitulate the main thesis and identify the problems we intend to address next. Throughout the discussion, this paper itself is used as a running example.

2 The Problems: Writing and Reading

The author has to solve several problems before writing can begin: Initially, there must be a *topic* chosen. The author must *understand* that topic sufficiently, and she must have *references* or *evidence* to support the claims made in the context of that topic. Even though these problems vary with the kind of document to be written, they constitute the basis on which authoring can take place (14). To a certain degree, these problems can still be solved iteratively during writing. But, as a prerequisite for writing, they are not addressed here.

When setting out to write, some external requirements are usually also defined. The size of the document may or may not be restricted, and the audience (in terms of knowledge and interests) should also be discernible. The **author's basic problem** is then to *convey a topic in a given amount of space, building on some required knowledge and serving some interest.*

The author has to *select the points that must be made* from everything that can be said about the topic. She must *devise a structure of the document* and *fill the structure with text* and other media.

After a certain amount of document has been written, the *editing loop* usually begins. The author reads and evaluates her text; adding, deleting or moving por-

tions. In this process, she must *ensure consistency and reasonable redundancy-freedom*. This loop may also include *internal or external review*.

Finally, the document has to conform to formal quality standards of *orthography, grammar, typography, layout, and reproduction*. If the document is to be published in some way, the editor and publisher invests time to *technically unify documents*, especially submissions to a conference or books of a series. The publisher extracts bibliographical information for Library of Congress archiving (or other national equivalences). Finally, the publication is prepared for *mass printing and distribution*.

When the document arrives at the reader, he is usually *interested only in parts* of it. These parts may be the central claims, some hard data, instructions for certain procedures or solutions to more-or-less concrete problems. So, the **reader's basic problem** is to *find the relevant sections of the text quickly and grasp their meaning*. This process may iterate as further reading turns out to be necessary.

As the author could not anticipate or address the specific reading interest, the reader has to make do with tables of contents, indices and headings. If the reader seeks this author's usage of some some terms or methods, however, this guidance is of little use. Since construction of indices is costly and uses printing space, articles usually come without any and books only with a selected few.

Our work concentrates on supporting the author so that conceptual information that is useful for publishers and readers is gathered as a side-effect. In this paper, we address the authoring problems of selecting the points to be made and structuring the text.

3 The Basis: Authoring Knowledge in Conceptual Structures

The proposed solution starts by capturing authoring knowledge in patterns, schemata and types that can be used in high-level (concept-level) authoring tools. This entails adding the conceptual structure explicitly to the document.

The enhanced document contains the **presentation** (the core conventional document), the **presentation structure** (a description of its immediate structure, extending the expressiveness of the conventional authoring application) and the **content model** (a model of the concepts described in the document and their relationships). Initially, there is an empty presentation, the presentation structure consists of one generic top-level structure concept [Presentation] and the content model of one generic concept [Content].

3.1 Patterns Are Top-Level Guidance

Patterns have been characterized in (2): "Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution

a million times over, without ever doing it the same way twice."[1] Apart from the descriptions of problem and solution, a pattern consists of a unique name and a discussion of consequences of its use.

In our context, we use patterns to help the author reach clarity on the audience and topic. By browsing through a catalog of patterns, she is invited to consider alternatives for creating the document, based on her understanding of the audience and the topic. These are the keys for matching a problem, where the solution is the value mapped to the keys. Once the tool comes with a larger body of patterns, they may be categorized so the author can consult the catalog in a dialog-like manner.

As there is surprisingly little advice on this level in the literature, the patterns presented here are tentative proposals. There are also hardly any usage examples, as required by (5). This is a violation of the principle that patterns are supposed to express *proven* solutions, for which the only remedy is further study.

Goal Approximation (Pattern 1) is the ideal research community setting. There is a combined effort of research groups communicating with publications to achieve a common goal. As a consequence, the goal can just be referred to and does not have to be discussed in detail. On the other hand, requirements on the solution itself are higher. **Relationship Contribution** (Pattern 2), tries to initiate a dialog between research fields. This is a consequence of the atomization of research activity based on presumptions determining potential tools and methodologies. Papers of this kind run a risk of rejection on the grounds of being not related and must therefore pay special attention to making the problem interesting.

Pattern 1. ***Goal Approximation***

Problem *Describe advances towards reaching a goal accepted by the audience.*

Solution *In a* Problem-Solution *schema, state the goal briefly, referring to the literature for further discussion. Focus on the open question you are about to answer (*Open-Problem *schema). Describe your findings (empirical data, technical implementation, proof) in appropriate patterns (*Data-Solution, Technical-Solution, Proof-Solution*). Connect these findings to the open problem, restating it appropriately (*Partial-Answer *or* Full-Answer*). If applicable, discuss other consequences of your findings, such as ruling out other overall answer attempts.*

Discussion *The aim of the publication is to be a noticeable step forward. As a prerequisite, the result must directly link to a question defined in the literature in two ways: it must provide [part of] an answer to the question, and the question must still be open in that respect. Once this is established, there is no further need to discuss the problem. A reformulation of the question may be acceptable, but must not alter its foundation. If the problem is sufficiently open, giving a speculative answer that may, through discussion, lead to a narrowing of the question, can be accepted.*

[1] p. x, cited after (5).

Pattern 2. ***Relationship Contribution***

Problem *Describe a development that is of interest to the audience, but not on their mind.*

Solution *In a general* Problem-Solution *schema, propose your solution as one for a problem immediately appealing to the audience (*Known-Problem *or weaker,* Topical-Problem*). Emphasize the aspects of the discussion where the relationship surfaces.* Method-Solution *is appropriate for the description of the solution, as it will probably emphasize higher-level similarities. Differences may still be handled in an* Technical-Solution *schema. Use the terminology of the audience primarily, citing their classical works. Cite your own classical works for further reading.*

Discussion *The critical part is to make the contribution interesting enough for the audience to accept the different roots. One source of interest are works from the audience that state similar problems. As a last resort topoi may be employed.*[2] *Put an emphasis on the application area and its interesting problems, not on your understanding of the target audience's roots. As the audience probably does not speak the same language, their relevant terminology from classical works or textbooks should be used.*

Another pattern is hinted at here, to illustrate the use of the paradigm to non-submission documents. The **Status Report** (Pattern 3) may be customized for various authorities and have a tight interplay with schemata and type definitions.

Pattern 3. ***Status Report***

Problem *Report to an authority the progress and status of a project.*

Solution *For short-term reports, omit the description of goals and previously submitted reports. Factually describe the events in the report scope, relating them to goals (*Partial-Answer*).*

Discussion ...

This paper adopts the **Relationship Contribution** pattern. As can be seen in the implementation section (5), its background lies in somewhat different implementation technologies, but its content can well be expressed with conceptual structures terminology on the basis of the common paradigm of graphs (rather than trees or strings).

[2] In Aristotelean rhetorics, a topos is a sentence commonly believed, regardless of its truthfulness. In the Topica, Aristotle gives rules for such topoi, for example: "Also, the same things are more worthy of choice when pleasure is added than when it is absent, and when accompanied by freedom from pain than when attended by pain." ((**?** , III, 2, a23ff/p. 393)).

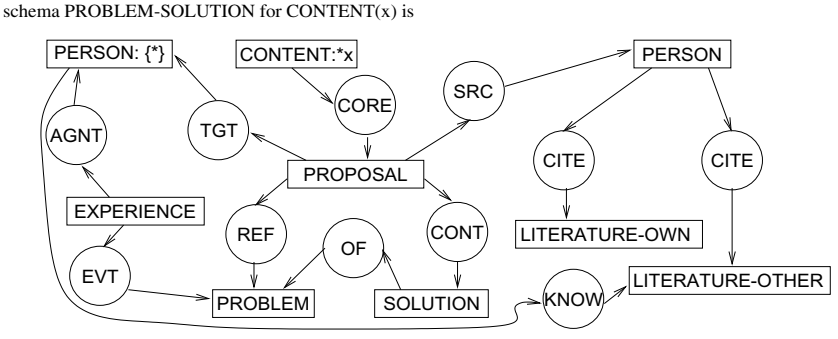


Fig. 1. Schema Problem-Solution

The Problem-Solution proposal references a problem, which must be of relevance to the audience (the target of the proposal), that is: they must have experienced the event as a problem. The author proposes a solution that solves the problem. Own literature may be cited to out-source detailed discussions or to add weight to the solution. Other literature is used to ground the problem and parts of the solution.

3.2 Schemata Provide a Guide to Requirements and Structure

Guided by the pattern, the author expands the starting concept [Content] with some schemata to build the model of the content to be described in the document. Concepts in the model may be related to concepts in the structure and finally to parts of the presentation.

To support the author, we extend the definition of a **schema** given in (12), Definition 4.1.1:

> A *schematic cluster* for a type t is a set of monadic abstractions $\{\lambda a_1 u_1, \ldots \lambda a_n u_n\}$, where each formal parameter a_i is of type t. Each abstraction $\lambda a_i u_i$ in the set is called a *schema* for the type t.

An abstraction, consisting of a conceptual graph in the body and the parameters, provides little guidance on its use. So, our schema consists of not only the abstraction, but also of documentation describing its usage and a name. The documentation may elaborate on the specific role a node plays and emphasize key components.

Content Schemata Help Organizing the Content Model. We begin the discussion of sample schemata with an overall Problem-Solution Schema. Schema 1 shows the base schema of a paper centered in a solution to a problem.[3] The problem must be related to the audience of the discussion. Depending on

[3] In the figures, many relations are actions rather than function words as proposed in (12). Technically, this is justified by Assumption 3.6.12 of (12), which introduced relational definitions. This leads to more compact and more human-readable graphs here.

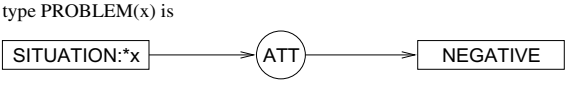


Fig. 2. Type Problem

A generic problem is a situation perceived as negative. The situation may be the openness of a question, the high cost of some procedure, etc.

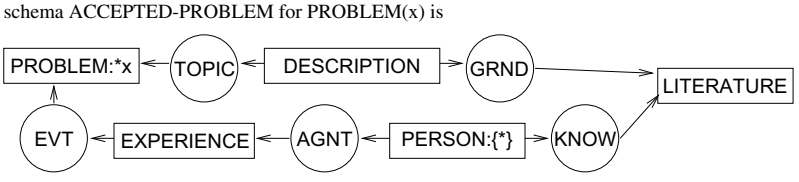


Fig. 3. Schema Accepted Problem

An accepted problem is described in the literature known to the audience. If there is active research activity, the Open Problem schema is more appropriate.

its further expansions, the problem may be related to literature known to the audience, or the author may ground the problem in common experience. The author of the document proposes a solution which solves the problem.

A problem is characterized by being a situation which is valued as negative (Type 2). This definition is of little use for the author, though. In the discussion, a problem may be structured as a known problem (it is described in some literature, Schema 3), or as an open problem (there are several proposed answers and some body of literature, Schema 4). Alternatively, a problem may be grounded on topoi commonly believed by the audience (Schema 5).

Solutions are the other side of a problem. Their type is omitted here. A solution may base its value on being implemented and used (Schema 6). An impressive example of this is found in (16), mentioning "120 documents of major proportions had been produced by the method" (p. 68/A-15). A conceptual or

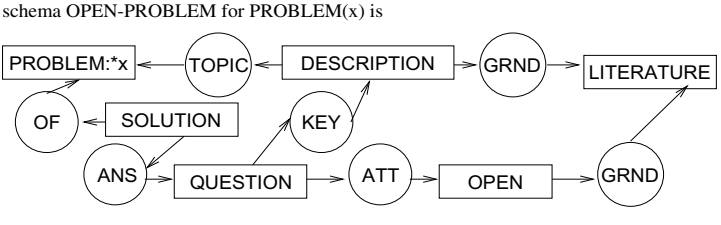


Fig. 4. Schema Open Problem

An open problem is the object of research activity, so there is literature on the overall goal and interim results. There may also be several not fully established answers.

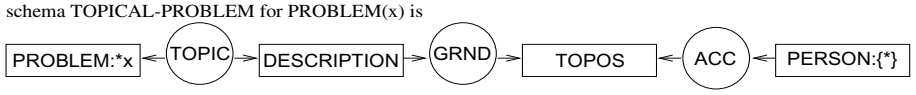


Fig. 5. Schema Topical Problem

The topical problem rests on the assumption that a topos is accepted in the audience. It needs no further justification of its existence. The relevance must still be assured.

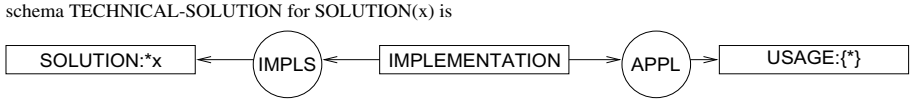


Fig. 6. Schema Technical Solution

The solution is available in some implementation, which may have been applied in a number of usages.

methodological solution may ground itself on its beauty or the novelty. Note that both schemata may be applied, so a solution may be somewhat implemented, but appeal mainly by its novelty.

In this paper, the introduction begins with stating the topical problem of writing complex documents. By citing authoring literature such as (14) and closer exposition, it states the author's and reader's problems as known problems.

Presentation Schemata Help Organize the Presentation. Depending on the space available and her judgment, the author expands a number of presentation schemata, populating the presentation with a structure of the usual hierarchy of sections and subsections. Specific concepts in the presentation structure may discuss concepts of the content, or one concept may subsume a number of others. By ordering and stating a certain depth, the schema guides the author in formulating a coherent text.

In the case of this paper, a schema that discusses problems first and presents a solution later (Figure 7) has been used. This top-level schema is just an overview, as the finer points of interconnection are provided by lower-level schemata for the presentation of, for example, problems or related work.

Alternative presentation schemata present a vision (there is no implementation, even the problem may be low-key) or report on some findings (fits together with Open Problem Schema (Schema 4)) or give an account of how time was spent on a project (a natural choice for **Status Report** patterns). The IMRD format of (14) or van Dijk's superstructures (17) are other candidates for presentation schemata.

Type Definitions Enforce Constraints. While schemata may be violated, a type definition may not. A type definition for a submission to a conference would usually require at least one author and a title for the document. As a

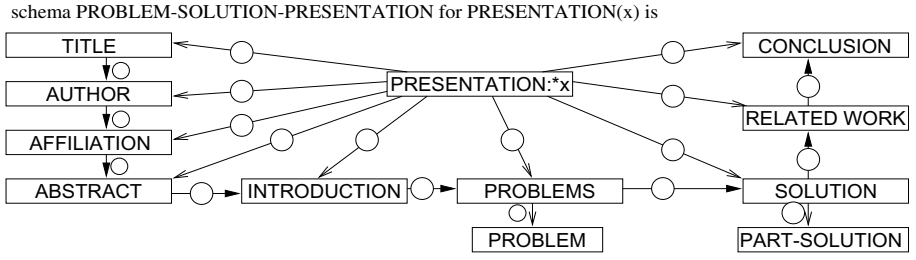


Fig. 7. Schema Problem-Solution Presentation

In the abstract, prerequisites (presumed facts or topoi) are skimmed, followed by a discernible description of the problem. The idea and main aspect of the solution are also mentioned. The introduction elaborates on these aspects to ease the reader into the realm of the paper. The problems section describes a number of problems and selects the ones that are about to be solved. The discussion of the solution can be layered or segmented in parts. In the related works section, the other work should address roughly the same problems, but provide no complete solution.

means of authoring support, the type serves to enforce basic requirements and reduces copy editing cost at the publisher. Type constraints may also be used on lower levels, such as prohibiting dangling cross-references.

4 High-Level Functionality of the CAS System Chasid

We are applying these plans in the prototype high-level authoring support system Chasid (Consistent High-level Authoring and Studying with Integrated Documents). It extends conventional documents and authoring environments with graph structures that capture the content and the presentation structure. On this basis, schemata can be expanded, and their fulfillment can be monitored and supported.

From the perspective of a user that is used to the conventional application, our implementation is called the "conceptual extension" or "semantics extension".

The author may interactively expand schemata or types to edit the content or presentation structure. She may also associate presentation nodes with portions of the conventional document. These operations are immediately visualized in a graphical display. Figure 8 shows a screenshot of the running prototype, with open pattern and schema browsers and a cutout of the conceptual graph of this paper. Connections to the hierarchical presentation structure or the external document are not displayed. Some operations perform arbitrary type-conforming operations on the graph. These operations are not necessarily canonical, but they do not violate the type expansion rules.

The graph structures may be scanned for problematic patterns, which are then flagged with warning nodes. The warnings may be dismissed on instance level by the author; for some warnings, there are also repair operations. For

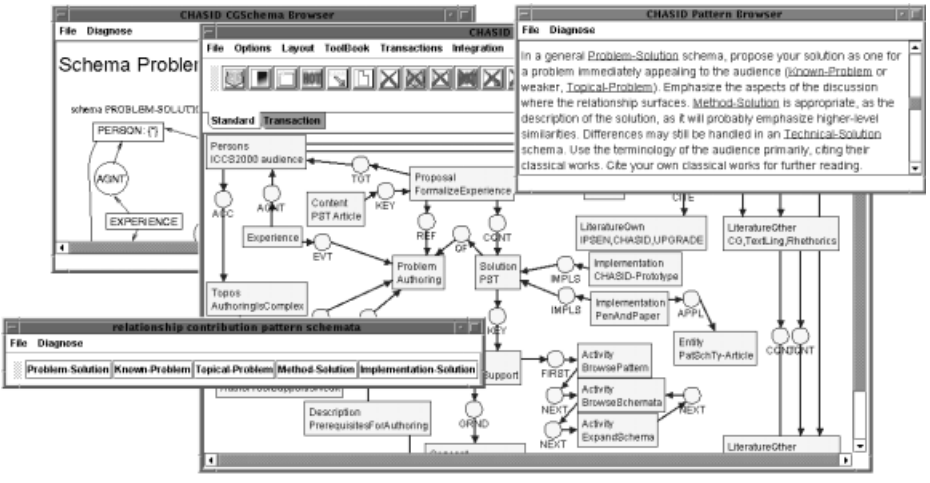


Fig. 8. Screenshot of the CHASID prototype.

example, hierarchy anomalies such as nodes with only one child are flagged. If the content model contains a group of concepts and that group is linked to a hierarchical concept, it is ensured that the latter contains the corresponding sub-concepts.

Manipulations of the graph structures are propagated to the conventional document. Correspondingly, edits in the conventional document are propagated into the graph structures, if they are structurally relevant.

The author may alternatively use the system bottom-up instead of top-down, as described previously: She may build the document in the conventional application and the graph-based extension keeps track of the structure as best as it can. This gives the author a graphical view of the presentation structure of the document. She may manipulate this, causing changes in the conventional document. From the structure, the content graph may be built by attaching *imported* and *exported* concepts to the presentation concepts. Between these content model concepts, relations such as prerequisite-for may be defined to trigger warnings should a prerequisite be described only after that which needs it. We discuss these features in more depth in (6).

5 Implementation Structure of Chasid

The system is being implemented using the high-level graph transformation language PROGRES.[4] It is based on declaring a graph schema (closer to database schemata than conceptual schemata) and productions, tests, path expressions, transactions and consistency constraints on them. The evaluation can proceed deterministically in imperative command sequences, or may use nondetermin-

[4] (8; 10), `http://www-i3.informatik.rwth-aachen.de/research/progres`.

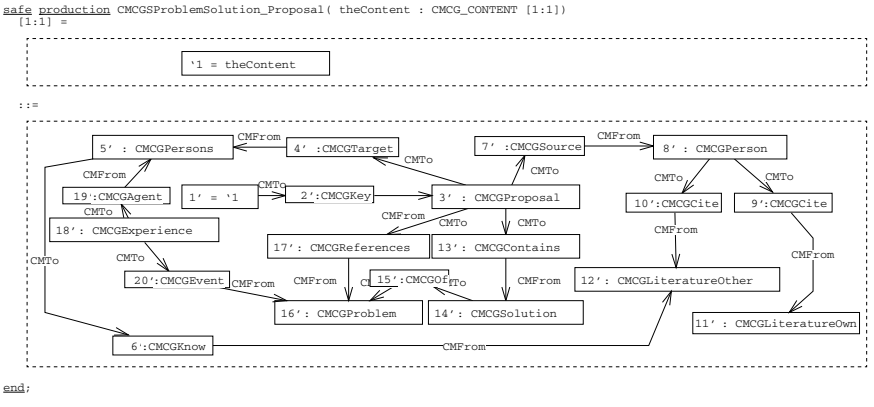


Fig. 9. PROGRES production to instantiate a PROBLEM-SOLUTION schema

ism and backtracking to find possible solutions. The most important language feature are graph rewriting operations (productions), where arbitrary graph patterns may be partially replaced with new graphs. As an example, figure 9 shows a graph production that inserts the schema PROBLEM-SOLUTION (Schema 1) into the current working graph. To edit PROGRES specifications, a language-sensitive editor and interpreter are available.

From a PROGRES specification, a prototype application can be generated where the productions and transactions of the specification can be executed interactively. This prototype is connected to the conventional application through various a-posteriori integration techniques. The integration code detects changes on a very low level and translates them into transactions on a portion of the graph that closely mirrors the structure of the conventional document. Consistency constraints trigger transactions that propagate these changes to changes in the "higher" parts of the graph.

Promoting changes back is implemented using database triggers. Each change in the graph causes a change in the graph database where it is stored. This in turn triggers events. These events are monitored by the integration code and translated into editing operations of the conventional application.

The current implementation integrates with ToolBook. This application has been chosen because it is built on a relatively open message-passing structure. This choice also reflects the roots of this project in the multimedia context.

The prototype consists of a framework written in Java (7), the C code generated from the specification and the graph database, which is currently a legacy system written in Modula-3 and C. The database restricts the prototype to the Solaris and Linux platforms. The integration code uses Delphi and the ToolBook OpenScript languages. It is integrated into ToolBook instructor II 6.1, which runs on MS Windows. The communication runs over a proprietary TCP protocol. The implementation is still far from complete.

6 Related Work

The work described here touches a number of research areas that provide a contribution, but not a solution to the problems outlined above.

Linguistics and its popular branches are a source for various kinds of schemata and patterns. T. A. van Dijk's research in superstructures (17) did reveal some presentation structures, but did not rate them in terms of readability. There is a wealth of material on the construction and formulation of arguments, with Toulmin (15) one of the most well-known protagonists. These treatises provide schemata on more fine-grained levels than described here.

One area where active authoring support of the kind presented here is given are pragmatic text schema books. As early as the 16th century, books have been published containing prefabricated letters for specific communication purposes (3). Today, such collections have been translated into electronic form and have become somewhat more flexible. Still, they heavily rely on the specific communication purpose, which may only be parameterized to a certain extent. There are also only few checks for consistency.

A different approach to the problem of structuring a document was adopted by the STOP team (16). They introduced "thematic quantization", forcing writing managers and writers to plan their documents so each two-page spread handles one theme. By enforcing early structuring, the task of the single writer becomes more productive, and editing loops are factored out. There are, however, little hints on structure beyond the two-page spread.

The "Writer Environment" family of computer-supported writing projects maintains a separation of the document into content space, rhetorical space, planning space and argumentation space (11; 13). The same concepts are used here with the content model (corresponding to the content space), presentation structure (argumentation space) and conventional document (rhetorical space). There is no separate organizing planning space as it does not directly contribute to solving writers' organization problems. The development has produced highly flexible teamwork hypermedia authoring environments. From our perspective, they have two drawbacks: they do not integrate tightly with existing applications, and there is little support in building the structure.

The idea of imports and exports of text blocks can be used to tailor a lesson for a specific reader from a large body of knowledge. This has been done by Nejdl (9). This system does not address the authoring support problems. Also, coherence may be hard to achieve when the informational units are rearranged.

7 Conclusion: Thesis, Usage, Plans

In this paper, I have presented an application of conceptual structures in supporting authors to construct coherent and lucid documents more efficiently. The application uses patterns as problem clarification guidance, various schemata as content and presentation structure help and type definitions for enforcing external requirements.

As the system is still in its implementation stage, it has not yet been applied to any text production. The concepts, however are used with pen and paper in the preparation of this document and some earlier publications. This has shown that a lasting integration of the graphs with the document is crucial: updating the paper graph has always been discontinued at some time during editing.

This state defines the further plans: Implement the integration and specification so that the system may be applied in real-life projects. We are also looking at some existing texts to determine more well-grounded patterns and schemata. In order to reach more text-oriented authors, an integration with Microsoft Word is planned.

Thanks

I would like to thank Oliver Meyer for the great cooperation, the PROGRES and UPGRADE teams for their neat tools and David Rebnord for proofreading the paper.

References

[1] M. J. Adler and C. van Doren. *How to Read a Book*. Simon & Schuster, Riverside, NJ, July 1997. ISBN 1-567-31010-9. First edition 1940.

[2] C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King, and S. Angel. *A Pattern Language*. Oxford University Press, New York, 1977.

[3] G. Antos. Mustertexte und Schreibprozeduren. Standardisiertes Schreiben als Modell zur Aneignung von Schreibprozeduren (Text Templates and Writing Procedures. Standarized Writing as a Model for Apprehending Writing Procedures). In J. Baurmann and R. Weingarten, editors, *Schreiben – Prozesse, Prozeduren und Produkte (Writing – Processes, Procedures and Products)*, pages 70-84. Westdeutscher Verlag, Opladen, 1995. ISBN 3-531-12627-X.

[4] E. S. Forster, editor. *Aristotle II*, volume 391 of *Loeb Classical Library*, chapter Topica, pages 265-740. Harvard University Press, Cambridge, MA, 1960. ISBN 0-674-99430-2. Reprinted 1989.

[5] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns – Elements of Reusable Object-Oriented Software*. Professional Computing. Addison-Wesley, Reading, MA, 1994.

[6] F. Gatzemeier and O. Meyer. Improving the Publication Chain through High-Level Authoring Support. In M. Nagl, A. Schürr, and M. Münch, editors, *Applications of Graph Transformation with Industrial Relevance* (AGTIVE), volume 1779 of LNCS, Heidelberg, 2000. Springer.

[7] D. Jäger. Generating tools from graph-based specifications. In J. Gray, J. Harvey, A. Liu, and L. Scott, editors, *Proceedings First International Symposium on Constructing Software Engineering Tools (CoSET '99)*. University of South Australia, School of Computer Science, 1999.

[8] M. Nagl, editor. *Building Tightly Integrated Software Development Environments: The IPSEN Approach*, volume 1170 of LNCS. Springer, Heidelberg, 1996. ISBN 3-540-61985-2.

[9] W. Nejdl and M. Wolpers. KBS Hyperbook – A Data-Driven Information System on the Web. In *Proceedings of the eigth international World Wide Web Conference*, Toronto, 5 1999.

[10] A. Schürr. Operationelles Spezifizieren mit programmierten Graphersetzungssystemen. PhD thesis, RWTH Aachen, Deutscher Universitätsverlag, Wiesbaden, 1991.

[11] J. B. Smith and M. Lansman. A Cognitive Basis for a Computer Writing Environment. In B. K. Britton and S. M. Glynn, editors, *Computer Writing Aids: Theory, Research, & Practice*, pages 17-56. Erlbaum, Hillsdale, NJ, 1989.

[12] J. F. Sowa. *Conceptual Structures: Information Processing in Mind and Machine.* The Systems Programming Series. Addison-Wesley, Reading, MA, 1984.

[13] N. A. Streitz, J. M. Haake, J. Hannemann, A. Lemke, W. Schuler, H. Schütt, and M. Thüring. SEPIA – A Cooperative Hypermedia Authoring System. In D. Lucarella, J. Nanard, M. Nanard, and P. Paolini, editors, Proceedings of ECHT'92, the Fourth ACM Conference on Hypertext, pages 11-22, Milano, Italy, 1992. ACM. URL ftp://ftp.darmstadt.gmd.de/pub/concert/publications/SEPIApaper.ps.Z.

[14] J. M. Swales and C. B. Feak. *Academic Writing for Graduate Students – A Course for Nonnative Speakers of English.* Michigan Series in English for Academic & Professional Purposes. University of Michigan Press, Ann Arbor, 1998.

[15] S. Toulmin, R. Rieke, and A. Janik. *An introduction to reasoning.* Macmillan Publishing co., Inc., New York, first edition, 1979. ISBN 0-02-421030-7.

[16] J. R. Tracey, D. E. Rugh, and W. S. Starkey. Sequential Thematic Organization of Publications (STOP): How to Achieve Coherence in Proposals and Reports. *Journal of Computer Documentation*, 23(3):4-68, Aug. 1999. Reprint of a January 1965 article.

[17] T. van Dijk. Macrostructures: *an interdisciplinary study of global structures in discourse, interaction and cognition.* Lawrence Elbaum Associates, Inc., Hilldale, NJ, 1980. ISBN 0-89859-039-6.

Conventions and Notations for Knowledge Representation and Retrieval

Philippe Martin

Knowledge, Visualization and Ordering Labororatory
School of IT, Griffith University, Queensland 9726
philippe.martin@gu.edu.au

Abstract. Much research has focused on the problem of knowledge accessibility, sharing and reuse. Specific languages (e.g. KIF, CG, RDF) and ontologies have been proposed. Common characteristics, conventions or ontological distinctions are beginning to emerge. Since knowledge providers (humans and software agents) must follow common conventions for the knowledge to be widely accessed and re-used, we propose lexical, structural, semantic and ontological conventions based on various knowledge representation projects and our own research. These are minimal conventions that can be followed by most and cover the most common knowledge representation cases. However, agreement and refinements are still required. We also show that a notation can be both readable and expressive by quickly presenting two new notations – Formalized English (FE) and Frame-CG (FCG) – derived from the CG linear form [9] and Frame-Logics [4]. These notations support the above conventions, and are implemented in our Web-based knowledge representation and document indexation tool, WebKB[1] [7].

1 Introduction

In [7], we argue that to permit *precise, flexible and scalable* retrieval and exploitation of knowledge representations (e.g. conceptual ontologies) and data indexed by them, the used metadata/knowledge representation languages should possess an expressive, intuitive and concise linear form; permit the indexation of any document and part of document; support the use of undeclared terms; and permit the specification of paths in a semantic network. We argued against the direct use of XML-based languages such as RDF[2], and shown how WebKB and its languages satisfy these requirements.

Precise, flexible and scalable knowledge retrieval also requires the agents (e.g. Web users or robots) who generate knowledge representations to follow conventions to permit subsequent comparison of the representations, and therefore their retrieval and fusion. In this paper, we propose lexical, structural, semantic and ontological conventions based on various knowledge representation projects (especially Conceptual Graphs [9] and RDF [1]) as well as our own research [5] [6].

[1] http://meganesia.int.gu.edu.au/~phmartin/WebKB/
[2] http://www.w3.org/RDF/

B. Ganter and G.W. Mineau (Eds.): ICCS 2000, LNAI 1867, pp. 41–54, 2000.
© Springer-Verlag Berlin Heidelberg 2000

We also show that a notation can be both readable and expressive by introducing two notations – Formalized English (FE) and Frame-CG (FCG) – derived from CGLF and Frame-Logics. We do not expect these to become standards but they are new alternatives to CGLF and CGIF. They support and encourage the above conventions and associated facilities, and are implemented in our Web-based knowledge representation and document indexation tool, WebKB [7].

2 General Conventions

Our conventions are general but, since we use the Conceptual Graph [CG] terminology to refer to components of knowledge representations, we assume these representations can be translated into such a directed graph model.

2.1 Lexical Normalisation

InterCap Style for Identifiers. XML[3] has become the *de facto* standard for data exchange, and RDF[4] and its XML notation ("RDF/XML") will probably become the standard for metadata exchange. Therefore, it seems important that identifiers within knowledge representations have legal XML names. This is not particularly restrictive (URLs are permitted). More importantly, the "InterCap style" has been adopted in RDF for expressing terms, with a lower case first letter for relation types[5] – as in *rhetoricalRelation* and *subClassOf* – and an upper case first letter for concept types[6] – as in *TaxiDriver*. These naming conventions have also been adopted by the "Meta Content Framework Using XML"[7].

High-level lexical facilities. To be used widely and reduce lexical problems, high-level languages or query interfaces should provide lexical facilities for the user. For instance, as is the case in WebKB, language analyzers should automatically normalize identifiers that include uppercase letters, dashes or underscores into the Intercap style, as well as exploit user-defined aliases.

Such analyzers also accept queries or representations that use undeclared type names (e.g. common words) when the relevant type names can be automatically guessed via the structural and semantic constraints in the queries or representations and the ontologies they are based upon. When different interpretations are possible, the user should be alerted to make a choice. This last facility, detailed in [7], is particularly interesting when the exploited ontologies reuse a natural language lexical database such as WordNet[8]: it spares the user the complex and tedious work of declaring and organizing each term used. This facility (along with high-level notations and interfaces) seems an essential step to encourage Web (human) users to build knowledge representations. Similar ideas for the exploitation of lexical databases such as WordNet are developed in [3].

[3] http://www.w3.org/XML/
[4] http://www.w3.org/RDF/
[5] http://www.w3.org/TR/REC-rdf-syntax/#usage
[6] http://www.w3.org/TR/1998/WD-rdf-schema/#intro
[7] http://www.w3.org/TR/NOTE-MCF-XML/#secA.
[8] http://www.cogsci.princeton.edu/~wn/

Nouns for identifiers. Generally a sentence can be re-written to avoid the use of adjectives and verbs (with the exception of "to be" and "to have"). For instance, "A cat named Tom jumps toward a wooden table" may be re-written into "The cat which has for name Tom is agent of a jump that has for destination a table the material of which is some wood". This sentence – which is a correct sentence in Formalized English (FE) – seems unnatural but makes the concepts and their relations explicit and therefore exploitable by an automated analyzer.

The convention of using nouns, compound nouns or verb nominal forms whenever possible within representations not only makes them more explicit, it also efficiently reduces the lexical and structural ways they may be expressed. It therefore increases the possibilities of matching them.

Concept types denoted by adjectives[9] can rarely be organized by generalization relations but may be decomposed into concept types denoted by nouns. Concept types denoted by verbs can be organized by generalization relations (though the organization of the top-level types is difficult) but cannot be inserted into the hierarchy of concept types denoted by nouns (and therefore cannot be compared with them) unless verb nominal forms are used. These nominal forms, e.g. *Driving*, also recall the need to represent the time-frame or frequency of the referred processes. For similar reasons, value restrictors should also be represented via noun phrases, e.g. *ImportantWeightForAMouse* and *ImportantWeightForAnElephant*, rather than via adjectives such as *Important*.

Most identifiers in current ontologies are nouns (e.g. the Dublin Core[10] or the Upper Cyc Ontology[11]), even in relation type ontologies such as the Generalized Upper Model[12] relation hierarchy. Avoiding adverbs for relation type names is sometimes difficult, e.g. for spatial/temporal relations. However, this does not create problems in organizing relation types by generalization relations. What should be avoided is the introduction of relation type names such as *isDefinedBy* and *seeAlso*. Better names are *definition* and *additionalInformation*. These names are consistent with the usual reading conventions (e.g. in CG [9] and RDF[13]) of graph triplets {concept source, relation, concept destination}:
"`<concept source> HAS FOR <relation> <concept destination>`" or
"`<concept source> IS <relation> <concept destination>`" or
"`<concept destination> IS THE <relation> OF <concept destination>`".

Singular nouns for identifiers. Most identifiers in ontologies are singular nouns. Category names must be in the singular in the Meta Content Framework Using XML. It is therefore better to avoid the introduction of plural identifiers whenever possible, e.g. by using keywords within representations such as the CG keyword `Dist` that specifies that a referent is distributive.

[9] We refer to types representing the meanings of some adjectives, not misnamed types such as *Abstract* when "Abstract Entity" is actually the intended meaning
[10] http://purl.oclc.org/dc/
[11] http://www.cyc.com/cyc-2-1/cover.html
[12] http://www.darmstadt.gmd.de/publish/komet/gen-um/node11.html
[13] http://www.w3.org/TR/REC-rdf-syntax/#statement

2.2 Structural and Semantic Normalisation

We have seen that lexical conventions and facilities influence the structural and semantic aspects of representations. We now focus on these aspects.

First-class relations. In CG and RDF, a relation is not *local* to an object, it is a first-class object itself and can be connected to any instance of the types given in the signature of the relation. This permits distributive developments (since anyone may represent anything about any object) and eases the process of comparing representations (since all terms are inter-related). This still permits the representation of relations necessarily or typically associated to objects of a particular type. Thus, although most frame-based systems also allow local relations, it is better to avoid them for the sake of knowledge reuse.

Binary basic relations. Most frame-based models (including RDF) only have binary and unary relations. It is therefore better for knowledge reuse to use only unary or binary relations in languages such as CGs that allow n-ary relations. Relationships of arity greater than 2 may always be represented using structured objects or collections, or more primitive binary relations. For instance, "the point A is between the points B and C" may be represented using the binary relation type *between* and a collection object grouping B and C, or using the relation types *left* and *right*, *above* and *under*, etc. Most often, such a decomposition makes a representation more explicit, precise and comparable with other representations.

Thus, relations should rather refer to simple/primitive relationships. As a rule of thumb, relations should not refer to processes and should rather be named with simple "relational nouns", e.g. *part* and *characteristic*. Some complex relational nouns such as *child* and *driver* are often too handy to be avoided but imply additional lexical or structural facilities (e.g. those of Ontoseek [3]).

Avoid disjunctions, negations and collections. Representations that include disjunctions, negations or collections are generally less efficiently exploitable for logical inference than conjunctive existential formulas and IF-THEN rules based on these formulas [14].

It is often possible to avoid disjunctions and negations without loss of expressivity using IF-THEN rules or by exploiting type hierarchies. For instance, instead of writing that an object X is an instance of *DirectFlight* OR of *IndirectFlight*, it is better to declare X as an instance of a type *Flight* that has *DirectFlight* and *IndirectFlight* as exclusive subtypes (i.e. types that cannot have common subtypes or instances). Exclusion links between types or in some cases between whole formulas are kinds of negations that can be handled efficiently, and are included in many expressive but efficient logic models, e.g. Courteous logic on which the Business Rules Markup Language (BRML) is based.

The introduction of identifiers for collections may also be avoided using keywords such as `Dist` to specify a distibutive interpretation or `Col` for a collective

[14] http://www.oasis-open.org/cover/brml.html

interpretation. Type definitions are also a way of representing facts about collections of objects that knowledge representation systems generally handle more efficiently than if collections are directly used.

Contexts are often unavoidable for expressivity sake but they can be handled efficiently if treated as positive contexts [2], that is, when only their structures are taken into account, and not the special semantics of the terms they include.

For efficiency reasons again, many knowledge representation systems, especially frame-based systems, work on rooted graphs. In a rooted graph, there is a head node representing the "central" object of the representation, i.e. the object the other nodes detail. Therefore, to help the translations of representations for various kinds of systems, it seems better to begin each representation with its central object, whatever the language used.

Precision, term definitions and constraints. The more precise the representations are, the less chance they conflict with each other and the more they can be cross-checked, merged and exploited to answer queries adequately. Hence, constraints should be associated to types, and representations should rather be contextualized in space, time and author origin. No relevant concept should be implicit. For instance, instead of representing that "birds fly", it seems better to represent that "a study made by Dr Foo (Foo@bird.org) found that in 1999, 93% of healthy birds can fly" and categorize the species of birds under the exclusive subtypes *BirdWhichCanNormallyFly* and *BirdWhichCannotNormallyFly*.

Most notations make it very difficult to represent such precise statements. Even Sowa's CG linear format (CGLF)[9] needs to be extended to refer to the distributive interpretation of 93% of all instances of a type X; we use the form "[X: ∀ @93%]" in the CGLF representation of the previous example: [15]

```
[Description: [Situation: [PhysicalPossibility:
   [[Bird:λ]->(chrc)->[Health:@good]: ∀  @93%]<-(Agent)<-[Flight]
                          ] ]->(time)->[Date:"1999"]
]->(source)->[Study]->(author)->[Person: Foo@bird.org]
```

In Section 3.1 below, we give the translations of this example in FCG and FE to show that more intuitive notations are possible and useful.

Before doing so, let us note that representations that include precise domain-oriented terms should still be retrievable via queries which include more general natural language terms. A way to do this is to specialize the terms of a natural language ontology such as WordNet with the domain-oriented terms. Extending such an ontology is often quicker (and safer) than creating an ontology from scratch, ensures a better reusability of the representations and automatic comparisons with representations based on the same ontology. These issues are discussed and implemented in Ontoloom/Powerloom[16].

[15] It would have been very difficult to represent "birds of most species can fly" and the result hardly exploitable for logical inference.

[16] http://www.isi.edu/isd/OntoLoom/hpkb/OntoLoom.html#RTFToC18

3 Notations

To permit Web users to precisely index Web documents, or more generally, represent knowledge, the notation(s) they use needs to be both expressive and intuitive. Otherwise, they will not use it or they will be forced or encouraged to represent information inadequately, which greatly reduces the value of the representations. A usual concern is that an increase in expressiveness leads to a model and a language too complex to handle efficiently. Actually, with structured models such as frames, RDF or CG, an inference engine may easily take more or less features into account to provide the degree of precision/efficiency required by a function or an application. For instance, a search engine can do a good and efficient job exploiting simple structure matching techniques and considering all contexts (e.g. modalities and negation) as positive contexts, so long as the retrieved representations are displayed with their associated contexts.

In this section, we list knowledge representation cases that are common in natural language sentences but rarely taken into account by current general-purpose knowledge representation languages. We do not assume any kind of exploitation or formalization. We simply show how these cases can be represented in CGLF, FCG and FE, and why it sometimes remains ambiguous in CGLF.

We created FCG and FE in order to have notations both more readable and expressive than CGLF (and therefore CGIF). We believed it was a necessary step for our Web-based information annotation/retrieval tool to be usable and useful. FE (Formalized English) has the same readablility and precision purposes as other controlled languages such as Attempto Controlled English (ACE)[17] but is not domain-dependent and is more expressive (none of the following examples seems to be representable with ACE). FCG and FE are CG notations derived from CGLF: the structure has been kept but some syntactic sugar is different. Because arrows have been removed (mainly replaced by colons in FCG and keywords such as "that has for" in FE) and since common English articles or other "modifier" words can be used as quantifiers, these notations seem simpler than CGLF (and sometimes more expressive where set quantification and modalities are concerned). A "frame-like" notation similar to FCG and also using "a", "the" and "every" as quantifier keywords is proposed by the Knowledge Machine [8] for the same readability purpose (however, it does not have other quantifiers). We cannot detail these notations here but their EBNF grammars and the Yacc+Lex grammars for their automatic translation to CGLF are Web-accessible [18].

All the terms in our examples are WordNet nouns apart from some keywords, URLs, relation types and concept type instances used as measure values.

3.1 Presentation of FCG and FE

Here is the previous example represented in English, CGLF, FCG and FE.

[17] http://www.ifi.unizh.ch/groups/req/staff/fuchs/. See also the Boeing Simplified English (http://www.boeing.com/assocproducts/sechecker/) for examples of restricted though still informal English.

[18] http://meganesia.int.gu.edu.au/~phmartin/WebKB/doc/grammars/

```
E:    A study made by Dr Foo (Foo@bird.org) found that in 1999,
      93% of healthy birds can fly.

CGLF: [Description: [Situation: [PhysicalPossibility:
        [[Bird:λ]->(chrc)->[Health:@good]: ∀  @93%]<-(Agent)<-[Flight]
                          ] ]->(time)->[Date:"1999"]
      ]->(source)->[Study]->(author)->[Person: Foo@bird.org]

FCG:  [[[93% of(bird,chrc:a good health),agent of#:a flight],time:1999],
       source: (a study, author: Foo@bird.org)]

FE:   ‘‘‘93% of [bird with chrc a good health] can be agent of a flight’
       time 1999’ with source a study that has for author Foo@bird.org’.
```

Contexts are delimited by square brackets in FCG and quotes in FE. At the same context level, structuration is done via parenthesis in FCG and the use of comma or keyword “and” in FE. Lambda-expressions are delimited by parenthesis in FCG, square brackets in FE. Apart from these distinctions, both notations share the same features: the quantifier keywords (e.g. “a” and “the” as existential quantifiers, “several” and “at least” as collection quantifiers), the lexical facilities (e.g. the automatic normalization of terms) and other facilities (e.g. the automatic typing of contexts – and the creation of intermediary contexts if necessary – according to the signatures of the relations connected to them).

3.2 Existential Statements, Contexts and Sentence Delimiters

Simple information or sentences from documents can be represented using most general-purpose knowledge representation languages (including CG, KIF and RDF) since they permit to represent existential statements and contextualize them (e.g. to represent modalities). We therefore do not detail these features.

However, most of these languages, even metadata languages such as XML or RDF, still lack delimiters to include arbitrary sentences of any language (e.g. English or XML) inside the representations. FCG and FE have several pairs of delimiters, including “$(“ and “)$’. The next FCG illustrates a use for them with the representation of the title of certain document. It also shows how the details of a representation are made explicit by representing the representation process itself with a concept and then connecting binary relations to it.

```
[a representation, agent: philippe.martin@gu.edu.au, language: FE,
  ontology: http://www.int.gu.edu.au/~phmartin/WebKB/kb/KADS1ontol.html,
  creationDate: "21/01/1999", expirationDate: "22/7/9999",
  source: (the title, part of: http://foo.bar.org/KADS1.html),
  object: $( KADS-I models in CG )$, /* <- text of the title */
  result: $( KADS-I has for part several models that are object of
             a representation which has for language CG. )$ /*FE repr.*/ ]
```

3.3 Collections and Intervals

Sowa [9] uses the symbols *Dist*, *Col* and *Cum* to make explicit the distributive, collective or cumulative interpretation of a collection referent. *Dist* or *Col* should not appear in more than one concept of a CG since this would lead to

ambiguities. Therefore, coreferences between collection concepts of various CGs. Unfortunately, this has not been dealt with by Sowa. The following example illustrates shows the problem with CGLF.

```
E:    Together, Fred, Tom and another man approved a resolution.
      A certain resolution was approved by each of them.
CGLF: [Person: *s Col{Fred,Tom,*}@3]<-(approver)<-[Resolution]
      [Set:*s]   [*s Dist{*}]<-(approver)<-[Resolution: @certain]
FCG:  [*g the group of persons *p {Fred,Tom,a man},
          approver of: a resolution]  [a resolution, approver: *p]
FE:   *g the group of persons *p {Fred,Tom,a man}
          is approver of a resolution. A resolution has for approver *p.
```

This example also shows how we used `[Set:*g]` in the CGLF to specify that each member of the group `*g` is different. In FCG and FE, a collection is by default a set. The distributive interpretation is also the default. The collective interpretation is specified with the keyword `group` and referred to via a variable – here `*g`. In the CGLF, we used Sowa's keyword `@certain` to specify that each member approved the same resolution. In FCG and FE, this is made explicit via the order of the concepts (as it is in English; we have adopted this solution because it is intuitive and it permits us to combine other kinds of quantifiers as the following examples show). The 's' at the end of the terms used for representing collections are automatically removed by the FCG parser.

Many keywords or special types need to be specified to permit the representation of collections, that is, (i) their kinds: `Set`, `Bag`, `OR-Bag`, `XOR-set`, etc. (ii) their restrictors: `most`, `mostly`, `at most`, `dozens`, etc. Below is an example involving two collections ("`*r`" is supposed to be pre-declared). The CGLF statement is ambiguous (the scopes of the quantifiers are not explicit).

```
E:    At least 3 persons, including Fred,
      have each approved most of the resolutions "r".
CGLF: [Person: Dist{Fred,*}@>=3]<-(Approver)<-[*r {*}@most]
FCG:  [most of *r, approver: at least 3 persons {Fred,*}]
FE:   Most of *r has for approver at least 3 persons {Fred,*}.
```

3.4 Universal Quantifiers

The keywords for collections are handy to reuse for quantifying over the instances of a type. If no restrictor is used (as in `[File:∀]->(Author)->[Agent]`), relations *necessarily* connected to the instances of a type (i.e. necessary conditions) are defined. The use of restrictors such as "`most`" or percentages are a way to define "typical" relations. Number intervals and keywords such as "`at least`" and "`at most`" may be used for representing relations of "entity-relationships" models, as in the FCG `[any company, employee: at least 1 person]`.

Modalities and physical possibilities may be represented via contexts. For readability and normalisation reasons, we introduced special keywords in FE (`can` and `may`) and FCG (`#:` and `<=`). Thus, the following FCG represents the sentence "any description describes something and *may* be believed by a cognitive agent": `[any description, descr of: a thing, believer<= a cognitive_agent]`

For the same reasons, we allowed some operators (`=>`, `<=>`, `<=`, `=`, `!=`, `<`, `=<`, `>`, `>=`) to be used as relations types in FCG and FE. Here is an example of the use of the logical operator `=>` and of the use of coreference inside relations for writing a second-order statement that is easy to read (in FCG).

```
FCG: [ [a relationType *r, chrc: the transitivity], =>
       [ [a thing *x, *r: (a thing *y, *r: a thing *z)], => [*x, *r: *z]]]

CG: ~[ [RelationType: *r]->(chrc)->[Transitivity]
      ~[ ~[ [Thing: *x]->(ρ r)->[Thing: *y]->(ρ r)->[Thing:*z]
           ~[ [*x]->(ρ r)->[*z] ] ] ] ]
```

4 Terms and Conventions for Ontological Cases

We now focus on how objects of certain categories can be inter-related.

4.1 Some General Categories

Relatively few top-level concept types are required for the signatures of most kinds of relations useful for general knowledge representation, e.g. for the representation of natural language sentences or images from documents. These concept types and the constraints associated to them (e.g. the exclusion links between them) are useful for organizing ontologies, guiding knowledge modelling and preventing certain inconsistencies. We detailed this in [7] and introduced a top-level ontology of 150 concept types and 150 basic relation types. We have used these concept types for organizing the upper levels of the WordNet ontology. In the remainder of this article, we focus on the most generic of these concept types and represent the necessary or possible relationships between their instances. In this way, we also propose a model for knowledge representation.

Before doing so, let us highlight with an example how these general types help render precise the knowledge representation process. We obtained a hierarchy of terms about computer and network technology. It appeared that the hierarchy was not a generalization hierarchy since it mixed various kinds of objects and therefore various kinds of relations. Here is an annotated extract.

```
Computing  //process or domain?
  Computer_hardware  //physical object!
    Compiler  //hardware or software?
  Computer_software  //description!
    Software_language  //description medium!
    Applications  //process description (may or may not be software)!
Networking  //process or domain?
```

Here is, represented in FCG, a part of the ontology that we have built to make explicit the general category of each of the above terms and thus permit semantic checks and the use of relations associated to the general categories. The author of the previous hierarchy had to be contacted to determine what some of the terms meant. Indentation is only for presentation purposes. [19]

[19] The relation *partition* connects a type to a set of partitions, each being a set of exclusive types. Thus, two pairs of brackets are required even for a single partition.

```
[Domain_object, subtype: {Computing_object Networking_object}]
  [Networking_object,partition:{{Network_software,Network_hardware}}]
[Physical_entity, subtype: Hardware]
  [Hardware, subtype: {Computer_hardware, Network_hardware}]
    [Network_hardware, partition: {{Local area network, Switch}}]
[Description, subtype: {Software, Application}]
[Descr_medium, subtype: {Software_language, Network_language}]
```

Here are the generalization and exclusion relations between our most generic concept types. Indentation is only for presentation.

```
[Thing (^the supertype of all first order concept types^),
   partition: { {Situation, Entity}, {Thing_playing_a_role} }]
  [Situation (^a thing that occurs in a region of time and space^),
     partition: {{State, Process}, {Phenomenon},
                 {Situation_playing_a_role}}]
  [Entity (^a thing that may be involved in a situation^),
     partition:{{Information_entity,Temporal_entity,Spatial_entity},
                {Collection}, {Entity_playing_a_role} }]
    [Spatial_entity (^an entity that occupies a space region^),
       partition: {{Space_location, Physical_entity,
                    Imaginary_spatial_entity}}]
      [Physical_entity (^a spatial entity made of matter^),
         partition: {{Inanimate_object, Living_thing},
                     {Goal_directed_entity(^cognitive entity^)}}]
    [Information_entity (^for information or its representations^),
       partition: { {Description, Descr_container,
                     Characteristic, Measure, Measure_unit} }]
      [Description (^description of a situation^),
         partition: { {Description_content, Description_medium} }]
        [Description_content,partition:{{Belief,Hypothesis,Narration}}]
        [Description_medium,partition:{{Symbol,Syntax,Language,Script}}]
      [Description_container,partition:{{File,Image,Document_element}}]
      [Characteristic (^ a dimension of something ^),
         partition: {{Psychological_characteristic,
              Physical_characteristic, Situation_characteristic}}]
      [Measure, partition: {{Psychological_measure,
                             Physical_measure,Situation_measure}}]
      [Measure_unit, partition: {{Psychological_measure_unit,
                  Physical_measure_unit,Situation_measure_unit}}]
  [Thing_playing_a_role (^subcategorisation is domain-dependent^)
     partition: {{Domain_object},{Thing_needed_for_a_process},
                 {Entity_playing_a_role,Situation_playing_a_role}}]
```

4.2 Descriptions

We now focus on necessary and possible relationships from concepts of the types listed above. We begin with "descriptions" and related objects.

```
[any thing,
  descr   <= a description,           //anything may be described
  descr_in<= a descr_container,       //e.g. in a file, a hologram
]

[any description,                     //or "proposition"
  descr of       : a thing,
  descr_medium   : a descr_medium,    //symbols or a language
  descr_container: a descr_container, //e.g. in a file
  author         : 1 entity,          //unique author
  believer   <= a cognitive_agent, modality <= a modality,
  logical_relation<=a description, rhetorical_relation<=a description
]

[any descr_container, descr_support: a physical_entity,
                      descr_in of: a thing]
[any descr_medium, descr_medium of: a thing]
[ [a thing *t, descr_in: a descr_container *c],
  <=> [*t, descr: (a description, descr_container: *c)] ]
```

Examples of logical relation types are *Or* and *Xor*. Examples of rhetorical relation types are *summary*, *motivation* and *antithesis*.

4.3 Characteristics and Measures

When representing characteristics, the characteristic itself should be distinguished from its measure(s), as in `[a pen, physChrc: (a length, measure: 12 cm)]`. Since this habit does not come naturally, abbreviations such as the following should probably be adopted as conventions: `[a pen, length: 12 cm]`. However, this implies additional constraints on the ontology and on its exploitation by the analyzers, e.g. *Length* should be declared as a subtype of a type *Characteristic* and this would have to be a predefined type in any analyzer. Here are relations for explicit representations.

```
[any thing, chrc <= a characteristic]  //e.g. speed, ingenuity
[any characteristic, measure <= a measure, chrc of <= a thing]
[any measure, quantity: a number, unit: a measure_unit,
              measure of: a characteristic]

[any physical_entity, physChrc <= a physical_characteristic]
[any goal_directed_entity,physChrc<=a psychological_characteristic]
```

A lot of concept types may be found in WordNet for physical or psychological features, e.g. *Memory* and *Cognition*, but unfortunately those types are not well organized and often mixed with misclassified types such as *Mind*, *Lexicon* and *Structure*.

4.4 Situations, Processes and Temporal Entities

Here are some representations of relationships that are – or maybe – connected to situations, processes and temporal entities. Most of the relations types were proposed by [9].

```
[any situation,
   s_succ of: a situation,  s_succ: a situation,
   location : a spatial_entity,
   time: a temporal_entity, duration : a duration,
   situationChrc<= a situation_characteristic
]
[any temporal_entity,
   time of <= a situation,  duration of <= a situation,
   temporal_order_relation <= a temporal_entity
]
[any process,
   triggering_event<= an event,       ending_event  <= an event,
   ending          <= a state,        ending of     <= a state,
   precondition    <= a state,        postcondition <= a state,
   initiator  <= a goal_directed_agent,  agent      <= an entity,
   instrument <= an entity,              object     <= a thing,
   experiencer<= a conscious_agent,      recipient  <= an agent,
   result     <= a thing,                sub_process<= a process,
   manner     <= a situation_characteristic,
   method     <= a description,       source <= a spatial_entity,
   destination<= a spatial_entity,    path   <= a spatial_entity
]
```

The relation types *input* and *output* may respectively be declared as subtypes of *object* and *result*. Further specializations are *input_output*, *object_to_modify* and *object_to_mute*.

Given we may use intervals and exclusive sets for representing time concepts, only two relation types seem necessary between situations and temporal entities: *time* and *duration*. Here are examples of two typical cases.

```
E:   On the 21/12/1999, John went to his office between 13h and 14h.
FCG: [[a travel, agent: John, destination: (an office, owner: John),
                 time: 13 to 14 hour], time: "21/12/1999"]


E:   John usually takes 20 min or 40 min to go to his office.
FCG: [most of (travel, agent: John,
                       destination: (an office, owner: John)),
         duration: {20 min | 40 min}]
```

These examples do not violate the signatures of *time* and *duration*. The following representation, which uses a context, should be considered by the analyzers as equivalent to the first of the above two examples.

```
[ [John, agent of: a travel, destination: (an office, owner: John)],
  time: 13 to 14 hour]
```

4.5 Miscellaneous

Here are additional common kinds of relationships.

```
[any thing, part <= a thing] //anything may have at least 1 part
[any physical_entity, material: a physical_entity]
[any collection, subset <= a collection,
                 element<= a thing, count: a natural]

[Order_relation, domain: Thing, range: Thing,
   partition: { {Spatial_order_relation, Temporal_order_relation},
                {Meet,In,Near,Before,After} }]
[Spatial_order_relation, subtype: {On,Above,Below}]
[On, subtype of: {Meet,Above}]
```

We have defined relation types such as *meet* and *near* as direct subtypes of *order_relation* and allowed them to connect any pair of concepts. Specializations of these types, e.g. *spatial_meet* and *temporal_meet*, could be defined to allow the use of more precise and constrained types upon which further semantic checks could be done. Such precise modelling may be found in the CYC and Ontolingua top-level ontologies (for instance, 2D and 3D spatial relations are distinguished). However, we cannot expect the average user to spend his time looking for the most specific terms in such libraries. Nonetheless, these libraries could be exploited by authoring tools to automatically find more specific relations that do not violate the signatures of the general relation used.

Though relations such as *part* or *subset* are partial order relations like *subtype*, for the sake of precision, they should not be directly connected to concept types. For instance, `[Airplane, part: Wing]` might be intended to represent the fact that "any airplane has for part a wing", but many alternative interpretations are possible: "any wing is part of a plane", "a wing is part of any plane", etc.

5 Conclusion

Information can be represented in many ways. For knowledge representations to be automatically comparable, conventions must be followed by authoring agents. We have proposed general lexical, structural and semantic conventions, then examined some knowledge representation cases that are common in natural language sentences but rarely taken into account by current general-purpose knowledge representation languages. We have introduced two notations (Frame-CG and Formalized English) that support and guide the use of these conventions (e.g. the syntax, the quantifiers and restrictors - "a", "the", "several", etc. - lead to the use of nouns as identifiers), cover the listed knowledge representations cases and remain intuitive. We argued that an inference engine can exploit expressive languages efficiently (at the expense of precision) by ignoring some of the more complex features. Finally, some top-level concept types and their relationships were provided to guide and check knowledge representation.

As highlighted above, the precise models that are found in CYC and Ontolingua top-level ontologies are certainly useful but will doubtfully be used directly

by human agents to represent information or quickly index documents, sentences or images. Instead, we expect people (e.g. Web users) to utilize a small set of relation types and use common words for concepts types: given the signatures of the relation types, a lexical database such as WordNet may be exploited by an authoring tool to derive the relevant concept types or ask the user for more precision [3] [7]. Users will also probably use scalable multi-user knowledge servers to refer, refine or complement lexical databases.

Acknowledgments

This work is supported by a research grant from the Australian Defense, Science and Technology Organisation.
I thank the reviewer who pointed out that the Knowledge Machine project [8] has similar principles to ours (frame-like notation with English articles, type partitions, etc.) and proposes a situation-based inference engine.
I also thank Pr. Peter Eklund for proof-reading this article and for our discussions on the ideas it contains.

References

1. Berners-Lee, T.: The Semantic Toolbox: Building Semantics on top of XML-RDF. http://www.w3.org/DesignIssues/Toolbox.html (W3C Note, 24 May 1999); see also the "Semantic Web Road map" at http://www.w3.org/DesignIssues/Semantic.html, and the "Web Architecture: Describing and Exchanging Data" at http://www.w3.org/1999/04/WebData
2. Chein, M., Mugnier, M.L.: Positive Nested Conceptual Graphs. In: ICCS'97, 5th International Conference on Conceptual Structures, Springer Verlag, LNAI 1257 (1997) 95–109
3. Guarino, N., Masolo, C., Vetere, G.: Ontoseek: Content-based Access to the Web. In: IEEE Intelligent Systems, Vol. 14, No. 3 (1999) 70–80
4. Kifer, M., Lausen, G., Wu, J.: Logical Foundations of Object-Oriented and Frame-Based Languages. In: Journal of the ACM, Vol. 42, 1995.
5. Martin, Ph.: Using the WordNet Concept Catalog and a Relation Hierarchy for Knowledge Acquisition. In: Peirce'95, 4th Peirce workshop, Santa Cruz (1995) http://www.inria.fr/acacia/Publications/1995/peirce95phm.ps.Z
6. Martin, Ph.: *Exploitation de graphes conceptuels et de documents structurés et hypertextes pour l'acquisition de connaissances et la recherche d'informations*, PhD Thesis, University of Nice - Sophia Antipolis, France (1996)
7. Martin, Ph., Eklund, P.: Embedding Knowledge in Web Documents: CGs versus XML-based Metadata Languages. In: ICCS'99, 7th International Conference on Conceptual Structures, Springer Verlag, LNAI 1640 (1999) 230–246. URL http://meganesia.int.gu.edu.au/~phmartin/WebKB/doc/papers/iccs99/iccs99.ps
8. Clark, P., Porter, B.: KM: The Knowledge Machine. http://www.cs.utexas.edu/users/mfkb/km.html
9. Sowa, J.F.: Conceptual Structures: Information Processing in Mind and Machine. Addison-Wesley, Reading, MA (1984)
   See also: http://www.bestweb.net/~sowa/ontology/index.htm

Ontology, Metadata, and Semiotics

John F. Sowa

Abstract. The Internet is a giant semiotic system. It is a massive collection of Peirce's three kinds of signs: *icons*, which show the form of something; *indices*, which point to something; and *symbols*, which represent something according to some convention. But current proposals for ontologies and metadata have overlooked some of the most important features of signs. A sign has three aspects: it is (1) an *entity* that represents (2) another *entity* to (3) an *agent*. By looking only at the signs themselves, some metadata proposals have lost sight of the entities they represent and the agents – human, animal, or robot – which interpret them. With its three branches of syntax, semantics, and pragmatics, semiotics provides guidelines for organizing and using signs to represent something to someone for some purpose. Besides representation, semiotics also supports methods for translating patterns of signs intended for one purpose to other patterns intended for different but related purposes. This article shows how the fundamental semiotic primitives are represented in semantically equivalent notations for logic, including *controlled natural languages* and various computer languages.

1 Problems and Issues

Ontologies contain categories, lexicons contain word senses, terminologies contain terms, directories contain addresses, catalogs contain part numbers, and databases contain numbers, character strings, and BLOBs (Binary Large OBjects). All these lists, hierarchies, and networks are tightly interconnected collections of signs. But the primary connections are not in the bits and bytes that encode the signs, but in the minds of the people who interpret them. The goal of various metadata proposals is to make those mental connections explicit by tagging the data with more signs. Those metalevel signs themselves have further interconnections, which can be tagged with metametalevel signs. But meaningless data cannot acquire meaning by being tagged with meaningless metadata. The ultimate source of meaning is the physical world and the agents who use signs to represent entities in the world and their intentions concerning them.

The study of signs, called *semiotics*, was independently developed by the logician and philosopher Charles Sanders Peirce and the linguist Ferdinand de Saussure. The term comes from the Greek *sêma* (sign); Peirce originally called it *semeiotic*, and Saussure called it *semiology*, but *semiotics* is the most common term today. As Saussure (1916) defined it, semiology is a field that includes all of linguistics as a special case. But Peirce (CP 2.229) had an even broader view of that includes every aspect of language and logic within the three branches of semiotics:

B. Ganter and G.W. Mineau (Eds.): ICCS 2000, LNAI 1867, pp. 55–81, 2000.
© Springer-Verlag Berlin Heidelberg 2000

1. *Syntax.* "The first is called by Duns Scotus *grammatica speculativa.* We may term it *pure grammar.*" Syntax is the study that relates signs to one another.
2. *Semantics.* "The second is logic proper," which "is the formal science of the conditions of the truth of representations." Semantics is the study that relates signs to things in the world and patterns of signs to corresponding patterns that occur among the things the signs refer to.
3. *Pragmatics.* "The third is... pure rhetoric. Its task is to ascertain the laws by which in every scientific intelligence one sign gives birth to another, and especially one thought brings forth another." Pragmatics is the study that relates signs to the agents who use them to refer to things in the world and to communicate their intentions about those things to other agents who may have similar or different intentions concerning the same or different things.

According to Peirce, semiotics is the science that studies the use of signs by "any scientific intelligence." By that term, he meant "any intelligence capable of learning by experience," including animal intelligence and even mindlike processes in inanimate matter. By Peirce's criteria, computer techniques for processing knowledge bases and databases could be called *computational semiotics.*

Unfortunately, most word processors deal only with a small subset of syntax. They have produced what St. Laurent (1999) calls *the WYSIWYG disaster*: "Plain text, dull though it may be, is much easier to manage than the output of the average word processor or desktop publishing program." In practice, the slogan "What you see is what you get" actually means WYSIAYG: "What you see is all you get." The text is so overburdened with formatting tags that there is no room for semantics or pragmatics. The so-called Rich Text Format (RTF) is semantically the most impoverished representation for text ever devised. Formatting is an aspect of signs that makes them look pretty, but it fails to address the more fundamental question of what they mean.

To address meaning, the markup languages in the SGML family were designed with a clean separation between formatting and meaning. When properly used, SGML and its successor XML use tags in the text to represent semantics and put the formatting in more easily manageable style sheets. That separation is important, but the semantic tags themselves must have a clearly defined semantics. Most XML manuals, however, provide no guidelines for representing semantics. Following is an excerpt from one of the proposed standards for representing *resources* in XML:

> A resource can be anything that has identity. Familiar examples include an electronic document, an image, a service (e.g., "today's weather report for Los Angeles"), and a collection of other resources. Not all resources are network "retrievable"; e.g., human beings, corporations, and bound books in a library can also be considered resources. (Berners-Lee, et al. 1998)

In that report, an electronic document is considered familiar, but human beings are unfamiliar "resources" mentioned only as an afterthought. Yet without the people, the document and its contents have no meaning.

Many of the ontologies for web objects ignore physical objects, processes, people, and their intentions. A typical example is SHOE (Simple HTML Ontology Extensions), which has only four basic categories: String, Number, Date, and Truth (Heflin et al. 1999). Those four categories, which are needed to describe the syntax of web data, cannot by themselves describe the semantics. Strings contain characters that represent statements that describe the world; numbers count and measure things; dates are time units tied to the rotation of the earth; and truth is a metalanguage term about the correspondence between a statement and the world. Those categories can only be defined in terms of the world, the people in the world, and the languages people use to talk about the world. Without such definitions, the categories are meaningless tags that confer no meaning upon the data they are attached to.

In discussing the Resource Description Framework (RDF), which is based on the XML facilities, Bray (1998) presented a broader view of the kinds of categories that web-based metadata should represent:

> It seems unlikely that one PropertyType standing by itself is apt to be very useful. It is expected that these will come in packages; for example, a set of basic bibliographic PropertyTypes like Author, Title, Date, and so on. Then a more elaborate set from OCLC, and a competing one from the Library of Congress. These packages are called **Vocabularies**; it's easy to imagine PropertyType vocabularies describing books, videos, pizza joints, fine wines, mutual funds, and many other species of Web wildlife.

This is a good statement of one issue, but it raises other issues: How are the packages related to one another? How is the Date property of the OCLC package related to the Vintage property of a wine package? Can packages inherit type definitions from other packages? If two packages are competing, is there any way to define conversion rules for translating or redefining the types of one in terms of another? A human reader may know that a wine vintage can be compared to an OCLC date, but without a formal definition, the computer cannot.

Ironically, the computer networks that make it easier to transmit data have made it more difficult to share data. In continuing his discussion, Bray raised further issues:

> Nobody thinks that everyone will use the same vocabulary (nor should they), but with RDF we can have a marketplace in vocabularies. Anyone can invent them, advertise them, and sell them. The good (or best-marketed) ones will survive and prosper. Probably, most niches of information will come to be dominated by a small number of vocabularies, the way that library catalogues are today.

There are already thousands, if not millions of competing vocabularies. The tables and fields of every database and the lists of items in every product catalog for every business in the world constitute incompatible vocabularies. When product catalogs were distributed on paper, any engineer or contractor could read

the catalogs from different vendors and compare the specifications. But minor variations in the terminology of computerized catalogs can make it impossible for a computer system to compare components from different vendors.

By standardizing the notations, XML and RDF take an important first step, but that step is insufficient for data sharing without some way of comparing, relating, and translating the vocabularies. Phipps (2000) warned that standardizing the vocabularies may create even more difficulties "by hiding complexities behind superficial agreements":

> To connect from the heart of my e-business to the heart of yours would be impossibly expensive in shared systems without XML, but even with it the system analysis needed to create the translation is a significant task. We should not assume that XML is a panacea or that the standardization of vocabularies will automatically bring interoperability. XML provides us with a medium to express our understanding of the meaning of data, but we still have to discern realities and differences of meanings when we exchange data.

More important than standardizing vocabularies is the development of methods for defining and translating vocabularies. To have a sound semantics and pragmatics, those methods must relate the terms in the vocabularies to the things they refer to and to the people who use them to communicate information about those things.

The purpose of this paper is to analyze the differences of meaning, to explore their implications for web-based metadata, and to show how the methods of logic and ontology can be used to define, relate, and translate signs from one vocabulary to another. Among the methods discussed in this paper are Peirce's systems of logic, ontology, and semiotics, which are presented in more detail in the book *Knowledge Representation*[1] by Sowa (2000).

2 Signs of Signs

Metalanguage consists of signs that signify something about other signs, but what they signify depends on what relationships those signs have to each other, to the entities they represent, and to the agents who use those signs to communicate with other agents. Figure 1 shows the basic relationships in a *meaning triangle* (Ogden and Richards 1923). On the lower left is an icon that resembles a cat named Yojo. On the right is a printed symbol that represents his name. The cloud on the top gives an impression of the neural excitation induced by light rays bouncing off Yojo and his surroundings. That excitation, called a *concept*, is the *mediator* that relates the symbol to its object.

The triangle in Figure 1 has a long history. Aristotle distinguished objects, the words that refer to them, and the corresponding experiences in the *psychê*. Frege and Peirce adopted that three-way distinction from Aristotle and used it

[1] See URL http://www.bestweb.net/.sowa/krbook/

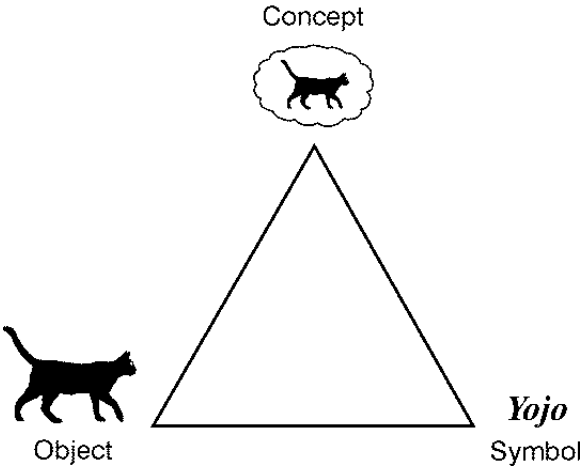


Fig. 1. The meaning triangle

as the semantic foundation for their systems of logic. Frege's terms for the three vertices of the triangle were *Zeichen* (sign) for the symbol, *Sinn* (sense) for the concept, and *Bedeutung* (reference) for the object. As an example, Frege cited the terms *morning star* and *evening star*. Both terms refer to same object, the planet Venus, but their senses are very different: one means a star seen in the morning, and the other means a star seen in the evening. Following is Peirce's definition of *sign*:

> A sign, or *representamen*, is something which stands to somebody for something in some respect or capacity. It addresses somebody, that is, creates in the mind of that person an equivalent sign, or perhaps a more developed sign. That sign which it creates I call the *interpretant* of the first sign. The sign stands for something, its *object*. It stands for that object, not in all respects, but in reference to a sort of idea, which I have sometimes called the *ground* of the representamen. (CP 2.228)

The terms *morning star* and *evening star* are distinct signs that create different concepts or interpretants in the mind of the listener. Both concepts stand for the same object, but in respect to a different ground, which depends on the time of the observation.

Aristotle observed that symbols could symbolize other symbols, as "written words are symbols of the spoken." Frege said that his logic could be used as a language to talk about the logic itself. But Peirce went further than either of them in recognizing that multiple triangles could be linked together in different ways by attaching a vertex of one to a vertex of another. By stacking another triangle on top, Figure 2 represents the concept of representing an object by a concept. The upper triangle relates the cloud that suggests the concept of Yojo to

the symbol [Cat: Yojo], which is a printable symbol for the more elusive neural excitation. At the very top is a cloud for the neural excitation that occurs when some person recognizes that Yojo is being represented by a printed symbol.

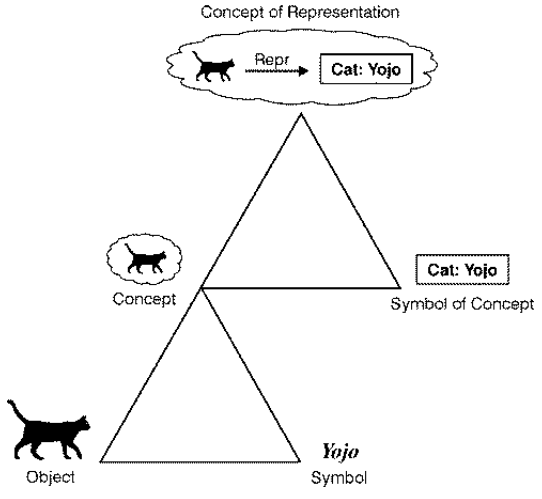


Fig. 2. Concept of representing an object by a concept

Meaning triangles can be linked side by side to represent signs of signs of signs. On the left of Figure 3 is the triangle of Figure 1, which relates Yojo to his name. The middle triangle relates the name *Yojo* to the quoted string "Yojo". The rightmost triangle relates that character string to its encoding as a bit string 0x596F6A6F. In each of the three triangles, the symbol is related to its object by a different metalevel process: naming, quoting, or representing. At the top of each triangle, the clouds that represent the unobservable neural excitations have been replaced by *concept nodes* that serve as printable symbols of those excitations. The concept node [Cat: Yojo] is linked by the *conceptual relation node* (Name) to a node for the concept of the name [Word: "Yojo"], which is linked by the conceptual relation node (Repr) to a node for the concept of the character string itself [String: 'Yojo']. The resulting combination of concept and relation nodes is an example of a *conceptual graph* (CG).

To deal with meaning, semiotics must go beyond relationships between signs to the relationships of signs, the world, and the agents who observe and act upon the world. Symbols are highly evolved signs that are related to actual objects by previously established conventions. People agree to those conventions by relating the symbols to more primitive signs, such as *icons*, which signify their objects by some structural similarity, and *indices*, which signify their objects by pointing to them. All these signs can be related to one another by linking series or even arrays of triangles. Additional triangles could show how a name is related to the

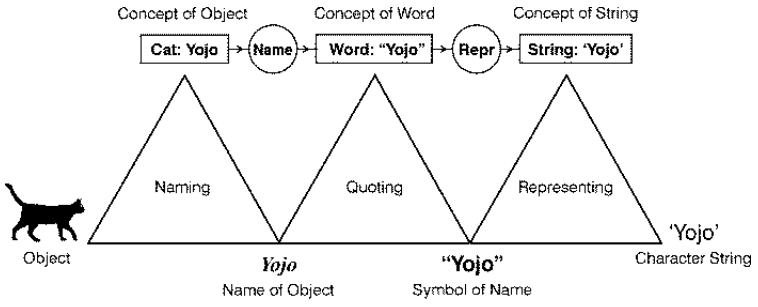


Fig. 3. Object, name of object, symbol of name, and character string

person who assigns the name, to the reason for giving an object one name rather than another, or to an index that points to some location where the object may be found.

Different kinds of applications require different levels of detail in the metadata. For information retrieval (IR), a simple string search can often find a web page with the desired information. To find information about Yojo the cat, it could search for the strings "Yojo" and "cat"; to find information about Queequeg's ebony idol in the novel *Moby Dick*, it could search for the strings "Yojo" and "Queequeg". IR systems depend on a human reader to decide which strings to search for and to interpret the results that are retrieved. Systems that go beyond simple search must be able to distinguish the physical object Yojo from an icon that resembles the object, the name of the object, and the character string that represents the name. Following is an interchange between a human user who asked a question and a computer system that did not make those distinctions:

```
Q: What is the largest state in the US?
A: Wyoming.
```

To answer questions about sizes, the computer would use the greater-than operator to compare numbers. When it applied that operator to the character strings, it found the last state in alphabetical order, which does not happen to be the largest state in either area or population. A loosely defined system of metadata may be adequate for finding information, but inadequate for any further processing. As Phipps observed, superficial agreements about vocabulary may hide complexities that make interoperability impossible.

3 Logic

The second branch of semiotics is semantics, or as Peirce called it, logic proper – the subject that studies what it means for a pattern of signs to represent a true proposition about the things the signs refer to. The first complete system of *first-order logic* (FOL) was the *Begriffsschrift* by Gottlob Frege (1879), who

developed a notation that no one else, not even his very few students, ever adopted. The second complete system was the algebraic notation for predicate calculus, independently developed by Charles Sanders Peirce (1883, 1885). With minor modifications, it became the most commonly used version of logic during the twentieth century. It is a much better notation than Frege's *Begriffsschrift*, but for many people, it is "too mathematical." The third complete system was Peirce's *existential graphs* of 1897, which he called his *chef d'oeuvre* – a strong claim by a man who invented the most widely used version of logic today.

With existential graphs, Peirce set out to determine the simplest, most primitive forms for expressing the elements of logic. Although he developed a graphical notation for expressing those forms, they can be expressed equally well in a natural language, an algebraic notation, or many different linear, graphical, or even spoken representations. The following table lists Peirce's five semantic primitives, each illustrated with an English example. Since these five elements are primitive, they cannot be formally defined in terms of anything more primitive; instead, the middle column of the table briefly states their "informal meaning."

Primitive	**Informal Meaning**	**English Example**
Existence	Something exists.	*There is a dog.*
Coreference	Something is the same as something.	*The dog is my pet.*
Relation	Something is related to something.	*The dog has fleas.*
Conjunction	A and B.	*The dog is running, and the dog is barking.*
Negation	Not A.	*The dog is not sleeping.*

Table 1. Five semantic primitives

The five primitives in Table 1 (cf. Section 1) are available in every natural language and in every version of first-order logic. They are called semantic primitives because they go beyond syntactic relations between signs to semantic relations between signs and the world. Any notation that is capable of expressing these five primitives in all possible combinations must include all of FOL as a subset. As an example, the `WHERE` clause of the SQL query language can express each of these primitives and combine them in all possible ways; therefore, first-order logic is a subset of SQL. Different languages may use different notations for representing the five primitives:

– *Existence.* In most natural languages, existence is implied by mentioning something. For emphasis, languages also provide an explicit *existential quantifier* such as the word *some*. In the algebraic notation for logic, existence may be expressed by an explicit symbol, such as $\exists$. In SQL, existence is stated implicitly by mentioning something or explicitly by using the keyword `EXISTS`.

- *Coreference.* To say that two different signs refer to the same thing, natural languages use a variety of methods, both explicit and implicit: pronouns, determiners, inflections, and forms of the verb *be*. Most linear notations for logic use variables and the equal sign, and graphic notations use connecting lines or ligatures. Like other linear notations, SQL uses variables and the equal sign.
- *Relation.* Content words in natural languages express some information about at least one entity, known as the *referent* of the word, but they may also relate or imply other entities as well. The verb *give*, for example, refers to an act of giving, but it also implies a giver, a gift, and a recipient. In SQL, relations are called *tables*.
- *Conjunction.* In both natural and artificial languages, conjunction may be expressed implicitly by making one statement after another or explicitly by a word like *and* or a symbol like $\wedge$. SQL uses the keyword `AND`.
- *Negation.* All natural languages and most versions of logic provide words, inflections, or symbols to express negation. The biggest variations from one language to another are in the methods for distinguishing the context or scope of what is negated from what is not negated. SQL uses the keyword `NOT` with parentheses to show scope.

Other logical operators can be defined in terms of these five primitives. Table 2 (cf. Section 2) shows three of the most common: the universal quantifier, implication, and disjunction. These operators do not qualify as semantic primitives because they are not as directly observable as the five in Table 1 (cf. Section 1). Seeing Yojo, for example, is evidence that some cat exists, but there is no way to perceive every cat. Seeing two things together is evidence of a conjunction, and not seeing something is evidence of a negation. But there is no direct way of perceiving an implication or a disjunction: *post hoc* does not imply *propter hoc*, and seeing one alternative of a disjunction does not indicate what other alternatives are possible. Although the three operators of Table 2 (cf. Section 2) can be defined in terms of the five primitives, any assertion they make about the world can only be verified indirectly and usually with less certainty than the basic primitives.

Operator	**English Example**	**Translation to Primitives**
Universal	*Every dog is barking.*	not(*there is a dog* and not(*it is barking*))
Implication	*If there is a dog, then it is barking.*	not(*there is a dog* and not(*it is barking*))
Disjunction	*A dog is barking, or a cat is eating.*	not(not(*a dog is barking*) and not(*a cat is eating*))

Table 2. Three defined logical operators

Instead of choosing existence and conjunction as primitives, Frege chose the universal and implication as primitives. Then he defined existence and conjunc-

tion in terms of his primitives. The result was not as readable as Peirce's algebraic notation, but it was semantically equivalent. Peirce's existential graphs (EGs) were also semantically equivalent to both of the other notations, but they had the simplest of all mappings to the five primitives. SQL also uses existence, conjunction, and negation as its three basic primitives, but it provides the keyword `OR` as well. SQL has no universal quantifier, which must be represented by a paraphrase of the form `NOT EXISTS... NOT`. To add logical operators to RDF, Berners-Lee (1999) proposed the tags <not> and <exists>, which can be combined with the implicit conjunction of RDF to define the operators of Table 2 (cf. Section 2).

To illustrate various notations for logic and their relationships to RDF, consider a typical sentence that might be used in a database specification: *Every human being has two distinct parents, who are also human beings.* Since this sentence introduces numbers and plurals, which go beyond the five primitives, start with the simpler sentence *Some human has a parent, who is also human.* Figure 4 shows an existential graph that represents the sentence.

EG for 'Some human has a human parent.'

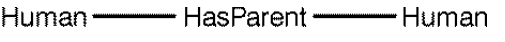


Fig. 4. EG for *Some human has a parent who is human*

In an existential graph, the words represent predicates or relations, and the bars represent existential quantifiers. The two bars in Figure 4 represent two individuals who are human, and the one on the left has the one on the right as a parent. In the algebraic notation, each bar would be assigned a variable, such as x or y, and an existential quantifier, represented by the symbol $\exists$. As a result, Figure 4 would map to the following formula:

```
(∃ x)(∃ y)(Human(x)  ∧  HasParent(x,y)  ∧  Human(y)).
```

The symbol $\wedge$ in the formula represents conjunction, which is implicit in the EG and RDF notations. Figure 4 could be represented by a *triple* in RDF: the first human could be treated as an RDF *resource*, the HasParent relation as an RDF *property type*, and the second human as an RDF *value*. The existence of the human on the left would be indicated by the proposed RDF quantifier <exists var="x">, and the one on the right by <exists var="y">.

The EG in Figure 5 would require an additional relation or property type before it could be represented in RDF. It represents the sentence *Some human has one parent who is human, another parent who is human, and the two are not identical.*

The bar that represents (∃ *x*) in Figure 4 is connected to both copies of the HasParent relation in Figure 5. Two bars represent each of the human parents. If they were connected, they would represent the same individual; but to represent

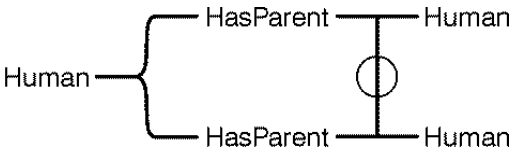


Fig. 5. EG for *Some human has two distinct human parents*

distinct individuals, the connection must be negated. In existential graphs, Peirce used an oval to indicate negation; in Figure 5, the oval negates part of the connecting bar. In the algebraic notation, Figure 5 would be represented by the following formula:

```
(∃ x)(∃ y)(∃ z)
   (Human(x) ∧ HasParent(x,y) ∧ Human(y)
     ∧ HasParent(x,z) ∧ Human(z) ∧ y≠z).
```

The inequality $y \neq z$ corresponds to the negated connecting bar in Figure 5. In EGs, the bar that represents a variable also represents coreference, and its negation represents inequality. Notations that have variables, such as predicate calculus, SQL, and RDF, must also have a coreference operator, such as = and its negation ≠. With new property types for Equal and NotEqual, Figure 5 could be represented in RDF by three existential quantifiers linked together by three RDF triples.

The small oval in Figure 5 is sufficient to negate the connection between the bar for one parent y and the bar for the other parent z, but an oval can be made as large as necessary to show the scope of negation. To show a universal quantifier, Table 2 (cf. Section 2) shows that two negations are necessary, which are represented by a pair of large ovals in Figure 6. Literally, the resulting graph may be read *It is false that there exists a human being who does not have two distinct parents*. It corresponds to the following formula:

```
∼(∃ x)(Human(x) ∧ ∼(∃ y)(∃ z)
   (HasParent(x,y) ∧ Human(y)
     ∧ HasParent(x,z) ∧ Human(z) ∧ y≠z)).
```

Two copies of the proposed RDF tag <not> and its ending tag </not> could be nested to provide the equivalent of the two nested ovals in Figure 6. To make RDF equivalent to existential graphs, however, new RDF rules would be needed to restrict the scope of the quantified variables to the contexts enclosed by the tags <not> and </not>.

As Table 2 (cf. Section 2) illustrates, a pair of negations can represent either a universal quantifier or an implication. The EG in Figure 6 may be read in either way. If the two ovals are considered an implication, Figure 6 could be read *If there exists a human, then that human has a parent who is human and another*

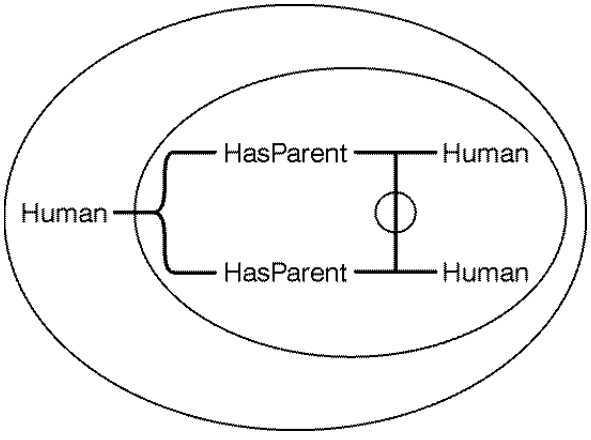


Fig. 6. EG for *Every human has two distinct human parents*

parent who is human and the two parents are distinct. Another option is to read an existential quantifier nested between two ovals as the universal quantifier ", which is expressed by the English word *every.* Then Figure 6 could be read *Every human has a parent who is human and another parent who is human and the two parents are distinct.* By using the defined operators of Table 2 (cf. Section 2), the formula could be rewritten in a form that shows the universal quantifier ∀ and the implication ⊃ explicitly:

```
(∀ x)(Human(x)  ⊃  (∃ y)(∃ z)
   (HasParent(x,y) ∧ Human(y)
      ∧ HasParent(x,z) ∧ Human(z) ∧ y≠ z)).
```

In English, this formula may be read *For every x, if x is human, then there exist a y and a z such that x has the human y as parent, x has the human z as parent, and y and z are not the same individual.*

With their minimal number of operators, Peirce's EGs have a single canonical form instead of the multiple synonymous sentences in languages with more built-in operators, such as English and predicate calculus. That property, which is sometimes an advantage, can be a disadvantage when the most natural or convenient translation is not obvious. Conceptual graphs (Sowa 1984, 2000) are a graphic notation for logic based on existential graphs, but with extended features that support more direct translations to natural languages. Figure 7 shows a conceptual graph that corresponds to the existential graph in Figure 6.

Logically, the CG in Figure 7 is semantically equivalent to the EG in Figure 6. To indicate the intended reading, the CG has two boxes explicitly labeled *If* and *Then* instead of the EG ovals. Unlike EGs, which use a bar to represent existential quantification, coreference, and connections between relations, those three functions are distinguished in CGs: boxes, called *concept nodes*, represent

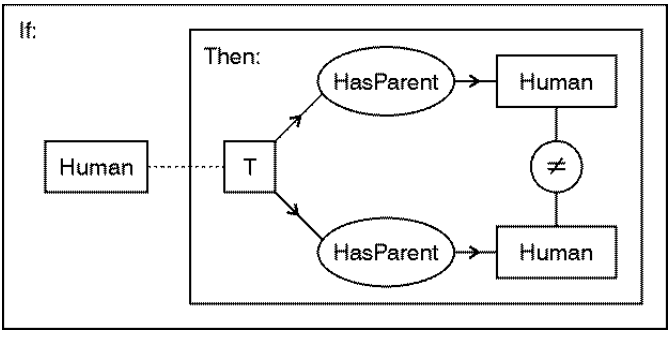


Fig. 7. CG for *If there is a human, then he or she has two distinct human parents*

quantification; dotted lines represent coreference; and solid lines represent connections between the concept nodes and the relation nodes. The node [T] in the `Then` context, which is coreferent with the node [`Human`] in the `If` context, corresponds to a pronoun, such as *he*, *she*, or *it*. Altogether, Figure 7 may be read *If there is a human, then he or she has two distinct human parents.* To improve the readability of logic expressed in RDF, Berners-Lee also proposed the tags <`if`> and <`then`> as synonyms for <`not`>.

Natural languages have a variety of quantifiers, such as the words *every, some,* or *all*, the numbers *two, seventeen,* or *half*, and the phrases *more than six* or *at least as many.* Those *generalized quantifiers* can be defined in logic by adding Peano's axioms to define numbers and set theory to define collections, but it is convenient to have such quantifiers built into the notation. In CGs, the default quantifier is the existential ∃ , which is normally represented by a blank, but concept nodes may also contain defined quantifiers, such as the symbol `"` or `@every` to represent the English word *every.* The CG in Figure 8 is equivalent to Figure 7 by the definition of the quantifier ∀ . It maps to the following formula in *typed predicate calculus*:

```
(∀ x:Human)(∃  y,z:Human)
   (HasParent(x,y) ∧ HasParent(x,z) ∧ y≠z).
```

In typed logic, monadic predicates such as `Human(`*x*`)` are replaced by *type labels* associated with the variables. The typed formula is more concise, but logically equivalent to the untyped formulas that represent the EG of Figure 6.

Figure 8 could be represented in RDF with the proposed <`forall`> quantifier, but CGs also support other generalized quantifiers that have not yet been considered for RDF. As an example, Figure 9 simplifies Figure 8 by introducing the *generic plural* symbol {`*`} to represent a set and the number 2 to represent its count or *cardinality.* The resulting CG can be mapped to the following formula:

```
(∀ x:Human)(∃ s:Set)(∀ y ∈ s)
   (Count(s,2) ∧ HasParent(x,y) ∧ Human(y)).
```

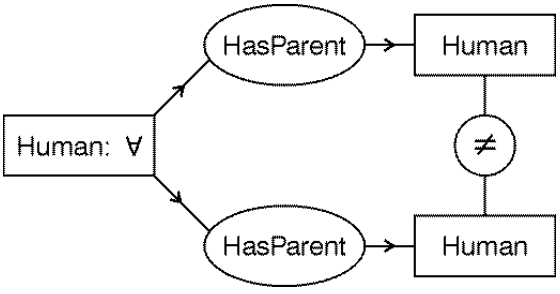


Fig. 8. CG for *Every human has two distinct human parents*

This formula may be read *For every x of type Human, there exists an s of type Set such that for every y in s, the count of s is 2, x has y as parent, and y is human.* The generalized quantifier {*}@2 in the CG maps to two quantified variables in predicate calculus: a variable *s* that ranges over sets and a variable *y* that ranges over the elements of the set *s*.

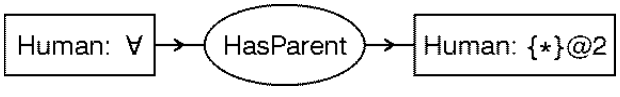


Fig. 9. CG for *Every human has a set of two human parents*

The CG in Figure 9 is closer to English, but it still isn't quite as simple as the more natural sentence *Every human has two parents.* That sentence could be expressed directly by the CG in Figure 10.

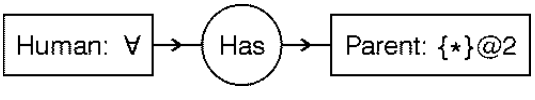


Fig. 10. CG for *Every human has two parents*

In English, the HasParent relation is normally expressed by the verb *have* combined with the noun *parent.* That noun belongs to a large class of *role words*, such as *spouse, pilot, lawyer, assistant, pet, weed, crop, entrance, obstacle,* or *facility.* Syntactically, those words resemble nouns like *man, woman, dog,* or *tree*; but semantically, they imply some relationships to external entities. In the ontology of the book *Knowledge Representation* (Sowa 2000), the primitive

relation Has is used to form dyadic relations by combining with concept types that represent roles. Figure 11 shows how the HasParent relation is defined in terms of the relation Has and the role type Parent.

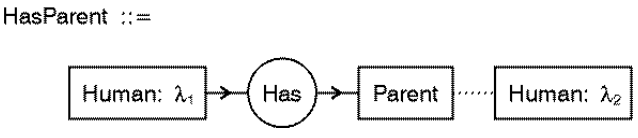


Fig. 11. Definition of the HasParent relation

Figure 11 defines the HasParent relation as a synonym for a conceptual graph that has two concepts designated as *formal parameters*: the symbol λ_1 marks the first parameter as a human who has a parent that is coreferent with another human, marked as the second parameter by the symbol λ_2. It may be read *The HasParent relation is defined as a relation between two humans; the first has a parent who is the second*. With this definition, Figure 10 can be mapped to or from Figure 9. With appropriate definitions of sets and numbers, Figure 9 can be mapped to or from Figure 8, which can be mapped to or from the existential graph or any of the algebraic formulas in typed or untyped predicate calculus. To support equivalent definitions, RDF would require a tag such as <`lambda`> or <`parm`> to mark a formal parameter.

In addition to the semantic primitives of Table 1 (cf. Section 1), Peirce distinguished a context-dependent primitive, which he called an *indexical*. In natural languages, indexicals are represented by pronouns, by *deictic words* such as *this* and *that*, and by noun phrases marked by the definite article *the*. In conceptual graphs, indexicals are marked by the # symbol. The concept [`Cat`], for example, represents some unspecified cat that happens to exist; but the concept [`Cat: #`] represents the cat that was most recently mentioned in the current context. Peirce observed that proper names are also indexicals. Within the context of this article, the name *Yojo* may refer to a cat or to Queequeg's ebony idol. On the Internet, it also refers to some Japanese people, to others who have adopted that word as a nickname, and to an organization of young journalists. The ambiguity of names and their context dependencies are major concerns addressed by the naming conventions of the Internet. Those conventions are semiotic features that can be represented by metalevel types and relations in conceptual graphs and other logic-based notations.

In summary, the algebraic notation for logic, which is popular with mathematicians, is only one of an open-ended number of semantically equivalent notations. The five semantic primitives of Table 1 (cf. Section 1) and the mechanisms for defining the other operators of first-order logic can be adapted to a wide variety of notations, including natural languages and the web-oriented notations of XML and RDF.

1. Logic can be and has been represented in a wide variety of graphic and linear notations of varying degrees of readability and suitability for different kinds of applications. EGs and CGs are graphic examples, and the Knowledge Interchange Format (KIF) is an equivalent linear form. Other linear versions can be written with the syntactic conventions of SQL, RDF, or even natural languages.
2. For better readability, it is possible to represent the logical operators in *controlled natural languages*, which use a subset of the syntax and vocabulary of natural languages. Although the task of translating unrestricted natural languages to any formal notation is still a research problem, it is much easier to translate conceptual graphs and other formal notations to a stylized version of natural language, such as the English readings given for Figures 4 through 11.
3. Besides notation, logic has rules of definition and inference, which allow one representation to be translated to or from other synonymous representations. Figures 6 through 10 can be translated automatically to or from one another or the equivalent formulas in predicate calculus – provided that an appropriate ontology has been defined. With its formally defined semantics, logic provides the means for generating semantically equivalent translations to and from other languages with radically different syntax.

For better readability, any of the logical notations mentioned in this section can be translated to controlled natural languages. One important application, for example, is the generation of comments and help messages automatically from the implementation. Such translations would guarantee that the comments and help would always be up to date, consistent with the implementation, and immediately available in every supported national language.

4 Combining Logic with Ontology

Pure logic is ontologically neutral. It makes no presuppositions about what exists or may exist in any domain or any language for talking about the domain. To represent knowledge about a specific domain, it must be supplemented with an ontology that defines the categories of things in that domain and the terms that people use to talk about them. The ontology defines the words of a natural language, the predicates of predicate calculus, the concept and relation types of conceptual graphs, the classes of an object-oriented language, or the tables and fields of a relational database. To illustrate the issues of defining an ontology, consider the conceptual graph in Figure 12, which represents the sentence *Yojo is chasing a mouse.*

Figure 12 uses three concepts and two conceptual relations. The concept [`Cat: Yojo`] represents a cat named Yojo; [`Chase`] represents an instance of chasing; and [`Mouse`] represents a mouse. The conceptual relation `(Agnt)` indicates that Yojo is the *agent* of chasing, and `(Thme)` indicates that the mouse is the

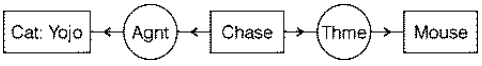


Fig. 12. CG for *Yojo is chasing a mouse*

theme or the one that is being chased. The CG is logically equivalent to the following formula in typed predicate calculus:

```
(∃ x:Cat)(∃ y:Chase)(∃ z:Mouse)
   (name(x,"Yojo") ∧ agnt(y,x) ∧ thme(y,z)).
```

This formula and the CG in Figure 12 introduce several ontological assumptions: there exist entities of types Cat, Chase, and Mouse; some entities have character strings as names; and Chase can be linked to concepts of other entities by relations of type Agent and Theme.

The representation of actions by distinct concepts follows Peirce's ontology, which represents an action such as chasing with three distinct entities: the one that is chasing, the one that is being chased, and the act of chasing itself. The relations `(Agnt)` and `(Thme)` are examples of the *case relations* or *thematic roles* of linguistics. Instead of Peirce's ontology, which is also called *event semantics*, some logicians would represent the verb *is chasing* by a single predicate, such as `chases`:

```
(∃ x:Cat)(∃ y:Mouse)
   (name(x,"Yojo") ∧ chases(x,y)).
```

The ontology of this formula could also be used in a conceptual graph:

```
[Cat: Yojo] (Chases) [Mouse].
```

This CG, which is written in the linear notation for CGs, can be translated to Figure 12 by defining the relation `(Chases)` in terms of the concept [`Chase`]:

```
Chases ::=
[Animate: λ1] (Agnt) → [Chase] → (Thme) ← [MobileEntity: λ2].
```

With this definition of `(Chases)`, the ontology of the previous CG can be translated to or from the ontology assumed in Figure 12.

Although the `Chases` relation allows shorter graphs and formulas than the concept [`Chase`], it introduces other complexities into the ontology. A general representation for tenses and modality, for example, would require a proliferation of relation types, such as `HasChased`, `WillChase`, and `MustHaveBeenChasing`. Furthermore, the dyadic relation `chases(`*`x,y`*`)` makes no provision for attaching adverbs and other modifiers to the verb. Figure 13 takes advantage of the more general representation to define the concept type Chase in terms of a graph for

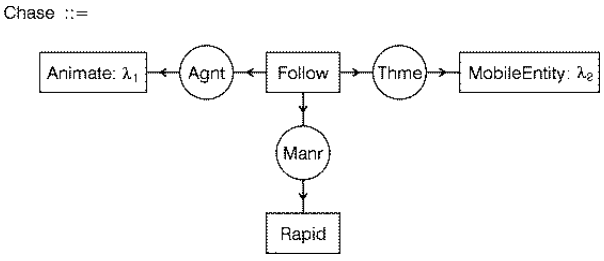


Fig. 13. Definition of the concept type Chase

an animate agent (parameter #1) that is following a mobile entity (parameter #2) in a rapid manner.

Figure 13 is only a partial definition because it represents a necessary, but not a sufficient condition. Runners in a race, for example, might be following one another rapidly, but only because they are pursuing a common goal. A more complete definition must include the purpose, which might be different for different senses of the word *chase*. Figure 14 defines one sense, called ChaseHunt, in which the purpose of the agent is to catch the mobile entity that is being chased.

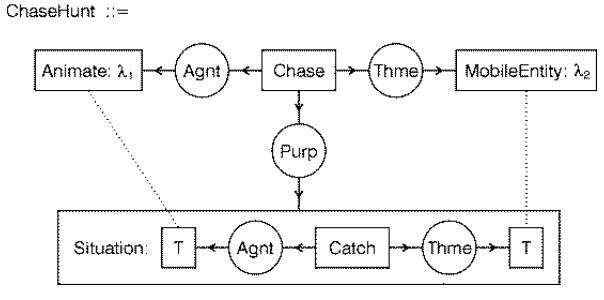


Fig. 14. Definition of the concept type ChaseHunt

In Figure 14, the purpose relation (`Purp`) links the action to a situation that would occur upon the successful completion of the chase. According to Peirce, purpose is a triadic relation, of which two arguments are shown explicitly in Figure 14. The implicit third argument is the agent of Chase, whose intention is to bring about the desired situation. That situation is nested inside a context box because its intentional status is different from the context of the act of chasing. If the chase is unsuccessful, the act of catching might never occur. Figure 15 defines another concept type ChaseAway, in which the agent's purpose is not to catch the mobile entity, but to cause it to leave its current location.

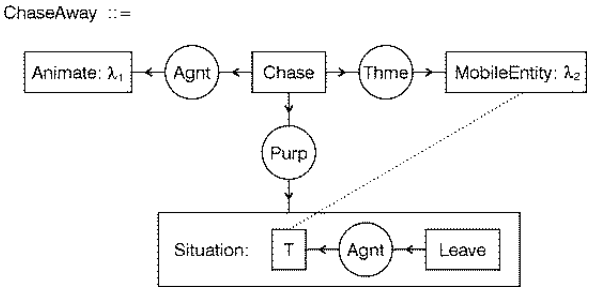


Fig. 15. Definition of the concept type ChaseAway

The ontology of situations and their representation in contexts is based on Peirce's logic combined with ideas developed in artificial intelligence, linguistics, philosophy, and logic over the past 40 years (Sowa 2000). A context box may enclose modal or intentional situations, as in Figures 14 and 15, or it may enclose temporally or spatially separated parts of a larger situation. In Figure 16, the large situation with its sequence of nested situations represents the following passage in English:

> At 10:17 UTC, there was a situation involving a cat named Yojo and a mouse. Yojo chased the mouse. Then he caught the mouse. Then he ate the head of the mouse.

These sentences show how indexicals are used to make context-dependent references. When new entities are first mentioned, they are introduced with the indefinite article, as in the phrases *a situation*, *a cat named Yojo*, and *a mouse*. The two middle sentences refer to the mouse with the definite article *the* and to the cat with the name *Yojo* or the pronoun *he*. In the last sentence, the head of the mouse, which had not been mentioned explicitly, is marked with the definite article because the introduction of the mouse implicitly introduces all of its expected parts. In Figure 16, the indexicals are marked with the # symbol: the pronoun *he* is represented as #he, and the definite article *the* is represented with the # symbol by itself.

The large context box of Figure 16 encloses the entire situation, which occurred at the point in time (PTim) of 10:17 UCT. It contains concept nodes that represent the cat Yojo, the mouse, and three nested situations connected by the (Next) relation. Before that CG can be translated to predicate calculus, the indexicals must be resolved to links or labels that explicitly show the coreferences. To avoid multiple line crossings, Figure 17 introduces the *coreference labels* `*x` for Yojo and `*y` for the mouse. Subsequent references use the same labels, but with the prefix `?` in the *bound* occcurrences of [`?x`] for Yojo and [`?y`] for the mouse. The # symbol in the concept [`Head: #`] of Figure 16 is erased in Figure 17, since the head of a normal mouse is uniquely determined when the mouse itself is identified.

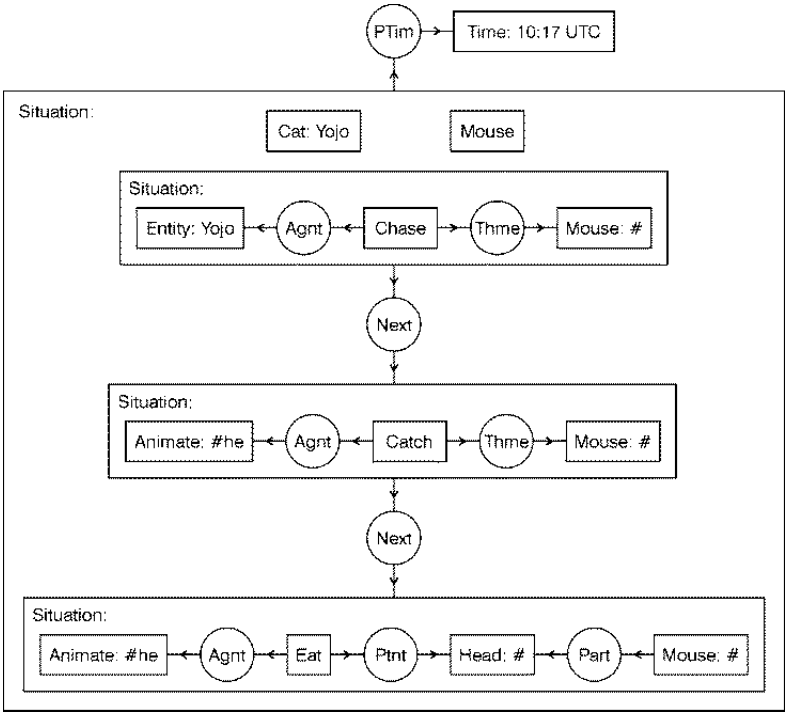


Fig. 16. Nested situations with unresolved indexicals

After the indexicals have been resolved to coreference labels, Figure 17 can be translated to the following formula in typed predicate calculus:

```
(∃ s1:Situation)(pTim(s1,"10:17 UTC")
    ∧ dscr(s1,
      (∃ s1,s2,s3:Situation)(∃ x:Cat)(∃ y:Mouse)(name(x,"Yojo")
        ∧ dscr(s2, (∃ u:Chase)(agnt(u,x) ∧ thme(u,y)))
        ∧ dscr(s3, (∃ v:Catch)(agnt(v,x) ∧ thme(v,y)))
        ∧ dscr(s4,
          (∃ w:Eat)(∃ z:Head)(agnt(w,x) ∧ ptnt(w,z) ∧ part(y,w)))
        ∧  next(s2,s3) ∧  next(s3,s4)))).
```

The *description* predicate `dscr(`*s*`,`*p*`)`, which corresponds to the context boxes of Figure 16, is a metalevel relation between a situation s and a proposition p that describes s. Figure 17 or its translation to predicate calculus could also be paraphrased in a version of controlled English that uses variables to show coreference explicitly: *At 10:17 UTC, there was a situation s involving a cat x named Yojo and a mouse y. In the situation s, x chased y; then x caught y; then x ate the head of y.*

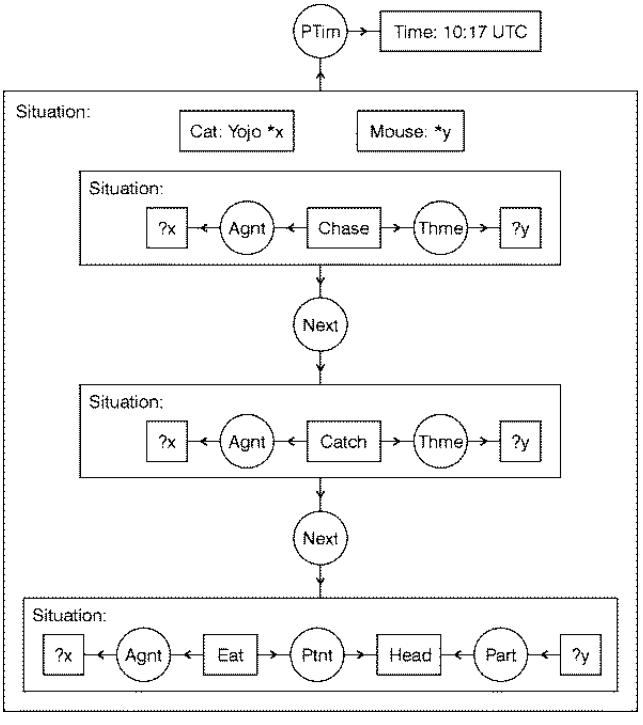


Fig. 17. Nested situations with indexicals resolved

The contexts of conceptual graphs are based on Peirce's logic of existential graphs and his theory of indexicals. Yet the CG contexts happen to be isomorphic to the similarly nested *discourse representation structures* (DRS), which Hans Kamp (1981a,b) developed for representing and resolving indexicals in natural languages. When Kamp published his first version of DRS, he was not aware of Peirce's graphs. When Sowa (1984) published his book on conceptual graphs, he was not aware of Kamp's work. Yet the independently developed theories converged on semantically equivalent representations; therefore, Sowa and Way (1986) were able to apply Kamp's techniques to conceptual graphs. Such convergence is common in science; Peirce and Frege, for example, started from very different assumptions and converged on equivalent semantics for FOL, which 120 years later is still the most widely used version of logic. Independently developed, but convergent theories that stand the test of time are a more reliable basis for standards than the consensus of a committee.

5 Extracting Logic from Language

Since all combinations of the five primitives of Table 1 can be expressed in every natural language, it is possible to represent first-order logic in a subset of any

natural language. Such a subset, called a *stylized* or *controlled* NL, can be read by anyone who can read the unrestricted NL. As examples, the English paraphrases of the CGs and formulas in this article represent a version of controlled English. With an appropriate selection of syntax rules, that subset could be formalized as a representation of FOL that would be semantically equivalent to any of the common notations for logic.

For most people, no training is needed to read a controlled NL, but some training is needed to write it. For computers, it is easy to translate a controlled NL to or from logic, but fully automated understanding of unrestricted NL is still an unsolved research problem. To provide semiautomated tools for analyzing unrestricted language, Doug Skuce (1995, 1998, 2000) has designed an evolving series of *knowledge extraction* (KE) systems, which he called CODE, IKARUS, and DocKMan (Document-based Knowledge Management). The KE tools use a version of controlled English called ClearTalk, which is intelligible to both people and computers. As input, the KE tools take documents written in unrestricted NL, but they require assistance from a human editor to generate ClearTalk as output. Once the ClearTalk has been edited and approved, further processing by the KE tools is fully automated. The ClearTalk statements can either be stored in a knowledge base or be written as annotations to the original documents. Because of the way they're generated, the comments that people read are guaranteed to be logically equivalent to the computer implementation.

The oldest logic patterns expressed in controlled natural language are the four types of statements used in Aristotle's system of syllogisms. Each syllogistic rule combines a *major premise* and a *minor premise* to draw a *conclusion*. Following are examples of the four sentence patterns:

1. *Universal affirmative*. Every employee is a person.
2. *Particular affirmative*. Some employees are customers.
3. *Universal negative*. No employee is a competitor.
4. *Particular negative*. Some customers are not employees.

These patterns and the syllogisms based on them are used in many controlled language systems, including ClearTalk. For inheritance in expert systems and object-oriented systems, the major premise is a universal affirmative statement with the verb *is*, and the minor premise is either a universal affirmative or a particular affirmative statement with *is*, *has*, or other verbs. For database and knowledge base constraints, the major premise is a universal negative statement that prohibits undesirable conjunctions, such as employee and competitor.

Other important logic patterns are the if-then rules used in expert systems. In some rule-based systems, the controlled language is about as English-like as COBOL, but others are much more natural. Attempto Controlled English (Fuchs et al. 1998; Schwitter 1998) is an example of a rich, but unambiguous language that uses a version of Kamp's theory for resolving indexicals. Following are two ACE rules used to specify operating procedures for a library database:

```
If a copy of a book is checked out to a borrower
   and a staff member returns the copy
then the copy is available.

If a staff member adds a copy of a book to the library
   and no catalog entry of the book exists
then the staff member creates a catalog entry
        that contains the author name of the book
           and the title of the book
           and the subject area of the book
   and the staff member enters the id of the copy
   and the copy is available.
```

Rules like these are translated automatically to the *Horn-clause subset* of FOL, which is the basis for Prolog and many expert system languages. The subset of FOL consisting of Horn-clause rules plus Aristotelian syllogisms can be executed efficiently, but it is powerful enough to specify a Turing machine.

For database queries and constraints, natural language statements with the full expressive power of FOL can be translated to SQL. Although many NL query systems have been developed, none of them have yet become commercially successful. The major stumbling block is the amount of effort required to define the vocabulary terms and map them to appropriate fields of the database. But if KE tools are used to design the database, the vocabulary needed for the query system can be generated as a by-product of the design process. As an example, the RÉCIT system (Rassinoux 1994; Rassinoux et al. 1998) uses KE tools to extract knowledge from medical documents written in English, French, or German and translates the results to a language-independent representation in conceptual graphs. The knowledge extraction process defines the appropriate vocabulary, specifies the database design, and adds new information to the database. The vocabulary generated by the KE process is sufficient for end users to ask questions and get answers in any of the three languages.

Design and specification languages have multiple metalevels. As an example, the Unified Modeling Language has four levels: the metametalanguage defines the syntax and semantics of the UML notations; the metalanguage defines the general-purpose UML types; a systems analyst defines application types as instances of the UML types; finally, the working data of an application program consists of instances of the application types. To provide a unified view of all these levels, Olivier Gerbé and his colleagues at the DMR Consulting Group implemented design tools that use conceptual graphs as the representation language at every level (Gerbé et al. 1995, 1996, 1997, 1998, 2000). For his PhD dissertation, Gerbé developed an ontology for using CGs as the metametalanguage for defining CGs themselves. He also applied it to other notations, including UML and the Common KADS system for designing expert systems. Using that theory, Gerbé and his colleagues developed the Method Repository System as an authoring environment for editing, storing, and displaying the methods used by the DMR consultants. Internally, the knowledge base is stored in conceptual graphs, but externally, the graphs can be translated to web pages in either

English or French. About 200 business processes have been modeled in a total of 80,000 CGs. Since DMR is a Canadian company, the language-independent nature of CGs is important because it allows the specifications to be stored in the neutral CG form. Then any manager, systems analyst, or programmer can read them in his or her native language.

Translating an informal diagram to a formal notation of any kind is as difficult as translating unrestricted NL to executable programs. But it is much easier to translate a formal representation in any version of logic to controlled natural languages, to various kinds of graphics, and to executable specifications. Walling Cyre and his students have developed KE tools for mapping both the text and the diagrams from patent applications and similar documents to conceptual graphs (Cyre et al. 1994, 1997, 1999). Then they implemented a scripting language for translating the CGs to circuit diagrams, block diagrams, and other graphic depictions. Their tools can also translate CGs to VHDL, a hardware design language used to specify *very high speed integrated circuits* (VHSIC).

No single system discussed in this paper incorporates all the features desired in a KE system, but the critical research has been done, and the remaining work requires more development effort than pure research. Figure 18 shows the flow of information from documents to logic and then to documents or to various computational representations. The dotted arrow from documents to controlled languages requires human assistance. The solid arrows represent fully automated translations that have been implemented in one or more systems.

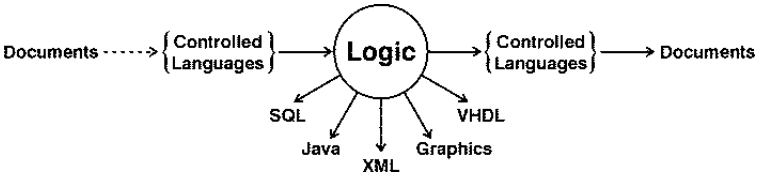


Fig. 18. Flow of information from documents to computer representations

For the KE tools, the unifying representation language is logic, which may be implemented in different subsets and notations for different tools. All the subsets, however, use the same vocabulary of natural-language terms, which map to the same ontology of concepts and relations. From the user's point of view, a KE system communicates in a subset of natural language, and the differences between tools appear to be task-related differences rather than differences in language.

References

1. Berners-Lee, Tim, R. Fielding, & L. Masinter, eds. (1998) *Uniform Resource Identifiers (URI): Generic Syntax*, Internet RFC 2396[2].
2. Berners-Lee, Tim (1999) *The Semantic Toolbox: Building Semantics on top of XML-RDF*, http://www.w3.org/DesignIssues/Toolbox.html[3].
3. Bray, Tim (1998) "RDF and Metadata," http://www.xml.com/xml/pub/98/06/rdf.html[4].
4. Bray, Tim, Dave Hollander, & Andrew Layman, eds. (1999) *Namespaces in XML*, W3 TR REC-xml-names.
5. Cyre, W. R., S. Balachandar, & A. Thakar (1994) "Knowledge visualization from conceptual structures," in Tepfenhart et al. (1994) *Conceptual Structures: Current Practice*, Lecture Notes in AI 835, Springer-Verlag, Berlin, pp. 275-292.
6. Cyre, W. R. (1997) "Capture, integration, and analysis of digital system requirements with conceptual graphs," *IEEE Trans. Knowledge and Data Engineering*, **9:1**, 8-23.
7. Cyre, W. R., J. Hess, A. Gunawan, & R. Sojitra (1999) "A Rapid Modeling Tool for Virtual Prototypes ," in *1999 Workshop on Rapid System Prototyping*, Clearwater, FL.
8. Frege, Gottlob (1879) *Begriffsschrift*, English translation in J. van Heijenoort, ed. (1967) *From Frege to Gödel*, Harvard University Press, Cambridge, MA, pp. 1-82.
9. Fuchs, Norbert E., Uta Schwertel, Rolf Schwitter (1998) "Attempto Controlled English – not just another logic specification language,"[5] *Proceedings LOPSTR'98*, Manchester.
10. Fuchs, Norbert E., Uta Schwertel, Rolf Schwitter (1999) Attempto Controlled English (ACE), Language Manual, Version 3.0,[6] Technical Report ifi-99.03, University of Zurich.
11. Genesereth, Michael R., & Richard Fikes, eds. (1998) *Knowledge Interchange Format (KIF)*[7], draft proposed American National Standard, NCITS.T2/98-004.
12. Gerbé, Olivier, & M. Perron (1995) "Presentation definition language using conceptual graphs,"[8] in G. Ellis, R. A. Levinson, & W. Rich, eds., *Conceptual Structures: Applications, Implementation, and Theory*, Lecture Notes in AI 954, Springer-Verlag, Berlin.
13. Gerbé, Olivier, B. Guay, & M. Perron (1996) "Using conceptual graphs for methods modeling,"[9] in P. W. Eklund, G. Ellis, & G. Mann, eds., *Conceptual Structures: Knowledge Representation as Interlingua*, Lecture Notes in AI 1115, Springer-Verlag, Berlin, pp. 161-174.
14. Gerbé, Olivier (1997)

[2] See URL http://www.ietf.org/rfc/rfc2396.txt
[3] See URL http://www.w3.org/DesignIssues/Toolbox.html
[4] See URL http://www.xml.com/xml/pub/98/06/rdf.html
[5] See URL http://www.ifi.unizh.ch/groups/CL/schwitter/lopstr98.ps
[6] See URL http://www.ifi.unizh.ch/techreports/TR-99/ifi-99.03.pdf
[7] See URL http://logic.stanford.edu/kif/dpans.html
[8] See URL http://www.hec.ca/pages/olivier.gerbe/publications/peirce95/gerbe1995.html
[9] See URL http://www.hec.ca/pages/olivier.gerbe/publications/iccs96/gerbe1996.html

"Conceptual graphs for corporate knowledge management,"[10] in D. Lukose, H. Delugach, M. Keeler, L. Searle, & J. Sowa, eds., *Conceptual Structures: Fulfilling Peirce's Dream*, Lecture Notes in AI 1257, Springer-Verlag, Berlin, pp. 474-488.

15. Gerbé, Olivier, R. Keller, & G. Mineau (1998) "Conceptual graphs for representing business processes in corporate memories,"[11] in M-L Mugnier & Michel Chein, eds., *Conceptual Structures: Theory, Tools, and Applications*, Lecture Notes in AI 1453, Springer-Verlag, Berlin, pp. 401-415.
16. Gerbé, Olivier, & Brigitte Kerhervé (1998) "Modeling and metamodeling requirements for knowledge management,"[12] in *Proc. of OOPSLA'98 Workshops*, Vancouver.
17. Gerbé, Olivier (2000) *Un Modèle uniforme pour la Modélisation et la Métamodélisation d'une Mémoire d'Entreprise*[13], PhD Dissertation, Département d'informatique et de recherche opérationelle, Université de Montréal.
18. Heflin, Jeff, James Hendler, & Sean Luke (1999) *SHOE: A Knowledge Representation Language for Internet Applications*, Technical Report CS-TR-4078 (UMIACS TR-99-71)[14], Dept. of Computer Science, University of Maryland at College Park.
19. Kamp, Hans (1981a) "Events, discourse representations, and temporal references," *Langages* **64**, 39-64.
20. Kamp, Hans (1981b) "A theory of truth and semantic representation," in *Formal Methods in the Study of Language*, ed. by J. A. G. Groenendijk, T. M. V. Janssen, & M. B. J. Stokhof, Mathematical Centre Tracts, Amsterdam, 277-322.
21. Kamp, Hans, & Uwe Reyle (1993) *From Discourse to Logic*, Kluwer, Dordrecht.
22. Lassila, Ora, & Ralph R. Swick, eds. (1999) *Resource Description Framework (RDF) Model and Syntax Specification*, http://www.w3.org/TR/REC-rdf-syntax[15].
23. Ogden, C. K., & I. A. Richards (1923) *The Meaning of Meaning*, Harcourt, Brace, and World, New York, 8th edition 1946.
24. Peirce, Charles Sanders (1885) "On the algebra of logic," *American Journal of Mathematics* **7**, 180-202.
25. Peirce, Charles Sanders (CP) *Collected Papers of C. S. Peirce* ed. by C. Hartshorne, P. Weiss, & A. Burks, 8 vols., Harvard University Press, Cambridge, MA, 1931-1958.
26. Phipps, Simon (2000) "Meaning, not Markup," XML Journal[16], vol. 1, no. 1, p. 66.
27. Rassinoux, Anne-Marie (1994) *Extraction et Représentation de la Connaissance tirée de Textes Médicaux*, Éditions Systèmes et Information, Geneva.
28. Rassinoux, Anne-Marie, Robert H. Baud, Christian Lovis, Judith C. Wagner, Jean-Raoul Scherrer (1998) "Tuning up conceptual graph representation for multilingual natural language processing in medicine," in M-L Mugnier & M. Chein, eds. (1998) *Conceptual Structures: Theory, Tools, and Applications*, Lecture Notes in AI 1453, Springer-Verlag, Berlin, pp. 390-397

[10] See URL http://www.hec.ca/pages/olivier.gerbe/publications/iccs97a/gerbe1997a.html
[11] See URL http://www.hec.ca/pages/olivier.gerbe/publications/iccs98/gerbe-iccs98.pdf
[12] See URL http://www.hec.ca/pages/olivier.gerbe/publications/oopsla98/gerbe-oopsla98.pdf
[13] See URL http://www.hec.ca/pages/olivier.gerbe/publications/gerbeThesis.pdf
[14] See URL http://www.cs.umd.edu/projects/plus/SHOE
[15] See URL http://www.w3.org/TR/REC-rdf-syntax
[16] See URL http://www.sys-con.com/xml/index2.html

29. St. Laurent, Simon (1999) *XML: A Primer*, second edition, M & T Books, Foster City, CA.
30. Saussure, Ferdinand de (1916) *Cours de Linguistique Générale*, translated by W. Baskin as *Course in General Linguistics*, Philosophical Library, New York, 1959.
31. Schwitter, Rolf (1998)
Kontrolliertes Englisch fúr Anforderungsspezifikationen,[17] Studentdruckerei, Zurich.
32. Skuce, Doug, & Timothy Lethbridge (1995)
"CODE4: A unified system for managing conceptual knowledge,"[18] *International J. of Human-Computer Studies*, **42** 413-451.
33. Skuce, Doug (1998) "Intelligent knowledge management: integrating documents, knowledge bases, and linguistic knowledge," in *Proceedings of KAW'98*, Calgary.
34. Skuce, Doug (2000) "Integrating web-based documents, shared knowledge bases, and information retrieval for user help," *Computational Intelligence* **16:1**.
35. Sowa, John F. (1984) *Conceptual Structures: Information Processing in Mind and Machine*, Addison-Wesley, Reading, MA.
36. Sowa, John F., ed. (1998) *Conceptual Graphs*, draft proposed American National Standard, NCITS.T2/98-003.
37. Sowa, John F. (2000) *Knowledge Representation: Logical, Philosophical, and Computational Foundations*, Brooks/Cole, Pacific Grove, CA.
38. Sowa, John F., & Eileen C. Way (1986) "Implementing a semantic interpreter using conceptual graphs," *IBM Journal of Research and Development* **30:1**, pp. 57-69.

[17] See URL http://www.ifi.unizh.ch/groups/CL/schwitter/DissBook.ps
[18] See URL http://www.csi.uottawa.ca/.tcl/papers/ijhcs95/contents.html

Pragmatically Yours,

Mary Keeler

Center for Advanced Research and Technology in the Arts and Humanities
University of Washington, Seattle
mkeeler@u.washington.edu

Abstract. Here is my attempt to find the roots of "the ontological problem" in AI research identified by Daniel Kayser at ICCS'98. I reconstruct just enough of C.S. Peirce's "scientific philosophy" to suggest how pragmatism responds to fundamental (metaphysical) issues in Knowledge Representation, and to indicate how Kayser's notion of "variable ontology" for "conceptual adaptation" might be interpreted as *pragmatic ontology*, a model methodology for Conceptual Structures research and development. At least, here may be an introduction to further investigations?

1 Introduction

I take my starting point from Daniel Kayser's plenary paper for ICCS in 1998, under the cordial title "Ontologically Yours." He considers the distinction between the traditional use of the term "ontology" in philosophy and the current use of it by AI researchers in building conceptual structures, finding nothing wrong with that "so long as what is meant remains clear," which his investigation indicates is not the case. Furthermore, he says both the philosophical and the AI meanings of the term are inadequate for appreciating the challenges faced in conceptual structures development. He specifically argues that AI's "quest for a unique ontology" of anything more than perfectly defined mathematical notions is a wrongly conceived objective (see his reference to Mugnier's example, 1: 47). And he concludes that "ontological multiplicity" implies the need for "conceptual adaptation," which is essential if we want tools truly adapted to real life domains with a high standard of rigor, not to their idealization [1: 47].

Kayser maintains that the question of how Knowledge Representation (KR) researchers choose their entities of concern (whether these are concepts, relations, individuals, or whatever) *is* indeed an ontological question. He indicates why it has not been considered so, and says traditional philosophy offers no help. His leading question asks: "When AI considers some questions as being ontological, does it really mean that we are concerned with metaphysical problems?" [1: 35]

AI is mainly concerned with the adequacy of its models for a purpose, and that adequacy, Kayser asserts, "is not grounded on being 'real'; [and furthermore] being realistic, i.e. aiming at the representation of 'all' aspects of reality, goes in the opposite direction to being adequate, for obvious complexity reasons"[1: 35]. He sums up the difference: AI wants to find the basis for efficient models; philosophy wants to

B. Ganter and G.W. Mineau (Eds.): ICCS 2000, LNAI 1867, pp. 82 -99, 2000.
©Sp ringerVerlag BerlinHeidelberg 2000

discuss the problem of real existence. "Their goals are not compatible, and their methods are completely contradictory" [1: 35-36].

I will specifically respond to Kayser's arguments and suggestions, after outlining what C.S. Peirce's "scientific philosophy" might offer in analyses of *existence*, *reality*, and *representation*. At the end of the nineteenth century, Peirce confronted philosophical circumstances that are historically significant in explaining what Kayser distinguishes as the AI and the philosophical goals. Peirce took issue with both goals, in terms of the nominalist understanding of representation (which replaced metaphysical questions of existence in philosophy with what amounts to a contest of models) and "seminary metaphysics" (which did not effectively distinguish reality from existence). Kayser's paper introduces the contemporary context of ontological issues for which Peirce urged construction of a "scientific philosophy" and claimed that pragmatism was the method needed to develop metaphysics as an essential part of it.

2 Modern Philosophy and Metaphysical Horror

Helpfully, Kayser begins examination of the ontological difficulty in terms of how "the layman" understands what "to exist" means, which he refers to as "the most 'natural' way of existing." He says, for example, "we often forget that 'tree' is a word that people use to refer to a category of objects, and that the *existence* of trees is nothing more than an agreement that . . . [specific objects] share common properties that we consider as important" [1: 36]. He then distinguishes that "natural existence" from "existence inside a model" in the mathematical terms of tautologies. He admits that the distinction between these two is ill-defined, because the first one "refers to an agreement about the 'obvious model' of Nature, hence existence in Nature is also existence inside of a model" [1: 36]. Then he raises what he calls the "deeper problem" of whether the existence of the tree *is* in Nature or only in the "obvious model" we construct for some convenient purpose. As the paper proceeds, he shows that this metaphysical puzzle, which seems to be purely philosophical, is fundamentally crucial in the pursuits of AI.

In Peirce's time (mid-to-late nineteenth century), European academic philosophy had framed that problem in terms of Realism vs Idealism. However, by that time, Kant's *Critique of Pure Reason* was beginning to undermine German Idealist metaphysics, and the British Nominalists had all but eliminated Realist (Scholastic) metaphysics from the domain of philosophical pursuits and relegated it to theology. Very few philosophers critically examined these dominant trends (as Peirce did), which eventually changed ontological questions into epistemological questions. A new academic dichotomy, Empiricism vs Rationalism, converted the fundamental philosophical question from "What exists?" to "How can we know anything?" Modern philosophy presumed that we can never ask meaningful questions about existence itself, and can only know through constructing models or theories of reality--that is, of what appears to be true through observation or rationalization. The primary philosophical controversy by the turn of century might be expressed this way: Can we build the best models of reality by rational or by empirical methods; or, which sort of methods should be foundational?

Continuing the nineteenth-century trend, influential philosophers early in the twentieth century ignored any notions from traditional metaphysics (as the search for certainty and for the "ultimate ground of knowledge"), and were encouraged to do so by the rising success of science. "Metaphysical horror," as Kolakowski refers to it, began without anyone saying it in so many words: "the very notions of existence and reality (unless they are empirically definable, as in the distinction between hallucinations and normal perceptions) became useless and misleading" [2: 21-22]. These "radical empiricists" were concerned with problems of valid (or consistent) model-building. They conveniently assumed the nominalist position on metaphysical questions: If what *exists* is a moot point, we need only a consistent model (or representation of knowledge) that conforms to whatever is "objectively experienced," as the sciences can observe and interpret it. In mainstream philosophy, epistemology subsumed or ignored any concerns that had been considered ontological, or metaphysical. In other words, philosophy assumed metaphysics to be irrelevant to the modern concerns of logic, mathematics, and science.

Meanwhile, Peirce (a logician, mathematician, and scientist) insisted, instead, that Kant's philosophy had put the final "lock on the door of philosophy" by claiming that only deductive reasoning can possibly be valid. He says, "Thus, we seem to be driven to this point. On the one hand, no determination of things, no fact, can result in the validity of probable argument; nor, on the other hand, is such argument reducible to that form which holds good, however the facts may be. This seems very much like a reduction to absurdity of the validity of such reasoning; and a paradox of the greatest difficulty is presented for solution" [*CP*: 5.347]. Kant's conclusion had implied to his nineteenth-century followers that logic (in its traditional state of development) was actually of no use to the dynamic and complex world of scientific discovery (or to all probable inference, whether induction or hypothesis; that is, inference from the parts to the whole or statistical inference). In response to this circumstance, as Tursman explains, "it is no accident that Peirce's system of logic was designed to be a theory of scientific discovery . . . [he] regarded every step in science as a lesson in logic and after many long and difficult lessons, the observant logician learns what logic must be in order to be powerful enough to encompass the domain of scientific discovery" [3: xi].

From a late twentieth century perspective, Bernstein broadly summarizes Peirce's confrontation with modern philosophy and its misunderstanding of scientific inquiry.

> Retrospectively, we can see how Peirce's philosophic inquiries span issues that have become central to both Anglo-American and Continental philosophy. As a practicing experimental scientist and one of the most creative logicians of his time, Peirce forcefully argued that the dominant conception of the nature of science that had been shaped by the empiricist and rationalist tradition was grossly inadequate. It distorted the true experimental spirit of scientific inquiry. . . . He elaborated a new image of scientific inquiry, and how it can be understood without indubitable foundations. He sought . . . a proper understanding of the intersubjective and communal dimensions of scientific inquiry. His "image of science"--and indeed all inquiry--focused on inquiry as a continuous self-corrective activity governed by the norms of a critical community of inquirers. . . . But Peirce was not just a philosopher of science and a logician. If we are to gain a genuine understanding of scientific inquiry, then it is necessary to probe metaphysical and cosmological

> issues. Meaning and truth in science must be examined from a more general theory of language and signification. [4: xx-xxi]

Karl-Otto Apel summarizes the implications of Peirce's philosophical view that knowledge brings about mediation for a consistent opinion about the real, or the 'representation' of our outer (hypothetical) states of affairs:

> The characteristic innovation of Peirce's logic of inquiry cannot be regarded as a return to metaphysical Realism or Idealism, but rather as a meaning-critical postulate in the framework of a semiotic transformation of Kant's "transcendental logic." This occurs when Peirce replaces [Kant's] concept of "incognizable things in themselves" with the concept of the "infinitely cognizable" and [Kant's] concept of the "transcendental subject" of cognition with the concept of the "indefinite community" as the subject of the "ultimate opinion." [5: 21-22, ix]

Metaphysical inquiry has certainly not improved since 1867, when Peirce wrote: "In fact the prevalent view of the present day is a heterogeneous hodgepodge of the most contradictory theories; its doctrines are borrowed from different philosophers while the premisses by which alone those philosophers were able to support their doctrines are denied; the theory thus finds itself totally unsupported by the facts and in several particulars at war with itself" [*CP* 7.580]

One of Peirce's primary objectives was to reinstate *existence* as an essential philosophical (even metaphysical) concept that could help make tractable the many traditional puzzles and seemingly irreconcilable oppositions of philosophical controversy, with the hope of arresting the endless nominalist debates about the superior validity of one or another model. All of his philosophical work might be understood as part of this objective, which we can identify as the root of the contemporary problems that Kayser examines. Some account of Peirce's philosophy will clarify the implications of his contribution for knowledge representation (in the broadest sense of that term). On that basis, Kayser's concerns and recommendations for conceptual structures development can be specifically considered.

Peirce conceived a pragmatic theory of inquiry and meaning as an alternative to the prevailing philosophical view that abstract terms and concepts must explain concrete experience, and Kant's conclusion that the most abstract (or metaphysical) concepts are ultimate and unanalyzable [see *CP*: 5.177, 207, 289, 294, 500ff]. Peirce insisted that philosophy, just as any science,

> deals with positive truth, indeed, yet contents itself with observations such as come within the range of every man's normal [universal] experience, and for the most part in every waking hour of his life. . . These observations escape the untrained eye precisely because they permeate our whole lives, just as a man who never takes off his blue spectacles soon ceases to see the blue tinge. Evidently, therefore, no microscope or sensitive film would be of the least use in this class. The observation is observation in a peculiar, yet perfectly legitimate, sense. . . . [and] every special science ought to take that little into account before it begins work with its microscope, or telescope, or whatever special means of ascertaining truth it may be provided with. [*CP*: 1.241, 1.246]

The sciences often conveniently ignore puzzles and paradoxes, he says, "and logic here seems to touch metaphysics" [*CP*: 6.182].

All along, in his construction of philosophy as a science, Peirce maintains a self-critical stance: "I am free to confess that objections to this way of thinking have forced themselves upon me and have been found more formidable the further my plummet has been dropped into the abyss of philosophy, and the closer my questioning at each new attempt to fathom its depths" [*CP*: 5.15]. I freely confess that the following account only attempts to interpret and reconstruct from the fragments that remain of his philosophical work (I limit references to *Collected Papers*, as much as possible, but even these can be misleadingly composed and incomplete). I barely skim the surface of both "the abyss" and his systematic effort to plummet and pursue a "scientific metaphysics." Please note: I avoid Perice's nineteenth-century terminology as much as possible (he says, himself: "the twentieth century would laugh at us if we were too squeamish about the word's legitimacy of birth" [*CP*: 2.7, ff p1]). And I do not presume to make more than a suggestion of what his philosophy offers.

3 Peirce's Scientific Metaphysics

Peirce followed Aristotle and Kant (whom he called "the two greatest of metaphysical systematizers" [*CP*: 2.36]) in struggling to make metaphysics (which he considered the philosophy of vagueness [1.204]) logically accessible for examination and use: "it is to be assumed that the universe has an explanation, the function of which, like that of every logical explanation, is to unify its observed diversity" [*CP*: 1.487]. He does not follow Aristotle and Kant in concluding that logic depends on metaphysics; instead, the reverse: "To me, it seems that a metaphysics not founded on the science of logic is of all branches of scientific inquiry the most shaky and insecure" [*CP*: 1.36]. And contrary to Aristotle and Plato, he places mathematics above logic in terms of abstractness [*CP:* 1.240fn.], joining many of his contemporary British and Continental philosophers. Although he expressed concern for the state of metaphysics, "a subject much more curious than useful, the knowledge of which, like that of a sunken reef, serves chiefly to enable us to keep clear of it" [*CP*: 5.410], he disagreed with the common philosophical opinion of his time: "that Metaphysics is backward because it is intrinsically beyond the reach of human cognition," and that "its objects are not open to observation." He blamed its neglect in modern philosophy on the fact that scientists were for the most part uncritical nominalists [see 8]. His philosophy contends that our everyday experience is so saturated with the data of metaphysics that we usually pay no attention to them [see *CP*: 6.2]. Peirce concurred with the opinion of several great thinkers that "the only successful mode [of metaphysical research] yet lighted upon is that of adopting logic as our metaphysics" [*CP* 7.580]. According to Esposito, Peirce's aim was "to discover a unified theory of logic, psychology, and metaphysics, and to present it in some sort of logical form" [6: 147].

Peirce's scheme for classifying the sciences (first constructed in 1896) will serve as a sort of reference diagram (see Figure 1). He uses the word "science" in a broader sense than we currently do, in English, which is roughly equivalent to the German "*Wissenschaft.*"

> Science is to mean for us a mode of life whose single animating purpose is to find out the real truth, which pursues this purpose by a well-considered method, founded on thorough acquaintance with such scientific results already ascertained by others as may be available, and which seeks cooperation in the hope that the truth may be found, if not by any of the actual inquirers, yet ultimately by those who come after them and who shall make use of their results. It makes no difference how imperfect a man's knowledge may be, how mixed with error and prejudice; from the moment that he engages in an inquiry in the spirit described, that which occupies him is science, as the word will here be used. [*CP*: 7.54].

If we view his Classification as a schematic guide to the relations he hypothesizes among scientific studies, the structure indicates that each science draws upon the principles of the studies above it on the list, mathematics being the simplest and most abstract science. Consequently, Peirce says, "We should expect to find metaphysics, judging from its position in the scheme of the sciences, to be somewhat more difficult than logic, but still on the whole one of the simplest of sciences, as it is one whose main principles must be settled before very much progress can be gained either in psychics or in physics" [*CP*: 6.4].

The Structure of Philosophy in a Classification of the Sciences of Discovery represents philosophy as a positive science, in the sense of discovering what really is true but limited to "so much of truth as can be inferred from common experience." The special sciences are principally occupied with the accumulation of new facts inferred from specific human and natural inquiries [see *CP*: 1.183-238].

I. Mathematics (the Conditional or Hypothetical Science, whose only aim is to discover not how things actually are, but how they might be supposed to be, if not in our universe, then in some other [*CP*, 5.40], studies what is and what is not logically possible, without making itself responsible for its actual existence)
II. Philosophy (in Three Branches: Phenomenology ascertains and studies the kinds of elements universally present in the phenomenon; meaning by "the phenomenon," whatever is present at any time to the mind in any way; Normative science distinguishes what ought to be from what ought not to be, and makes many other divisions and arrangements subservient to its primary dualistic distinction; Metaphysics seeks to give an account of the universe of mind and matter. Normative science rests largely on phenomenology and on mathematics; Metaphysics on phenomenology and on normative science [*CP*: 1.186])

> **A. Phenomenology** (the science of experience [in terms of Category Theory: Firstness, Secondness, Thirdness]. Peirce says: "The most fundamental fact about the number three is its generative potency. This is a great philosophical truth having its origin and rationale in mathematics. . . . I analyze experience, which is the cognitive resultant of our past lives, and find in it three elements. I call them Categories. . . . I will so far follow Hegel as to call this science Phenomenology although I will not restrict it to the observation and analysis of experience but extend it to describing all the features that are common to whatever is experienced or might

conceivably be experienced or become an object of study in any way direct or indirect" [*CP*: 4.390, 2.84, 5.37])

B. The Normative Sciences (the science of the laws of conformity of things [as phenomena] to ends, that is, perhaps, to Truth, Right, and Beauty" [*CP*: 5.121])

1. Esthetics (the science of ideals; considers those things whose ends are to embody qualities of feeling)

2. Ethics (the theory of self-controlled or deliberate conduct; considers those things whose ends lie in action)

3. Logic (Formal Semiotic, or the science of self-controlled or deliberate thought; considers those things whose end is to represent something)

a. Philosophical Grammar (Speculative Grammar, or the theory of meaning)

b. Critical Logic (the theory of inference)

i. Abduction (logic of hypothesis)

ii. Induction (logic of sampling)

iii. Deduction (formal logic)

c. Philosophical Rhetoric (Speculative Rhetoric, i.e., theory of method)

C. Metaphysics (the science of Reality. Reality consists in regularity. Real regularity is active law. Active law is efficient reasonableness, or in other words is truly reasonable reasonableness. Reasonable reasonableness is Thirdness as Thirdness" [*CP*: 5.121-129; see also V 6, B 1])

1. Ontology (general metaphysics)
2. Psychical metaphysics (religion)
3. Physical Metaphysics or Cosmology (natural laws)

III. The Special Sciences: these are now represented by the various disciplines in a college of arts and sciences, apart from philosophy and mathematics. Peirce divides these into what we now think of as the humanities and human studies (including psychology and the social sciences) and the natural sciences, which would correspond roughly to the distinction between the *Geisteswissenschaften* and the *Naturwissenschaften*) [see 7].

Figure 1.

Peirce's Classification represents his hypothetical *generative* relations among the sciences. That is, from his logical perspective, the more concrete or special sciences would require the results of the more abstract or general sciences as fundamental principles. While mathematics and philosophy would make use of the results (or selected data) of the former only to support generalizations as needed, he insists that their abstractions must, in some manner, give account of themselves in terms of concrete experience in human affairs (in demonstrations, dialogue, and other conduct in which their indices are indispensable and function as relative pronouns that refer--accidentally or indirectly--to existing or possibly existing things [see *CP*: 2.611]). The deplorable state of metaphysics is evidence of insufficient philosophical understanding of what he says is the function of this "hypostatic abstraction" in normative science (which, for example, tranforms "it is light" into the testable

hypothetical, "there is light here" [see *CP*: 4.235]) [see *CP*: 6.2]. Metaphysics, just as any scientific inquiry, must insist that all abstractions be interpreted *hypothetically*, in terms of "what they would or might (not actually will) come to in the concrete" [*CP*: 6.485]. A highly speculative and fallible science, metaphysics should be cautiously undertaken and its results treated as highly provisional [see 7].

Existence is the ultimate hypostatic abstraction, the ultimate hypothesized condition (that is, the ultimate concept of what experience refers to) that gives normative science the concept of *truth*, and the special sciences the concept of *objective reality*. The postulation of existence gives us the hope that, "if the truth of any question can ever be got at, we shall eventually get at it" [CSP-MS 831; 1900] and that "the real is that which sooner or later, information and reasoning would finally result in, and which is therefore independent of the vagaries of you and me" [*CP* 5.311]. Pragmatism is the method of scientific investigation that reminds us to keep our concepts of what is real growing, guided by that hope. Peirce's Classification begins to clarify the nature of investigations needed to build a scientifically useful philosophy. Based on the phenomenological categories (Firstness, Secondness, and Thirdness), his philosophy constructs a normative theory of how our experience can continue to *grow* from instinctive (or metaphysical) experience to knowledge through inquiry, or in learning by self-governed conduct (in thought and action).

> I hold that we can directly observe [these categories] in the elements of whatever is at any time before the mind in any way. They are the being of the qualitative possibility, the being of the actual fact, and the being of law [or mediation, or regularity] that will govern facts in the future [*CP*: 1.23].

Peirce's metaphysics postulates that these modes of being (or existence, as we phenomenally experience it) are all *real*, in the experiential terms of sensory stimulation, intuition or imagination, judgments and beliefs; while the purely ontological question of what *exists* beyond experience, remains forever hypothetical. This metaphysical theory asserts that, fundamentally, we know to the extent that our experience is constrained by what exists—and that, ultimately, we know to the extent of the growth in our experience through our capability to test those constraints, which depends on how effectively we can represent them. In the representation of constraints, they become cognitive possibilities. Our capability to represent the 'external' constraints we experience *of whatever exists*, has evolved to culminate in the "scientific method," which Peirce formulates as pragmatism. Without the philosophical (or metaphysical) acknowledgment of existential constraints (no matter how uncertainly or provisionally we must conceive them), knowledge becomes hopelessly relative to individual human tastes (nominalistic), which makes communication and the collaborative knowing we call "science" irrelevant and even impossible, in any sense that truly meets human needs.

These logical implications convinced Peirce that metaphysics needed to be especially carefully reconstructed, by pragmatic inquiry. Otherwise, uncritically and dogmatically adopted beliefs would continue to influence the special sciences: "Find a scientific man who proposes to get along without any metaphysics . . . and you have found one whose doctrines are thoroughly vitiated by the crude and uncriticized metaphysics with which they are packed" [*CP*: 1.129]. The sciences would remain in a state of confusion as to what is *real*, unconsciously caught in what Hookway calls a

" web of words" (that is, words referring to other words as definitions), which is just the condition in which we now find much of cognitive science and AI research [8 :259].

Peirce's scientific, metaphysical postulation of existence allows him to develop a theory of experience as inquiry that accommodates linguistic and conceptual relativity, but not as a fundamental principle to assume; instead "man's reason is allied to the originating principle of the universe" [*CP*: 2.24]. The history of science, among other evidence, shows that "man's guesses at the course of nature are more often correct than could be otherwise accounted for, while the same facts equally prove that this [reasoning instinct] is extremely uncertain and deceptive, and consequently unfit to strengthen the principles of logic in any sensible degree" [*CP*: 2.25]. Our ultimate concern, then, is not merely to establish consensus that would simply resolve diverse opinions, but to reach consensus about interpretations (as hypotheses) that could then continue to be tested and modified in further, concerted investigation of the (always hypothetical) existential conditions. "There would not be any such thing as truth unless there were something which is as it *is* independently of how we may think it to be" [*CP*: 7.659]. "Truth being, then, the agreement of a representation with its object" [*CP*: 8.3], which we know only approximately, as *tending to be true*.

In an unpublished manuscript, Peirce clarifies the subtle but crucial distinction he draws between reality and existence, in terms of the necessary metaphysical relationship we must postulate between representations and their objects, or between the model and the modeled.

> By real, I always mean that which is such as it is whatever you or I or any generation of men may opine or otherwise think that it is. There must not be any confusion between reality and existential,--that is real which is as it is no matter what one may think about it, the existential is that which is as it is whatever one may think about anything. No doubt there are grades of reality, meaning that objects of signs may yield with more or less resistance to opinion or representation. According to the definition absolute resistance is essential to reality. But an approach to reality, something that is not in the slightest of the nature of a pretense is found wherever an object of thought is sufficiently obstinate to enable us to say, it has not those characters but it does have these, there is already a lesson in logic. Namely, that one may lay down the very best of definitions, going to the very heart of things; and yet there will be, as it were, a little living mouse of a quasi exception which will find or make a hole to get in when all seemed hermetically closed. This mouse will not be a mere pest to be got rid of and forgotten. It will be a fellow being to be remembered and to be appraised. [MS 498: 32-33]

In the current terms of AI research, we might consider *reality* as an evolving ontological model of existence, for which pragmatic methods in normative science would define the significant contextual relations (or concepts) of particular application domains as they inevitably evolve in operation. "The effect of pragmatism [as a scientific method of conduct] is simply to open our minds to receiving any evidence" [*CP*: 8.259]; so that, "as we remain disposed to self-criticism and to further inquiry, we have in this [pragmatic] disposition an assurance that if the truth of any question can ever be got at, we shall eventually get at it" [MS 831: 4-5]. In the terms most pertinent to AI, the pragmatic disposition would give assurance that

if the definition of any concept can ever be got at, we shall eventually get at it. I will hope to clarify these remarks in what follows (see also [9]).

4 Pragmatic Ontology and Knowledge Representation

Peirce's scientific philosophy does not rely on a deductive or axiom-theorem model [see 6], nor on the inductive or empiricist-positivist model that, in the twentieth century, we commonly considered to be scientific. Instead, he introduces pragmatism as the method to mediate between such opposing approaches in constructing the normative model, or, what I call a "generative model." If we follow his work from that late 1890's into the early twentieth century, we find that he refers to pragmatism variously as "the logic of abduction" (or of "retroduction," properly translated) [*CP*: 5.195-205], "the method of methods" [*CP*: 7.59], "critical common sensism" [*CP*: 5.494], and "the master key to all the secrets of metaphysics" [*CP*: 5.17].

Peirce tells us that the "authors of pragmatism" include the great philosophers of Western tradition, and that he only served as the "civil engineer" who subjected its elements to tests and constructed it to serve his scientific purpose:

> It is expected to bring to an end those prolonged disputes of philosophers which no observations of facts [alone] could settle, and yet in which each side claims to prove that the other side is in the wrong. Pragmatism maintains that in those cases the disputants must be at cross-purposes. They either attach different meanings to words, or else one side or the other (or both) uses a word without any definite meaning. What is wanted, therefore, is a method for ascertaining the real meaning of any concept, doctrine, proposition, word, or other sign. The object of a sign is one thing; its meaning is another. Its object is the thing or occasion, however indefinite, to which it is to be applied. Its meaning is the idea which it attaches to that object, whether by way of mere supposition, or as a command, or as an assertion. [*CP*: 5.5-7]

Ontological metaphysics, far from being irrelevant in Peirce's philosophy, concerns the most general questions of all science: "What is reality? Are necessity and contingency real modes of being? Are the laws of nature real? Can they be assumed to be immutable or are they presumably results of evolution? Is there any real chance, or departure from real law?" [*CP*: 5.496]. These are matters that underpin our everyday, instinctive or intuitive behavior and unexamined feelings, reactions, and beliefs. They are assumptions upon which all our judgments and rational conduct depend: "our logically controlled thoughts compose a small part of the mind, the mere blossom of a vast complexus, which we may call the instinctive mind" [*CP*: 5.212]. His pragmatic theory of inquiry (or learning by experience) begins with the assumption that "All human knowledge, up to the highest flight of science, is but the development of our inborn animal instincts" [*CP*: 2.754, 6.604]. And yet, no creature can have instincts for every possible circumstance, and "when one's purpose lies in the line of novelty, invention, generalization, theory--in a word, improvement of the situation . . . instinct and the rule of thumb manifestly cease to be applicable" [*CP*: 2.178].

Human reason exercises instinct and intuition as imagination: "[We] can stare stupidly at phenomena; but in the absence of imagination they will not connect

themselves together in any rational way" [*CP*: 1.46]. What I call Peirce's *pragmatic ontology* (or generative study of our experienced relations to existence) conceives the evolution of meaning as a "bootstrapping operation" that is not aimless, due to the hope which the postulation of *existence* provides: in scientific reasoning, we begin with what we can imagine the truth must be [see *CP*: 1.46]. Peirce's normative logic investigates the intelligent urge and ability to clarify our beliefs and opinions (as hypotheses) to make them more effectively testable and comparable with the opinions of others, in terms of three modes of scientific reasoning, again, related in generative form.

> Abduction is the process of forming an explanatory hypothesis. It is the only logical operation which introduces any new idea; . . . Abduction merely suggests that something may be. Its only justification is that from its suggestion deduction can draw a prediction which can be tested by induction, and that, if we are ever to learn anything or to understand phenomena at all, it must be by abduction that this is to be brought about. [*CP*: 5.171]

The three normative sciences (esthetics, ethics, and logic) correspond in their order to Peirce's phenomenal categories (Firstness, Secondness, and Thirdness), and pragmatism is the method by which normative science can investigate the data of metaphysics. These data "appear in their psychological aspect as Feeling, Reaction, and Thought" [*CP*: 8.256], elements of experience that are too obvious for us to notice without the pragmatic method of *self-critical* investigation. Significantly, he contends: "It is not to philosophy only that pragmatism is applicable. I have found it of signal service in every branch of science that I have studied. My want of skill in practical affairs does not prevent me from perceiving the advantage of being well imbued with pragmatism in the conduct of life" [*CP*: 5.14].

In everyday life each of us must assume or believe something (make judgments), in order to direct our conduct with respect to whatever exists—to make our actions more than simple physical reactions (to mediate our feelings and actions by means of inferences about what appears to be true). The urge to reach conclusions, or to "take our maps to *be* the territory" (or take our beliefs to be the truth), is a necessary part of effective behavior; but intelligence gives us the capability to believe provisionally and, by observation and imagination, to examine the actual and possible outcomes of our behavior, self-critically, by as many means as we can create to do so (special skills of observation, multiple powers of expression and comparison of these observations, and elaborate technological augmentations of these skills and powers through media). In science (or any human enterprise conducted as science) we must suppose that such "pragmatic mediation" will continue indefinitely, and that our concepts, definitions, and ontologies (or representations of any sort) will never be complete and precise, but only indicate what is possible evidence to test in further experience.

What I call "pragmatic ontology" would be the hypostatic abstraction of indefinite inquiry (that is, what hypothetically represents the real model of existence as the hypothetical goal of inquiry; I will hope to clarify this confusing philosophical expression in terms of a model methodology!). Accordingly, the method by which we pursue inquiry must represent and support the evolution of knowledge in terms of *regularity*, or *structural tendencies in the growth of meaning*. Peirce's concern about nominalism can be crudely expressed as the individual's (possibly instinctive) urge to believe that some particular "map of reality," or ontological assumption, might as

well be considered to be the correct conceptual construct of "the territory," or what exists (since no one can know for sure otherwise), thereby rendering the idea or hope of genuine inquiry pointless. In response to this hazardous instinctive conclusion, his generative philosophy requires the pragmatic investigation of our *reliance on representations*, by which we continue to increase our knowledge of the *regularity* of any existential conditions. His new question for a scientific philosophy is "How does meaning grow?"

We "block the road of inquiry" (which is the motto Peirce said should be above any door of philosophy) to the extent we *believe* that any abstract structure can finally contain meaning. The human predicament is that we must believe we can contain or record meaning (in our expressions and thoughts) to some extent, or else we would never be able to reason at all—much less conduct ourselves on the basis of our thoughts, or express them to conduct ourselves jointly. Pragmatism invites us to accept this as an ontological constraint or normative condition of inquiry, but to employ it consciously by *provisionally* using concepts, categories, and defined terms to construct as many ontological models as possible, while *continuing* to examine these representations critically, and to test them experientially. Considered as the logic of abduction, Peirce says pragmatism should perform two functions: "Namely, it ought, in the first place, to give us expeditious riddance of all ideas essentially unclear. In the second place, it ought to lend support, and help to render distinct, ideas essentially clear, but more or less difficult of apprehension; and in particular, it ought to take a satisfactory attitude toward the element of Thirdness" [*CP*: 5.206].

Peirce reminds us: "The nominalists' difficulty . . . is their habit of reducing the possible to the actual and of not distinguishing the actual [or our immediate sense of the existential] from the real [or our mediated experience of the actual as possible]" [*CP*: 1.422]. The nominalist assumption that the identity of the real is merely a matter of representation, which is ultimately undefinable, inverts his phenomenal Categories (Thirdness, Secondness, Firstness) to claim that abstractions explain concrete experience. What I interpret as "pragmatic ontology" contends that representations must *continue* to give an account of themselves, in tests of concrete experience, if we want reliable knowledge (not merely what purports to be logically valid knowledge). Only simple qualities of sense or feeling, or blind reactions between these, can be indefinable; concepts are relational and definable, but never finally and absolutely [see *CP*: 8.305]. He contends that Reality is a conception of *representation as representation*, that is, representation *as* the mediation between actuality and possibility in experience). "To be a nominalist consists in the undeveloped state in one's mind of the apprehension of Thirdness as Thirdness [or representation as representation]. The remedy for it consists in allowing ideas of human life to play a greater part in one's philosophy" [*CP*: 5.121-129]. In response to what Kayser identifies as AI's "ontology problem," this remedy calls on AI research to question its assumptions about concepts, relations, and other "entities of concern" in building efficient ontological models.

5 Pragmatism and Ontological Problems in Conceptual Structuring

Peirce's philosophy implies the need for pragmatically evolving models of ontology in scientific investigation of metaphysical assumptions, to rescue AI research from the nominalist's problem of trying to ignore metaphysical issues. Peirce even encourages: "If the idealist school will add to their superior earnestness the diligence of the mathematician about details, one will be glad to hope that it may be they who shall make metaphysics one of the true sciences" [*CP*: 8.118]. Kayser invites such investigation in the last line of his paper: "This idea [of conceptual adaptation] provides by itself no methodology; at least, endorsing it means to stop looking for oversimplistic solutions where they obviously do not work" [1:47]. We can now interpret Kayser's main thesis as a metaphysical problem made puzzling by AI's unconscious adoption of an idealistic metaphysics, for which Kayser himself suggests the pragmatic solution:

> [T]he inadequacy of current AI ontologies--as soon as they concern a domain dealing with concepts which are not perfectly defined logically--originates in an idealistic view which does not meet AI needs. This view should be replaced by a much more flexible one, taking into account variability, i.e, the fact that ontological uniqueness is a wrong requirement for AI, and borrowing from the linguistic processes by which the meaning of a word gets adapted to its context of use [1: 36-37].

More specifically, Kayser says, "AI has completely neglected that what we choose concepts to refer to must be revisible," and he suggests the idea of "variable ontology" to respond to the "real world" circumstance of our changing conceptual identifications. He tentatively concludes: "The only consequence that we draw for now is that the solution of an 'ontological problem' should not be a structure such that the concepts of the application rigidly map to a unique entity of the [conceptual] structure. The degrees of freedom that are needed should answer the problems of vagueness or fuzziness as well as the problem of the identity of objects" [1: 39].

Kayser gives examples of how difficult it is to use language in structures that are 'ontologically correct,' even for specific application contexts. If we arbitrarily assign a "basic" concept to an object by means of a term, he explains, then "nearly every occurrence of the term must be treated as a metonymy, that our knowledge of the usual situations [as the experience of interactions with the object] allows us to decode more or less accurately, in order to get an expression using the 'basic' meaning" [43]. He reviews the difficulties to be acknowledged:

> – a choice of the "basis" [or basic meaning], for which no solid ground seems available,
> – a tedious, sometimes reckless, translation of the occurrences of the term, resulting in
> – an overall impression of artificiality and inadequacy of the conceptual representation. [1: 43]

These specific difficulties can be construed as the same problems for which Peirce urged the application of pragmatism to build a scientific philosophy. Rather than looking for 'basic meaning,' AI needs to be able to identify what meaning is unclear in

any particular application context, and either eliminate the term and concept that represent it, or render the meaning distinct by considering (imagining what would be) the consequences of accepting specific definitions, in the conduct of specific human affairs. Meaning should never be assumed to reside in the terms, definitions, relations, or conceptual representations of any sort; they are only the medium [or Thirdness] by which meaning continues to grow. Real meaning is ascertained by experimentation in real contexts of actual (or existential) operation. Pragmatism is "no metaphysical doctrine, no attempt to determine any truth of things," says Peirce; this "method of ascertaining the meanings of words and concepts is no other than that experimental method by which all the successful sciences . . . have reached the degrees of certainty that are severally proper to them today; this experimental method being itself nothing but a particular application of an older logical rule, 'By their fruits ye shall know them'" [*CP*: 5.464, 465]. The method prescribed "is to trace out in the imagination the conceivable practical consequences, -- that is, the consequences for deliberate, self-controlled conduct, -- of the affirmation or denial of the concept; and . . . herein lies the whole of the purport of the word, the entire concept" [*CP*: 8.191].

Although Kayser agrees that the question of how KR researchers choose their entities of concern is an ontological question, he concludes that traditional philosophy offers no ontological help to KR researchers in choosing entities of concern to build representational structures [1: 35]. "AI wants to find the basis for efficient models; philosophy wants to discuss the problem of real existence; their goals are not compatible and their methods are completely contradictory." [1:35-36].

Peirce conceived pragmatism to reconcile the different goals (in terms of nominalist and realist). The efficiency of intelligence in establishing reliable habits (models, beliefs, judgments, definitions, conduct, and so forth) depends on our capability to conceive past results and postulate future outcomes (possible causes and effects), through communication (with oneself and in cooperation with others) in inquiry. Genuine knowledge progresses to the extent that we are capable of representing our observations effectively (as a basis for further observation), and can continue to develop this capability by means of tools to serve that purpose. Pragmatic methodology is conceived to maintain more valid (internally consistent) and reliable (externally responsible) evolution of representation. Efficiency entails real effectiveness; validity entails existential reliability. Efficient models *are* models of reality, in Peirce's terms.

Peirce's ontology claims that the human mind must have been attuned to the truth of things in order for science to discover what it has discovered. Instinct is the very bedrock of logical truth [*CP*: 6.476], on which all reasoning (as regularity in thought) must be built [*CP*: 6.500]. However, "an absolutely determinate term cannot be realized, because, not being given by sense, such a concept would have to be formed by synthesis, and there would be no end to the synthesis because there is no limit to the number of possible predicates. A logical atom, then, like a point in space, would involve for its precise determination an endless process. We can only say, in a general way, that a term, however determinate, may be made more determinate still, but not that it can be made absolutely determinate" [*CP*: 3.93].

Kayser pinpoints the problems of object identity ("extensional" view of what constitutes a concept) and the problem of the inevitable vagueness of concept definition ("intentional" view of what constitutes a concept). "What we consider in the real world to be an object tends to be much more problematic than most of us are ready to admit [and] language adds a lot of difficulties that we cannot ignore" [1: 38].

He considers the difficulties exhibited in the Generative Lexicon a hint as to the pointlessness of keeping concepts pure from any linguistic contamination [see 1:43-44].

Pragmatic ontology would analyze these metaphysical difficulties in phenomenological terms, as Peirce suggests in the following excerpt, and normative science would investigate the implications for any special scientific purpose, such as AI's.

> The mind, by its instinctive adaptation to the Outer World, represents things as being in space, which is its intuitive representation of the clustering of reactions. What we call a Thing is a cluster or habit of reactions, or, to use a more familiar phrase, is a centre of forces. In consequence, of this double mode of association of ideas, when man comes to form a language, he makes words of two classes, words which denominate things, which things he identifies by the clustering of their reactions, and such words are proper names, and words which signify, or mean, qualities, which are composite photographs of ideas of feelings, and such words are verbs or portions of verbs, such as are adjectives, common nouns, etc. [*CP*, 4.157].

Finally, Kayser suggests the possible solutions of semantic adaptation and conceptual adaptation, based on awareness that a word can take virtually an unlimited set of meanings in various contexts, and that dictionaries keep only the most typical semantic values (a seldom acknowledged limitation to text understanding). He speculates that concept adaptation may be as necessary in conceptual structuring as semantic adaptation is inescapable for text understanding, implying the need to employ multiple conceptual structures, without being misled by identity of concept names. He suggests the idea of introducing levels, with *stable* concepts being structured for each domain of interest at different levels (levels could be discarded when modification makes no sense or regular patterns give meaningful results). He hypothesizes that conceptual adaptation would be needed to cope with ontological multiplicity, if we want tools truly adapted to real life domains; then he deduces.

> More precisely, I argue that any formula (F) representing some piece of knowledge (or some questions to be answered, e.g. If the formula has free variables) in a conceptual language should not be matched with (for consistency checking) or fed into (for inference) a single conceptual structure, identifying the concept names appearing in the formula with those occurring in the pre-existing structure. Instead of that, the formula should rather be confronted with several such structures, without being misled by identity of concept names. In each case, a process of *conceptual adaptation* should be invoked, in order to discard the levels where (F) makes no sense, and to modify (F), following regular patterns, in order to get from it meaningful results. [1: 45-46]

6 Conclusions

I argue that Peirce's philosophy would supply the theoretical and methodological guidance needed to carry out Kayser's suggestions (or abductions, expressed above), pragmatically:

> A hypothesis must explain the phenomena in question. An [abductive] analysis of its logical purport, of its would-bes, allows an inquirer to determine this. Deduction then develops the implications of the would-bes, and induction tests for the reality of the generality that is the hypothesis or, more accurately, the object of the hypothesis, and thus "gives us the only approach to certainty concerning the real that we can have" [*CP*: 8.209].

Elsewhere [10], I have suggested what is called the "collaboratory testbed model" (currently being advanced in scientific research) as the context in which KR research and development could be conducted pragmatically. In that context, technological innovation evolves by means of "partnerships" involving those who need technology to augment their collaborative scientific inquiry and those who build augmentation technology to support that work. The evolution of such systems could be conceived in terms of pragmatic ontology. I have suggested that the communications infrastructure for the testbed methodology could be advanced by integrating KR tools for that purpose. Conceptual Graphs and Formal Concept Analysis employed in various modes of communication (both normative and algorithmic) could create an effective medium for the conceptual *structuring* that collaboration requires. Many Conceptual Structures tools already exhibit promise of that potential and only need a collaboratory research program, in which they could be considered tests of Peirce's scientific philosophy--whether AI researchers care about the metaphysical status of what they are studying or not! The Sisyphus testbeds [10] for conceptual structures research indicate the rationale and benefits of a pragmatic research methodology; and I have suggested that the PORT (Peirce Online Resource Testbeds [11]) project would be a significant context for developing, evaluating, and continuing to improve KR tools *by means of, and for* further collaboratory infrastructure advancements. In that research and development context, application systems would be treated as theoretical hypotheses to be tested and modified for continuing improvement. Tools would be embodied hypotheses of what is required for specific purposes in specific contexts of operation [see 12]. Pragmatic ontology would need an evolving communication medium, in which truly challenging puzzles could be investigated. Here are two (which Peirce says would require his existential graphs):

> They are the puzzle of the relation of signs to minds, and of their communication from one mind to another, and the puzzle of the composition of concepts and the nature of the judgment, or, as we of the antipsychological school say, of the proposition [MS 498: 29; an unpublished MS, entitled by Peirce, “On Existential Graphs as an Instrument of Logical Research”]

In our development of media for representation and communication, particularly technology for the progressively more efficient production of written language, we have been led to believe that we can contain meaning, or record it in permanent, identically reproducible representational (and idealized) structures, such as the "correct ontologies" that Kayser says are too simplistic. But we now have an electronic medium of representation with which we may hope instead to *represent the evolution of meaning*, and to realize that the structures we create are not *ends* (ontologies) but *means* (media) for continuing the growth of experience and knowledge. We no longer need to identify stability of meaning with the ideal of establishing absolute definitions that we imagine to be consistently and perfectly interpreted. Metaphysically observed, what we think of as "the stability of

phenomena" is *regularity*, or *tendency* in growth of meaning. Crucial for any Conceptual Structures project is to build an *operation* that will support *continuing* conceptual growth. Peirce's pragmatic philosophy does not simply encourage us to develop yet another conventional mode of analyzing media and expressions, such as formal logic serves for linguistic instances, as though they were existential conditions—timeless "facts," or the ontological foundations of traditional philosophy. Instead, he developed his philosophy to *investigate and discover* how concepts (along with the media and modes we use to express them) evolve: an evolution that will continue, whether theoretical logic and science help us comprehend it or not [see 13].

AI ontologies are experimental representations of reality, based on tentatively defined relations as evidence of what might possibly exist. We cannot expect that medium of representation to give us a reasonable chance of knowing and controlling actual conditions, validly and reliably, without metaphysical postulation and normative testing in their evolution.

> Memory supplies us a knowledge of the past by a sort of brute force, a quite binary action, without any reasoning. But all our knowledge of the future is obtained through the medium of something else. To say that the future does not influence the present is untenable doctrine. It is as much as to say that there are no final causes, or ends. The organic world is full of refutations of that position. Such action [by final causation] constitutes evolution. But it is true that the future does not influence the present in the direct, dualistic, way in which the past influences the present. A machinery, a medium, is required. Yet what kind of machinery can it be? Can the future affect the past by any machinery which does not again itself involve some action of the future on the past? All our knowledge of the laws of nature is analogous to knowledge of the future, inasmuch as there is no direct way in which the laws can become known to us. We here proceed by experimentation. That is to say, we guess out the laws bit by bit. We ask, What if we were to vary our procedure a little? Would the result be the same? We try it. If we are on the wrong track, an emphatic negative soon gets put upon the guess, and so our conceptions gradually get nearer and nearer right. The improvements of our inventions are made in the same manner. The theory of natural selection is that nature proceeds by similar experimentation to adapt a stock of animals or plants precisely to its environment, and to keep it in adaptation to the slowly changing environment. But every such procedure, whether it be that of the human mind or that of the organic species, supposes that effects will follow causes on a principle to which the guesses shall have some degree of analogy, and a principle not changing too rapidly. [*CP*: 2.86].

By investigating fragmentary manuscript evidence from the past, we might hope to make Peirce's existential abduction Pragmatically Ours, for the future of knowledge representation. "Indeed, it is the reality of some possibilities that pragmaticism is most concerned to insist upon" [*CP:* 5.453].

References

Note: "MS" references are to Peirce's manuscripts archived at the Houghton Library, Harvard; for *CP* references, *Collected Papers of Charles Sanders Peirce,* 8 vols., ed.

Arthur W. Burks, Charles Hartshorne, and Paul Weiss (Cambridge: Harvard University Press, 1931-58).

1. Kayser, D. "Ontologically Yours," In: Mugnier, M.-L., Chein, M. (eds): *Lecture Notes in Artificial Intelligence*, vol. 1453, Springer-Verlag, (1998) 35-47.
2. Kolakowski, L. *Metaphysical Horror*. Oxford: Basil Blackwell, 1988.
3. Tursman, R. *Peirce's Theory of Scientific Discovery: A System of Logic Conceived as Semiotic*. Bloomington: Indiana University Press, 1987.
4. Bernstein, R. J. "Peirce's Theory of Perception." In: E.C. Moore and R.S. Robin, eds., *Studies,* 1960:165-189.
5. Apel, Karl-Otto. *Charles S. Peirce: From Pragmatism to Pragmaticism* (John Michael Krois, trans.). New Jersey: Humanities Press (paperback edition), 1995.
6. Esposito, J. L. *Evolutionary Metaphysics: The Development of Peirce's Theory of Categories.* Athens: Ohio University Press, 1980.
7. Ransdell, J. M. "Arisbe, a philosophical website, providing coordinated access to the internet resources relevant to the life and work of, and continuing interest in the American philosopher, scientist, and humanist, Charles Sanders Peirce": www.door.net/arisbe.
8. Hookway, C. *Peirce.* London: Routledge and Kegan Paul, 1985.
9. Keeler, M.A. "Investigating Transparency in the Conditions of Mediation from a Semeiotic View." *Semiotica* 82:1-2 (1990):15-41.
10. See "Sisyphus I" projects. In: Tepfenhart, W., Cyre, W. (eds): *Lecture Notes in Artificial Intelligence*, vol. 1640, Springer-Verlag, (1999) 272-355.
11. Keeler, M., Kloesel, C., and Searle, L. "PORT: A Testbed Paradigm for Knowledge Processing in the Humanities." In: Searle, L., Sowa, J., Lukose, D. and Keeler, M. (eds): *Lecture Notes in Artificial Intelligence*, vol. 1257, Springer-Verlag, (1997) 100-113.
12. Dillon, A. "Artifacts as theories: convergence through user-centered design." In: Kinny (ed): *Proceedings of the 54th ASIS annual meeting*, Medford, NJ: Information Today, Inc. (1995): 208-10.
13. Ransdell, J.M. "Some Leading Ideas of Peirce's Semiotic." *Semiotica* 19:3/4 (1977):157-78.

Conceptual Modeling for Distributed Ontology Environments

Deborah L. McGuinness

Associate Director and Senior Research Scientist,
Knowledge Systems Laboratory, Gates Building 2A,
Stanford University, Stanford, CA 94305 USA
dlm@ksl.stanford.edu

Abstract. As ontologies become common in more applications and as those applications become larger and longer-lived, it is becoming increasingly common for ontologies to be developed in distributed environments by authors with disparate backgrounds. Ontologies that are expected to be collaboratively created and maintained over time by authors in many locations present special challenges to the problem of conceptual modeling. In this paper, we will discuss conceptual modeling issues and focus on those topics with elevated importance in distributed environments. We will draw on our experience creating and maintaining ontologies in differing knowledge representation and reasoning environments over the last decade. Many of our recent observations are drawn from our experiences in the DARPA High Performance Knowledge Base Program. This program generated dozens of knowledge bases authored by people of varying expertise in both knowledge representation and reasoning as well as domain experience. Our efforts in merging the ontologies, loading them for coordinated use, and modifying them to meet evolving needs shape much of the material in this paper. Additional sources of observations are from designing and building a number of e-commerce ontologies (with content merged from multiple sources) and also from a few families of description logic applications including the PROSE/QUESTAR family of configurators and the FindUR knowledge-enhanced search applications.

1 Introduction

In recent years ontologies have become the subject of interest in communities beyond just those of knowledge representation and library science. They moved out of the research labs and are included in most expert system applications in the 80s and 90s and many of them supported sophisticated inference as well as retrieval. Many times these ontologies were built by people highly trained in knowledge representation and

B. Ganter and G.W. Mineau (Eds.): ICCS 2000, LNAI 1867, pp. 100-112, 2000.
© Springer-Verlag Berlin Heidelberg 2000

reasoning. Many times the author(s) also became highly literate in the domain as well. Although challenging, conceptual modeling for these kinds of situations has been studied and literature exists on proposed strategies for conceptual modeling for projects in which the knowledge acquisition is a fairly centralized process.

Taxonomies, sometimes faceted, have also been the domain of library science for a number of years. They have been successfully developed and maintained by trained staff. One of the more famous examples is the Dewey Decimal System, which is still in use today over a century after it was conceived, and has an active staff associated with it [Vizine-Goetz and Mitchell, 1996]. We will not focus in this area where trained library scientists do their classification work but note that just as the knowledge representation community is now reaching out to the non-trained public, the library science community is also reaching out to "non-catalogers" (such as the work by the Dublin core). Arguably, much of this is driven by the ubiquity of the web and efforts such as the resource description framework (RDF) [Brickley and Guha, 2000].

Recently large ontologies have become more common in broad consumer applications ranging from search (such as Yahoo, Lycos, etc.), to e-commerce and auctions (such as Amazon, EBay, etc.), to configuration (such as Dell, PC-Order, etc.), to more general information sites (such as cnet.com). Sometimes the larger ontologies are so broad that, almost by their nature, they are best designed and maintained in a distributed manner by multiple experts. Sometimes the vocabularies are just simple taxonomies, but many recent background knowledge organizations present structure in the form of properties (such as wine properties in Virtual Vineyards or electrical component properties in specification search on necx.com). It is this class of ontologies that we will be considering in this paper – those where one expects the ontologies to be created and maintained by a staff and possibly not be highly coordinated in its evolution. We will include both the simple concept taxonomies as well as ontologies with structure and many inter-relationships.

One motivational model is the Open Directory project by DMOZ where the goal is to build an "Internet brain" by leveraging the expertise of many experts. (At publication time, there were over 25,000 registered experts, which is up 25% in the last four months, over 250,000 categories, and over 1.75 million sites.) This may be at the extreme end of the spectrum since the ontology is fairly simplistic in nature and enormous in scope, but there are a number of e-commerce and other ontology efforts that have "cybrarian or ontologist" staffs in the dozens and content areas that are very broad. It is our speculation from empirical observation that ontology staffs will continue to grow, the training of entry-level staff will broaden, and ontologies will continue to become more ubiquitous. Given these speculations, the role for ontology environments in distributed settings is becoming more critical.

1.1 Three Motivational Application Areas

Since we believe that expected and actual ontology usage heavily impacts resulting ontology design, we feel it is instructive to provide a context from which to judge our

observations and conclusions. This paper draws largely on three somewhat distinct ontology efforts including a decade of description logic applications (and description logic system and environment design and development), co-directing the Stanford University Knowledge Systems laboratory's high performance knowledge base program (and its frame system environment design and development), and commercial consulting in ontologies. We begin, of course, with rather extensive training in knowledge representation and reasoning.

The description logic applications fall broadly into a family of configurators (used by AT&T and Lucent to configure transmission equipment [Wright, et. al, 1993, McGuinness and Wright, 1998a, 1998b], McGuinness, et. al, 1995]), data mining applications [Selfridge, et. al., 1992], (a successor application was marketed by NCR), and ontology-enhanced search applications and environments (used in online calendars [Fuoss, et. al], electronic yellow-pages [McGuinness, 1998], AT&T Worldnet, and medical applications [McGuinness,1999a]). Additionally of course, there was active development on the representation and reasoning system [Borgida, et. al, 1989, Brachman, et. al, 1990] and the supporting environment for use which led to work on pruning languages [McGuinness, 1996, Baader, et. al, 1999, Borgida and McGuinness, 1996], explaining reasoning systems [McGuinness, 1996, McGuinness and Borgida, 1995, Borgida, et. al, 1998], and usability and environment work [McGuinness and Patel-Schneider, 1999, Brachman, et. al., 1999].

These applications typically had fairly intricate structure of the underlying representation, and at least in the configurators, had extensive use of the reasoner. Their representation in a description logic provided the opportunity for knowledge engineers to input term definitions with precise semantics. Most of the knowledge engineering was done in a structured and "project-managed" fashion so that it was possible to train people about how to build knowledge bases and how to extend them. Environments were built that supported knowledge engineers in building their own new applications so that knowledge representation experts were not required to do much work when new configurators were added (nor when new content areas were added within the framework of the knowledge-enhanced search applications). The data mining applications also had fairly intricate and inter-related object structure. The object structure was used to connect (and essentially to join) a number of legacy database systems. The inference level was not as extensive as that of the configurator applications. The main goal was to provide data analysts with natural and modular access to the data to support their quest for finding patterns in the data. The knowledge-enhanced search applications had the simplest modeling and reasoning structure. They were initially motivated by AT&T's Personal Online Services' browsing and search needs. They were expected to be used by a very broad user base. The application scope expanded to include more targeted user populations in narrower domains as well as broad user communities in wide domains. These applications hold the greatest similarities to the ontologies we observed in recent commercial ontology applications

The work on the High Performance Knowledge Base program [Cohen, et. al, 1998, MacGregor and McGuinness, 1999] used the Stanford ontology environment including Ontolingua [Farquhar, et. al., 1996], OKBC [Chaudhri, et. al, 1998], KIF [Gene-

sereth and Fikes, 1992], and Chimaera [McGuinness, et. al, 2000a, 2000b] to build and maintain many large knowledge bases for intelligence analyst usage. It had a much more distributed nature than the work previously mentioned. The source documents appeared to be generated by multiple experts (usually not trained in knowledge representation). The knowledge engineers were distributed throughout the country and did not always have a lot of interactions with each other or with the original authors of the source documents. The knowledge representation experts were also distributed and sometimes time pressure did not allow them to coordinate their conceptual models as much as they would like. Additionally, ontologies were many times written by a single author or group for one purpose and then picked up by other groups for other purposes in the same content area.

The ontologies typically had fairly intricate structure both in terms of including objects with dozens of properties and many inter-relationships and also in terms of having fairly deep hierarchies. The controlled vocabularies of the upper level ontologies had significant size as well. Inference was sometimes complicated and also many times required common sense reasoning in broad areas (such as geographic reasoning over much of the world). Thus, these ontologies required breadth and depth in most dimensions that one would usually consider when measuring ontology complexity.

The work consulting on commercial ontologies [McGuinness, 1999b] is less academic but has been quickly deployed and thus has more recent empirical basis. The time frames common in startup applications allow (and force) quick deployment and evaluation of ideas. The ontologies were sometimes designed on paper and developed in house using internal (sometimes proprietary tools), sometimes designed and implemented using a combination of Stanford tools and CLASSIC and sometimes handed over to be maintained internally, and sometimes maintained through consulting. The ontologies for consulting purposes tended to contain less intricate structure and less nesting but sometimes had significant depth, breadth, and total size. In a number of commercial ontologies, it was important to include existing "standard" published vocabularies as a portion of the final ontology.

2 Distributed Ontology Desiderata

We will now present some general guidelines abstracted from the previously mentioned ontology experiences. For every principle, we will attempt to motivate its need and benefits. We will also include a discussion of some of the related challenges and include a list of operational guidelines.

2.1 Incorporate Standard Concept Vocabularies (and Capture Their Semantics)

A goal is to include industry standard vocabularies that are familiar to the major classes of users of the system. Assuming the ontology will evolve, those classes of users include both the knowledge engineers as well as the users of the application.

While it is important in centralized ontologies to use standard terms, it becomes more critical as ontologies become larger, more difficult to browse, more difficult to remember, and more complicated to extend and maintain.

Essentially every current generation set of principles for building an ontology includes articulating the expected uses of the ontology. We would add to this, the notion of articulating the expected user profiles (including both knowledge engineers assuming the ontology will evolve and also the application users). In the user articulation, there should be some characterization of what (if any) controlled vocabularies are expected to be familiar. Also, information about the expected ambiguity of an ontology should be included. For example, do the vocabularies provide necessary and sufficient conditions for membership in a class? Do they provide examples of class membership (and possibly examples of non-membership)? Are the ontologies just hierarchies of potentially ambiguous terms? Additionally, it should be noted if incoming data is likely to be consistent (or tagged) with a particular vocabulary. For example, if one is beginning a new e-commerce business-to-business application, one should consider how emerging standards like the UN/SPSC might interact with the resulting ontology. If, for example, any major sources of input data are expected to use a controlled vocabulary, then knowledge engineers will have an easier job if the application ontology is either compatible with or already includes the controlled vocabulary. Similarly, if end users are expected to be in environments where they are using terms from a controlled vocabulary, then they will have more intuitive search and browsing experiences with the application if those terms are recognized by the new application ontology.

If the application ontology contains all of the notions included in a user's vocabulary yet it uses different term names, then it will slow the user down and sometimes will make the user ineffective. For example, if the user expects to see "car" in the ontology and thus does a search for the string "car", he or she will fail to find cars in an ontology that uses only the terms "vehicle" and "auto" (if a standard syntactic retrieval is performed). There are a number of coping strategies for multiple names for the same notion such as thesaurus incorporation, query expansion, browsing support, translators, etc., but before they all end up as requirements to the system design, it is worth determining if a different base ontology would help solve a number of problems. In a growing number of areas, there appear to be some well-designed candidate ontologies that are becoming better supported and accepted. Examples include SNOMED and UMLS in medicine, UN/SPSC in e-commerce, etc.

Another issue related to standard vocabularies is that not only are they emerging, many different vocabularies are emerging as viable options. As the field of common ontologies sorts itself out, one can expect to feel compelling needs to support compatibility with multiple controlled vocabularies. This will lead to another principle discussed below concerning environmental support for interacting with and merging multiple ontologies with systematic environmental support tools.

As ontologies become larger, browsing becomes more difficult. For example, in the high performance knowledge base program upper level ontology, there exist thousands of terms and it is not always easy to find the term for which one is looking. Even after some "rationalization", undoubtedly, there are remaining issues in terms of

alignment that should be handled. In our merging effort (required to load many independently developed ontologies), we discovered a number of missing sub-class links. For example, we found that "strait" was defined to be a "body-of-water" and that "narrow-body-of-water" was also defined to be a sub-class of "body-of-water", yet there was no sub-class relationship between "strait" and "narrow-body-of-water". We hypothesized, and later confirmed, that what was apparently a missed sub-class relationship was due to the fact that the author of the "strait" term was unaware of the previously defined term "narrow-body-of-water".

In order to solve this problem, one cannot do simple synonym matching. Here an explicit representation of the semantics of the terms would be useful. If one had a precise definition, say that narrow bodies of water have a width of less than two miles and straits have widths of less than a mile, then in a classification system such as a description logic, a reasoner could automatically recognize that one is a sub-class of another. Still, even without precise definitions and classifiers, additional support can be provided. For example, in the initial (less specific) encoding of strait as a "body-of-water" (but not a "narrow-body-of-water"), a sibling analysis would note that "strait" is a sibling to "narrow-body-of-water" since they are both direct sub-classes of "body-of-water". Sibling analysis would then ask if the new term – "strait"- can be made a sub-class (or super-class) of its current siblings and also if it is of the same level of specificity as its siblings.

Additionally, some representation languages allow knowledge engineers to represent classes that are disjoint from each other. Such classes may be considered members of a disjoint partition. In ontologies that contain partition or covering information, questions should be asked as to whether the new class should be added to any partitions under its parent class. For example, consider an ontology that contains a partition under the class "weapon" that distinguishes between the disjoint classes "biological-weapon" and "chemical-weapon". An addition of a new sub-class of weapon, say "nuclear-weapon" should trigger questions such as

- Should "nuclear-weapon" be either a super-class or sub-class of "chemical-weapon" or biological-weapon".
- If "nuclear-weapon" is accurately a direct sub-class of weapon, then is it disjoint from the other sub-classes of weapon, and should it be added to the weapon class partition.

Some operational guidelines include:

- Articulate anticipated ontology usage as well as expected user profiles.
- Use a controlled vocabulary that is familiar to users if one exists.
- Specify mappings between multiple standard controlled vocabularies if multiple vocabularies are standard in the domain(s) of interest.
- Allow for user extensibility of the mappings (thus, support users in adding new synonyms into a thesaurus).
- Allow for controlled vocabulary extensions.
- Provide supporting mechanisms such as query expansion in search to help facilitate sub-class matches.
- Specify semantics of terms.

- Provide semantic retrieval instead of just syntactic retrieval.
- Provide additional structural retrieval methods such as sibling retrieval and analysis.
- Provide partition extensibility.

2.2 Incorporate Standard Property Vocabularies (and Capture Their Semantics)

As we mentioned in the introduction, it is becoming increasingly common for web ontologies to include inter-relationships. There has been more effort dedicated to developing standard hierarchies of classes (or noun phrases) than there has been on generating standard groupings (or hierarchies) of properties (or relationships between objects). Still, there are a number of organizations such as NIST, UN (in the UN/SPSC effort), NLM, etc., which are attempting to generate standard vocabularies of property names. Thus, we observe that even today a number of standard role vocabularies exist and we speculate that their growth will accelerate.

We also note that just as there are issues with retrieving classes from large and unfamiliar ontologies, there are also (at least as many) issues with finding role names in the same ontologies. All of the same problems exist and there may be less synonym support for simple synonym matching or query expansion. We have observed many times that role proliferation can quickly become a problem in large knowledge bases and we speculate that one major reason is because knowledge engineers are not finding existing useful roles. For example, a knowledge engineer who does not know that shipping-weight is already in the ontology may add a new role called weight. If the knowledge engineer did not know that shipping-weight existed, then there would be no connection between weight and shipping-weight. Now both terms will be in the ontology and with time, one could expect that some knowledge engineers find weight and thus fill in that role (without filling in shipping-weight) and other knowledge engineers will find shipping-weight and fill in that role (without filling in weight) and some may manage to fill both in (possibly inconsistently). Thus, some objects will have weights but not shipping weights and vice-versa. Possibly, even more problematically, there will be no enforceable constraints between the two roles. (Using commonsense, a shipping-weight would be expected to be equal to or slightly higher than the actual product weight. So if an object had an actual weight of 3 pounds and a shipping weight of one pound, then there would seem to be parts missing and thus an inconsistency.) If there is no relationship between the roles, this inconsistency could not be detected and we would also have no way of letting the system do some work of deducing fillers or at least lower bounds on fillers for object shipping-weights (or possibly estimates of actual-weights) for users.

When we reviewed a number of knowledge bases that contained large numbers of concepts and roles, we found this role proliferation and lack of connection between related roles to be extremely common. Another connection between roles that was missed routinely was an inverse relationship. For example, one source of data may include people, countries, and information about which people are leaders of coun-

tries. Thus, an ontology might include a property named "leader-of-country" and in the case of the individual "Clinton", "leader-of-country" would be filled with "United-States". Similarly another data source may include the property "has-leader" that has a domain including countries or possibly even more specifically a property with the singular domain of countries that might be called "country-has-leader". If there were no inverse relationship stated between "leader-has-country" and "country-has-leader" then it would appear that an application did not know who the filler of "country-has-leader" is on the United States even though it knows that United States fills the role "leader-has-country" on Clinton. In our analysis of a number of knowledge bases designed for simpler purposes and later extended for broader usage, we found this kind of role inconsistency and lack of inverse relationships to be quite common.

We also found that objects can quickly gather many properties in ontologies but many times only a few of the properties are useful for a particular purpose. It is not uncommon in our configuration examples and also in our data mining examples, for objects to have hundreds of properties. It is also not uncommon for the user to be only interested in a few property values. Large and complicated objects typically require some sort of pruning mechanism to help users focus their attention on the aspects of the object presentation that is relevant to them. Some knowledge representation systems like CLASSIC[Brachman, et. al., 1991] include a compositional language extension that allows knowledge engineers and users to specify (context-sensitive) matching patterns to be used when displaying or explaining objects. Simpler approaches include simple markup languages or just flags that tell the system if it should display portions of objects.

Some operational guidelines include:

- Use a controlled vocabulary that is familiar to users if one exists.
- Specify domains and ranges of roles (for example, the domain of "leader-of-country" is "person" and the range is "country").
- Specify inverse relationships between roles (for example, "leader-of-country" is the inverse of "country-has-leader").
- Specify active inferences to infer constraints between role values (for example, "shipping-weight" is greater or equal to the "actual-weight").
- Specify conversion rules for presenting different views of fillers for properties (for example, provide a rule that can calculate a filler for the price role in German Marks if a price filler is known in United States Dollars and a multiplier is available).
- Provide some sort of markup language to allow knowledge engineers and users to prune out roles for certain presentation (and explanation) views.

2.3 Utilize (or Develop) Environments to Support Ontology Evolution

As ontologies become larger and more structure is put in place, it becomes increasingly important to use tool environments to enforce consistency and also to aid users

in focusing their attention in the areas where they are needed to resolve problems. It seems to be a given that most large ontology applications will end up needing to merge multiple ontologies together given that most specifications will require supporting more than one standard vocabulary. Merging small ontologies may not be difficult to do manually, but once ontologies become large, it becomes more critical to provide systematic tool support. Merging tools should identify terms that clearly should be merged (i.e., objects with the same names with the same internal structure and semantics), terms that may be merged (i.e., terms with names that are known to be synonyms for each other with internal structure that is not incompatible), terms that may not be merged (i.e., terms that are known by their definition to be disjoint).

Merging tools also can be used to focus the attention of the user into areas that are likely to require human re-work. For example, if there is a term that appears as a suffix of a number of other terms, it may be the case that sub-class relationships should exist. For example, if one ontology includes the term "weapon" and another ontology includes the terms "biological-weapon" and "chemical-weapon", it is a likely guess that "weapon" should be a super-class of both "biological-weapon" and "chemical-weapon". Merging tools may also include the sibling and partition analysis that we mentioned in Section 2.1.

Verification and validation of ontologies is also an increasingly important task, particularly when ontologies are generated as the result of merging two or more sources. When ontologies become too large for experts to scan, they need support in identifying problem areas. There are a number of things that can be done automatically. For example, authoritative sources, if they exist, may be used to check data when it is input from multiple sources. General deductions may be enforced such as "no element may be an instance of more than one class in a disjoint partition".

Another portion of a diagnostic tool is a more subjective analysis. For example, it is rarely the case that cycles should exist in class graphs. For example, it is unlikely that we would want to say that a cat is a mammal, a tiger is a cat, and a mammal is a tiger. (Just as an empirical observation, cycles do show up in a number of ontologies currently deployed on web sites today as we discovered from crawling a number of sites. It is our analysis, that most, if not all of the cycles we found can be and should be broken.) Cycles can be identified and brought to the attention of the user. Similarly, there are many "rules of thumb" that can be used to "critique" an ontology. For example, typically one finds multiple sub-classes of a super-class. If a class has one and only one sub-class, it is usually an indication that that portion of the hierarchy has not been completed. A note in a log of all of the single child classes can be useful for knowledge engineers in focusing their attention on the areas of the taxonomy that are likely to be incomplete

An environment should provide support for multiple versions. In the simplest case, applications will have the ontology that is currently deployed and the one that is under construction. More typically, in a distributed environment, there may be multiple sub-portions of the development ontology undergoing simultaneous development.

An environment should include support for extending both high level and lower level ontologies. That may be best done by a combination of social and programmatic support. As ontologies become larger, systematic and consistent naming and

organizational rules become more important. Simple naming rules can help quite a bit. In many of our larger applications we used consistent naming conventions such as prefixing all roles with "has-" so that a quick glance would determine if "mother" is the class of female persons who have at least one child or if "mother" is the binary relation between children and their mothers. If "mother" is the class and "has-mother" is the binary relation, then there is no ambiguity in the name. Environments that are used to generate ontologies can easily support and enforce naming conventions.

Many conventions may be useful and/or required. For example, if one is conducting business in a state that taxes differently on luxury vs. non-luxury items, then it is critical for all products for sale to have a way of figuring out if they are luxury items or not. Otherwise, pricing cannot be done on orders. Required properties can be clearly stated in a published ontology construction methodology and also may be enforced by ontology editing and analysis environments. We are exploring options of supporting required naming conventions in commercial ontology environments.

Some operational guidelines include:

- Incorporate an ontology merging tool into the environment.
- Employ focus of attention techniques to help assist humans in their analysis tasks.
- Provide a diagnostic tool set that analyzes provably incorrect information.
- Provide a diagnostic tool that suggests possible problems.
- Provide a critique analysis that suggests potentially better representation style.
- Provide a version control mechanism.
- Provide support for extending upper and lower level ontologies.
- Make conventions explicit on how the ontology is constructed and how one should make extensions – including naming conventions, organizational principles, what new levels mean (and when they should be added or deleted), when partitions are used and how they are added/modified), etc.

3 Discussion

Others have preceded us in the areas of conceptual modeling. We embrace most of the guiding principles laid down in the fields of conceptual graphs, description logics, general frame systems, object-oriented modeling, knowledge acquisition, etc. for building consistent, and principled ontologies. We are also not the first to focus in the areas of conceptual modeling for large or distributed environments (consider the work done at Cycorp or the work of [Swartout, et. al, 1996, Erikson, et. al, 1999, Fridman, et. al, 1999, Oliver, 1999], etc.). We may have had a wider range of experiences in our ontology environments including multiple description logic environments along with frame system and theorem provers and also may have had broader ranging needs across the application families. The work has been driven by large corporations, such as AT&T, Lucent, NCR, and recently Cisco, Internet startups, and government-funded academic research. Our recent work on merging and ontology

diagnostics attempts to synthesize the learnings across all of the domains and may provide a unique combination of facilities aimed at today's world of extensible, distributed ontologies with emphasis on diagnostics and ontology critiquing. Our goal with this paper is to provide a focus for the evolving field of distributed ontology management. Another goal is to attempt to encourage the publication of suggested protocols for distributed ontology generation and maintenance.

4 Summary

In this paper, we have focused on issues that have greater importance when one is managing ontologies in a distributed manner. We have attempted to synthesize our guidelines from a wide range of applications developed in varying knowledge representation and reasoning environments with a breadth of ultimate application goals. We presented some structure in guidelines for conceptual modeling in distributed environments and welcome evolutionary suggestions to help refine the desiderata.

Acknowledgements

Our work on ontologies for government analysts was supported by the DARPA High Performance Knowledge Base program and built on the shoulders of much previous work done by many researchers at Stanford (particularly Fikes, Rice, and Farquhar), SRI, and at other HPKB contractor's organizations. Our work on description logic applications was supported by Bell Laboratories and AT&T Laboratories Research and also built on previous work by our main collaborators: Borgida, Brachman, Patel-Schneider, and Resnick. Our work on knowledge-enhanced search was funded by AT&T and was heavily influenced by our many applications partners (particularly Resnick, Beattie, Manning, Fuoss, Solomon, Moore, and Maulitz).

This paper is dedicated in loving memory to my father, Richard Charles McGuinness, who passed away during the preparation of the paper. His consistent love, support, and optimistic spirit were a key to my success in work and in life. His joyous spirit lives on but his physical presence will be sorely missed.

References

1. Franz Baader, Alex Borgida, Ralf Kuesters, and Deborah L. McGuinness. "Matching in Description Logics". In Journal of Logic and Computation -- Special Issue on Description Logics. Volume 9, number 3, June 1999.
2. Alex Borgida and Deborah L. McGuinness. "Inquiring about Frames." In Proceedings of Fifth International Conference on the Principles of Knowledge Representation and Reasoning Cambridge, Massachusetts, November, 1996. Morgan Kaufmann. Also appears in Proceedings of International Workshop on Description Logics, Cambridge, Mass., November 1996.

3. Alex Borgida, Enrico Franconi, Ian Horrocks, Deborah L. McGuinness, and Peter Patel-Schneider. “Explaining ALC subsumption” Proceedings of the International Workshop on Description Logics - DL-99, pp. 33-36, Linköping, Sweden, July 1999.
4. Ronald J. Brachman, Peter G. Selfridge, Loren G. Terveen, Boris Altman, Alex Borgida, Fern Halper, Thomas Kirk, Alan Lazar, Deborah L. McGuinness, Lori Alperin Resnick. “Knowledge Representation Support for Data Archaeology.” In Proceedings of the First International Conference on Information and Knowledge Management, November, 1992.
5. Ronald J. Brachman, Alex Borgida, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lori Alperin Resnick. “Living with CLASSIC: When and How to Use a KL-ONE-Like Language.” In Principles of Semantic Networks: Explorations in the representation of knowledge, ed. John Sowa. San Mateo, California: Morgan Kaufmann, 1991, pages 401--456.
6. Brickley and Guha, editors. Resource Description Framework (RDF) Schema Specification 1.0 W3C Candidate Recommendation. March, 2000. http://www.w3.org/TR/2000/CR-rdf-schema-20000327/.
7. Vinay K. Chaudhri, Adam Farquhar, Richard Fikes, Peter D. Karp, & James P. Rice. “OKBC: A Programmatic Foundation for Knowledge Base Interoperability”. Proceedings of the Fifteenth National Conference on Artificial Intelligence; Madison, Wisconsin; July 26-30, 1998. Also, KSL Technical Report KSL-98-08.
8. Paul R. Cohen, Robert Schrag, Eric Jones, Adam Pease, Albert Lin, Barbara Starr, David Easter, David Gunning and Murray Burke. 1998. The DARPA High Performance Knowledge Bases Project. In Artificial Intelligence Magazine. Vol. 19, No. 4, pp.25-49.
9. DMOZ – Open Directory Project. http://www.dmoz.org.
10. H. Eriksson, R. W. Fergerson, Y. Shahar, & M. A. Musen. Automatic Generation of Ontology Editors. Twelfth Banff Knowledge Acquisition for Knowledge-based systems Workshop, Banff, Alberta, Canada, 1999.
11. Adam Farquhar, Richard Fikes, and James Rice. “The Ontolingua Server: a Tool for Collaborative Ontology Construction”. In Proceedings of Knowledge Acquisition Workshop, Brian Gaines ed. Banff, Canada 1996.
12. N. Fridman Noy and Mark A. Musen. SMART: Automated Support for Ontology Merging and Alignment. In the Proceedings of the Twelfth Workshop on Knowledge Acquisition, Modeling and Management. Banff, Canada, 1999. Available as SMI technical report SMI-1999-0813
13. Mike R. Genesereth and Richard Fikes. “Knowledge Interchange Format”, Version 3.0 Reference Manual. Knowledge Systems Laboratory, KSL-92-86, 1992.
14. Robert MacGregor and Deborah L. McGuinness. “DARPA's High Performance Knowledge Base (HPKB) Program”. Proceedings of the International Workshop on Description Logics - DL-99, Linköping, Sweden, July 1999.
15. Deborah L. McGuinness. Ontology-enhanced Search for Primary Care Medical Literature. In the Proceedings of the International Medical Informatics Association Working Group 6-Medical Concept Representation and Natural Language Processing Conference, Phoenix, Arizona, December 16--19, 1999.
16. Deborah L. McGuinness. “Ontologies for Electronic Commerce”. Proceedings of the AAAI '99 Artificial Intelligence for Electronic Commerce Workshop, Orlando, Florida, July, 1999.
17. Deborah L. McGuinness. “Ontological Issues for Knowledge-Enhanced Search". In the Proceedings of Formal Ontology in Information Systems. June 1998. Also published in Frontiers in Artificial Intelligence and Applications, IOS Press. Washington, DC, 1998. Available from http://www.research.att.com/~dlm/findur .

18. Deborah L. McGuinness. “Explaining Reasoning in Description Logics". Ph.D. Thesis, Rutgers University, 1996. Technical Report LCSR-TR-277.
19. Deborah L. McGuinness and Alex Borgida. “Explaining Subsumption in Description Logics." In Proceedings of the 14th International Joint Conference on Artificial Intelligence, Montreal, Canada, August, 1995.
20. Deborah L. McGuinness, Richard Fikes, James Rice, and Steve Wilder. “An Environment for Merging and Testing Large Ontologies”. In the Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning (KR2000), Breckenridge, Colorado, USA 12-15 April 2000. Available from http://www.ksl.stanford.edu/software/chimaera.
21. Deborah L. McGuinness, Richard Fikes, James Rice, and Steve Wilder, ``The Chimaera Ontology Environment," To appear in the Proceedings of the The Seventeenth National Conference on Artificial Intelligence (AAAI 2000), Austin, Texas, July 30 - August 3, 2000.
22. Deborah L. McGuinness and Peter F. Patel-Schneider. “Usability Issues in Knowledge Representation Systems". In Proceedings of the Fifteenth National Conference on Artificial Intelligence, Madison, Wisconsin, July 1998. This is an updated version of “Usability Issues in Description Logic Systems" published in Proceedings of International Workshop on Description Logics, Gif sur Yvette, (Paris), France, September 1997.
23. Deborah L. McGuinness, Lori Alperin Resnick, and Charles Isbell. “Description Logic in Practice: A CLASSIC: Application." In Proceedings of the 14th International Joint Conference on Artificial Intelligence, Montreal, Canada, August, 1995.
24. Deborah L. McGuinness and James R. Wright. ``Conceptual Modeling for Configuration: A Description Logic-based Approach." In the Artificial Intelligence for Engineering Design, Analysis, and Manufacturing Journal - special issue on Configuration, 1998.
25. Deborah L. McGuinness and James R. Wright. “An Industrial Strength Description Logic-based Configurator Platform". IEEE Intelligent Systems, Vol. 13, No. 4, July/August 1998, pp. 69-77.
26. Diane Oliver. “Change Management for Shared and Locally Divergent Healthcare Terminologies.” In the Proceedings of the International Medical Informatics Association Working Group 6- Medical Concept Representation and Natural Language Processing Conference, Phoenix, Arizona, December 16--19, 1999.
27. Bill Swartout, Ramesh Patil, Kevin Knight and Tom Russ. “Toward Distributed Use of Large-Scale Ontologies.” In Proceedings of the Tenth Knowledge Acquisition for Knowledge-based Systems Workshop, November 9-14, 1996. Banff, Alberta, Canada.
28. Diane Vizine-Goetz and Joan S. Mitchell. 1996. "Dewey 2000." Annual Review of OCLC Research July 1995. Dublin, OH: OCLC Online Computer Library Center, Inc.:16–19. Available at: http://www.oclc.org/oclc/research/publications/review95/part1/vizine.html.
29. James R. Wright, Elia S. Weixelbaum, Karen Brown, Gregg T. Vesonder, Steven R. Palmer, Jay I. Berman, and Harry H. Moore, “A knowledge-based configurator that supports sales, engineering, and manufacturing at AT&T network systems," in Proceedings of the Innovative Applications of Artificial Intelligence Conference, pp.183--193, 1993.

Discovery of Class Relations in Exception Structured Knowledge Bases

Hendra Suryanto and Paul Compton

Department of Artificial Intelligence, School of Computer Science and Engineering,
University of New South Wales, Sydney, Australia
{hendras,compton}@cse.unsw.edu.au

Abstract. Knowledge-based systems (KBS) are not necessarily based on a well-defined ontologies. In particular it is possible to build very successful KBS for classification problems, but where the classes or conclusions are entered by experts as free-text sentences with little constraint on textual consistency and little systematic organisation of the conclusions. This paper investigates how relations between such 'classes' may be discovered from existing knowledge bases. We have based our approach on KBS built with Ripple Down Rules (RDR). RDR is a knowledge acquisition and knowledge maintenance method which allows KBS to be built very rapidly and simply by correcting errors, but does not require a strong ontology. Our experimental results are based on a large real-world medical RDR KBS. The motivation for our work is to allow an ontology in a KBS to 'emerge' during development, rather than requiring the ontology to be established prior to the development of the KBS. It follows earlier work on using Formal Concept Analysis (FCA) to discover ontologies in RDR KBS.

1 Introduction

Most knowledge acquisition methodologies first build a model of domain knowledge before applying this to building a particular problem solver e.g KADS and CommonKADS [20], Protege2000 [24]. Although this approach facilitates re-use it does not overcome the knowledge-acquisition and maintenance bottleneck and these problems are present both in the development of the ontology and consequent problem solver.

The RDR approach starts knowledge acquisition (KA) to build the problem solver immediately without any modelling apart from a simple attribute-value data representation [17]. Even the attribute-value representation can be developed while KA is in progress. The focus of the approach is to make the addition of each incrementally added piece of knowledge as simple and as reliable as possible. Although this approach facilitates KA and maintenance, it does not facilitate re-use because of the lack of an ontology.

Richards and Compton [18] have previously applied formal concept analysis (FCA) to develop conceptual models from RDR systems and for example have discovered interesting models from a 60 rule blood-gas KB which was itself part of a larger RDR KB. The RDR approach, along with other KBS approaches, allows a

B. Ganter and G.W. Mineau (Eds.): ICCS 2000, LNAI 1867, pp. 113-126, 2000.
© Springer-Verlag Berlin Heidelberg 2000

class to be covered by a number of disjoint rules. The approach further encourages this in that RDR does not allow modification of existing rules and all rules have to be added as new disjuncts. As will be discussed below a new rule is added by the expert selecting conditions to cover a misclassified case but exclude other cases previously correctly classified[9], [10].

For example, one of the classes in a real-world RDR KBS consists of a disjunction of 74 rules with each rule a conjunction of conditions. If this is displayed using an FCA tool, the class is scattered across many different nodes and it is difficult to get an overall impression[18]. In a simple example in figure 2, the class BUS is in two nodes. The real-world KB we are using has 3710 rules and 101 classes. If we try to display such a knowledge base using FCA tools the problem is exacerbated. Mineau et al. argue that in the FCA framework *"the choice of conceptual scales depends on the purpose of the analysis. For instance, other attributes may be suitable for market analysis than for decision support"* [15]. This may well be appropriate in exploring data. However, we assume that since we are dealing with rules rather than raw cases, the relevant attributes have already been identified and extracted. Our aim is to discover the appropriate ontology given that the relevant attributes (and values) are already well-identified.

A second aspect of the problem is that in a real-world system, attributes are multi-valued rather than boolean. Rule conditions can subsume each other, be disjoint etc. For example age >10 subsumes age > 50, whereas age >40 and age <10 are disjoint. Our method needs to not only combine information about classes from across the knowledge base, but to address the way in which conditions based on multi-valued attributes interrelate (see Fig. 1).

Class: Satisfactory lipid profile previous raised LDL noted
(LDL <= 3.4) AND Triglyceride is NORMAL) AND (Max(LDL) > 3.4) **OR**
((LDL is NORMAL) AND (Triglyceride is NORMAL) AND (Max(LDL) is HIGH)
AND (Net_Change(LDL) <0))

Fig. 1. An example of multi-valued attributes in RDR.

Finally, the problem is compounded, by the way in which the expert adds conclusions. When an RDR KB makes an error, the expert identifies the attributes and values that justify a different conclusion. In adding the conclusion, the expert can select from a list of pre-existing conclusions organised into broad categories, but is also free to simply type in a new conclusion – and often does this. In medical pathology result interpretation, the evaluation domain here, the conclusions added by the pathologist may provide advice to the referring clinician on patient diagnosis, management, how treatment is progressing, whether the tests ordered were appropriate, what tests might still be necessary or any combination of the above. It is quite clear to both the expert and the receiver of the advice what information is being provided in the free text interpretation, but these interpretations are a long way from the well-defined classes of a formal ontology. A task analysis would assess this domain as a classification problem, but this does not imply well-defined classes. Hence, the problem is not only that disjuncts for a class may be scattered across the KB, but that the same class may be represented by different text strings or a text string may cover a combination of different classes. Some examples are given in Table 6.

The question perhaps arises of whether it would be more appropriate to start with a well-developed ontology, as suggested by most KA researchers. There have been a number of RDR papers over the years presenting data on the practical advantages of the incremental KA approach provided by RDR. Commercial RDR systems are now available from Pacific Knowledge Systems Pty. Ltd. In evaluation studies of this software a large Australian pathology laboratory has developed about 8 RDR KBs ranging from a few hundred to 7000 rules. These are all in routine use and to date have processed over 500,000 laboratory reports, handling much of the laboratory's report output. They were all built by pathologists or laboratory scientists after about 2 hours training. The laboratory has documented that KA time was about one minute per rule and appropriateness of the interpretations is >98%. (It is a trivial matter to improve this further if really needed.) Since this development has been incremental it has had minimal impact on the laboratory's normal work-flow and has been a trivial addition to the normal duties of staff. These results have not yet been formally published, but they strongly confirm the previous RDR research results. Rather than leading to the conclusion that it would be better to start with a well-developed ontology, these results raise the question of whether in practice it may be simpler to re-develop rather than re-use. However, if we can discover the ontologies implicit in these incrementally developed systems, we may be able to have the best of both worlds. The 3710 rule KB we are using here is an early version of the 7000 rule PKS KB.

In proposing techniques to discover relations between rule conclusions (i.e. the class relation model (CRM)) we consider three basic relations: subsumption, mutual-exclusivity and similarity. The application of these is shown in the following simple RDR example. RDR will be explained in more detail later. For the moment it is sufficient to note that a class in RDR is the set of disjoint rule paths each giving the same conclusion and that a rule path consists of all the conditions from a rule's predecessor rules plus the conditions of the rule itself. The rule paths (disjuncts) for this example are derived from the RDR KBS in figure 2. They are:

- rule path 2 : class VECHICLE ← hasEngine=YES.
- rule path 3 : class VAN ← hasEngine=YES, passenger>5.
- rule path 4 : class BUS ← hasEngine=YES, passenger>10
- rule path 5 : class SPORT_CAR ← hasEngine=YES, passenger=2
- rule path 6 : class TRUCK ← weight>2000kg, speed>30km/h
- rule path 7 : class BUS ← weight>2000kg, speed>30km/h, publicTransport=YES
- rule path 8 : class GLIDER ← moveOn=AIR
- rule path 9 : class GROUND_VEHICLE ← moveOn=GROUND

The basic idea of the technique is to combine all rules of the same class and then compute a quantitative measurement *(from 0 to 1)* of each relation *(subsumption, mutual-exclusivity, similarity)* between two classes. We use this quantitative measurement as an informal confidence measure as to whether these relations exist. For example the class VEHICLE subsumes class BUS with degree of confidence 0.9; class VAN and class SPORT_CAR are mutually exclusive to each other with degree of confidence 1.0; class BUS and TRUCK have a degree of similarity of 0.3. (The derivation of these quantities is discussed in detail later)

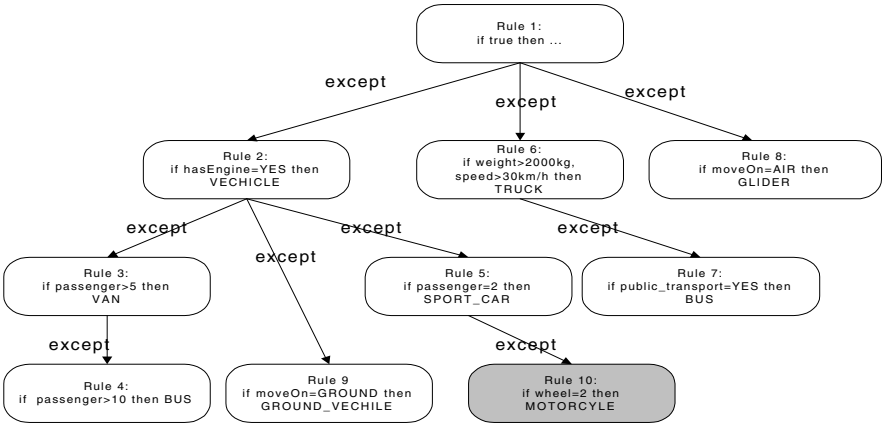


Fig. 2. An example of a simple MCRDR KBS.

The *subsumes* quantitative measure provides information on whether a class almost subsumes another class (for example, class VEHICLE subsumes class BUS with degree of confidence 0.9. Using this quantitative measure we can say class VECHICLE almost subsumes class BUS, rather than simply saying class VEHICLE subsumes class BUS or class VEHICLE does not subsume class BUS.

These measures find the application in real examples such as: *class The mild hypothyroidism may contribute to hyperlipidaemia* is subsumed by *class Hypothyroidism may exacerbate hyperlipidaemia* with degree of confidence 0.75. Boolean *(true/false)* values also are inappropriate for the relations *subsumption, mutual-exclusivity* and *similarity* in real domains.

In the 3710 rule application we considered, we found only 4 subsumption relations with degree of confidence 1.0; 181 mutually-exclusive relations with degree of confidence 1.0 and no similarity relations with degree of confidence 1.0.

Since we learn from rules, not from cases, our method does not need to consider which attributes are relevant. Gaines [6] argues that a rule in a knowledge base is worth more than many cases for learning. We adopt the same viewpoint.

In the next section we outline RDR and FCA. We then discuss the derivation of the quantitative measures. Finally we describe our experiment to test this technique on the 3710 rule KB.2. Ripple Down Rules and Formal Concept Analysis

1.1 Ripple Down Rules

RDR is an attempt to deal with the problem that experts never explain how they reach a conclusion, rather they justify why their conclusion is appropriate and this justification varies with the context in which it is given [3]. With RDR the expert's task is to correct errors made by the KBS. The expert decides a case should have a different conclusion from the one given by the KBS and justifies this by indicating features in the case which distinguish it from a case for which the conclusion provided by the KBS would have been appropriate. This results in the refinement

structure shown in figure 2 illustrating where a rule (*rule 10*) has been added to correct an error.

During RDR inference, data is passed down the tree and if a rule fires, its conclusion is added to the case. Child rules are only evaluated if a parent rule is satisfied. However if one or more child rules fires, their conclusions are added to the case and the parent conclusion removed. Their children are then evaluated. This process continues down each path until a leaf node is reached or no child rule fires. There are a number of RDR structures. This one is known as multiple classification RDR (MCRDR) and was the structure used for the 3710 rule KB.

To correct one of the conclusions given, the RDR system will add a new child rule under the rule that gave the wrong conclusion. The expert selects conditions for this rule which distinguish the case from other cases for which the parent rule conclusion was correct. In practice, the cases used for this are the cases for which other rules have been previously added, known as 'cornerstone cases'. The case for which this new rule is being added will also be stored as a future cornerstone case. When a new rule is added, any 'cornerstone cases' that can reach the parent rule are retrieved. The expert adds a rule which excludes all of these cornerstone cases or can assign the conclusion as an extra conclusion for a particular case. In practice, even with hundreds of cornerstone cases that might reach the rule, the expert selects conditions to make a sufficiently precise rule after no more than two or three cornerstone cases have been considered . This is essentially a verification and validation technique incorporated into knowledge acquisition [13].

Figure 2 illustrates a simple example of an MCRDR KB. For example if we have a case: hasEngine=YES, passenger=2, moveOn=GROUND, this case will fire rule 2, rule 5 and rule 9. In MCRDR if a child rule is fired, the child conclusion replaces the parent conclusion. Thus 3 nodes fire but 2 conclusions are given: SPORT_CAR and GROUND_VEHICLE. If we get a new case: MOTORCYCLE ← hasEngine = YES, passenger = 2,wheel = 2, this case will be misclassified as SPORT_CAR, so the expert uses this to build rule 10 as a child of rule 5 and the case is stored as a cornerstone case for future rule addition.

The RDR exception structure provides a compact knowledge representation [2], [8], [14], [19], [21]. Further, despite the random order in which cases are presented, and the likelihood of experts providing less that ideal rules, manually built RDR systems produce compact KBs [4], [11], [22]. Gaines has also generalized RDR to Exception Directed Acyclic Graphs (EDAGs). He argues EDAGs are more compact than RDR since the EDAG graph structure avoids the fragmentation problem of the RDR tree structure [7].

Initial RDR development was concerned with classification tasks, first single and later multiple classification. RDR has since been extended to configuration [16], heuristic search [1], document retrieval [12] and a more general RDR system for construction tasks has been proposed [5]

1.2 Formal Concept Analysis

Formal Concept Analysis, as proposed by Wille [23], offers a formalisation of mathematizing concepts as units of thought constituted by their extension and intension. FCA defines a formal context as a triple (G,M,I), where G is a set of objects, M is a set of attributes and I is a binary relation between G and M, that is I

$\subseteq G \times M$. For example the sentence, *the object g has the attribute m,* in the FCA framework can be written as gIm ($\Leftrightarrow$ (g,m) $\in$ I).

In our example MCRDR KBS (see figure 2), G is {vehicle, aeroplane, ground vehicle, sport car, bus, truck, van} and M is {hasEngine, weight>2000, speed>30, moveOnGround, moveOnAir, passenger=2, passenger>5, passenger>10, publicTransport}. That is, MCRDR attributes have many values, giving a *many-valued context*. To obtain a concept lattice we need to convert these attributes to single valued boolean attributes giving a *single-valued context*.

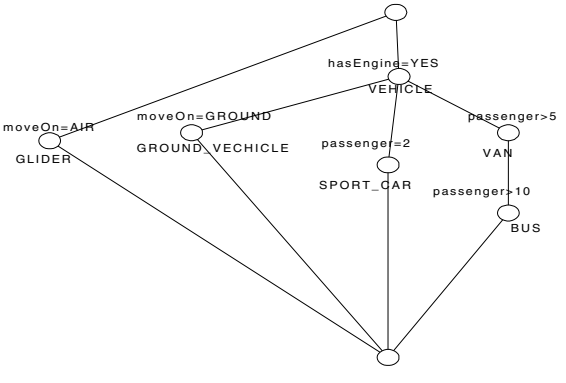
A formal concept of a formal context (G,M,I) is defined as a pair (A,B), where $A \subseteq G$ and $B \subseteq M$ such that (A,B) is maximal with the property $A \times B \subseteq I$. The set A is the *extent* of the formal concept (A,B) and the set B is the *intent* of the formal concept (A,B).

The subconcept-superconcept relation is defined by $(A_1, B_1) \leq (A_2, B_2)$: $\Leftrightarrow A_1 \subseteq A_2$ ($\Leftrightarrow B_1 \supseteq B_2$). The concept lattice is a complete lattice which consists of the set of all concepts of a context (G,M,I) together with the order of the relation $\leq$.

In the formal concept analysis framework, the choice of attributes depends on the purpose of the analysis. In the example in figure 3 we analyse the passenger capacity and the type of movement.

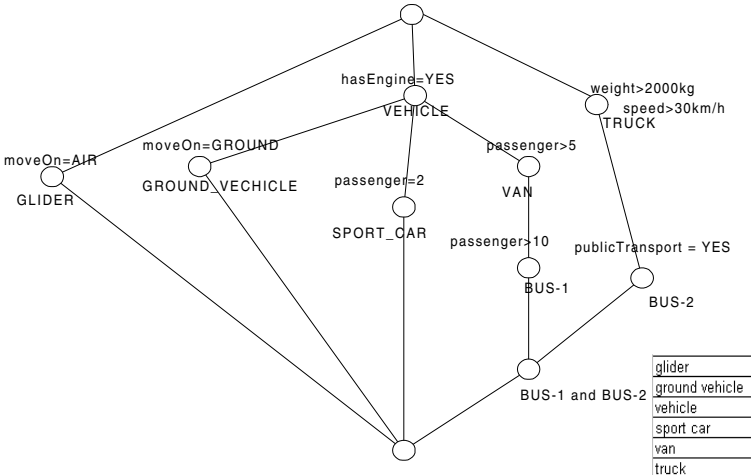
In the example in figure 4, we derive a formal context for all the attributes, since the concept BUS is created from two different contexts which we rename as BUS-1 and BUS-2. Both of these describe the same concept but in different contexts, so they appear as two formal concepts. An imaginary *concept BUS-1 and BUS-2* is a sub concept of BUS-1 and BUS-2. Since we have many classes that consist of disjunctions of many rules, then those classes will scattered in many nodes of the concept lattice if displayed in a general context (of all attributes).

In a real-world MCRDR KBS the class Satisfactory lipid profile is a disjunction of 2 rules (see Fig 1) similar to the toy example of the class BUS.



	has engine	move on air	move on ground	passenger > 5	passenger = 2	passenger > 10
glider		X				
ground vehicle	X		X			
vehicle	X					
sport car	X				X	
van	X			X		
bus	X			X		X

Fig. 3. A concept lattice and formal context table



	has engine	move on air	move on ground	passenger > 5	passenger = 2	passenger > 10	weight>2000	speed>30	public transport
glider		X							
ground vehicle	X		X						
vehicle	X								
sport car	X				X				
van	X			X					
truck							X	X	
bus-1	X			X		X			
bus-2							X	X	X
bus-1 & bus-2	X			X		X	X	X	X

Fig. 4. A concept lattice and formal context table

2 The Class Relations Model

The measures we have developed are strictly heuristic. Other superior and perhaps more well founded measures may be possible. The studies here represent simply a first attempt at carrying out this type of analysis and integrating across disjuncts. The second point to note is that the purpose of these relations is to deal with non-boolean data.

The technique is based on set theory. If we have two sets A and B, then the following indicates the possible relations between them according to the three relations we are considering.

- $A \subset B$
- $B \subset A$
- $A \cap B \neq \varnothing$
- $A \cap B = \varnothing$
- $A = B$
- $A \neq B$

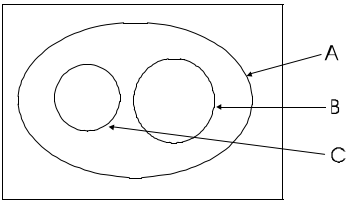


Fig. 5.

Fig.5 shows a particular example where: $B \subset A$, $C \subset A, B$, $B \cap C = \varnothing$. That is, B and C are mutually exclusive.

Let X be a class in the MCRDR framework. $\{X_0 \ldots X_m\}$ is a set of rules which have class X as their conclusion. $\{X_{i0} \ldots X_{in}\}$ is a set of conditions for rule X_i, n+1 is the number of possible distinct attributes in the domain. However, in a give rule some of these conditions may be empty because the rule may not contain every possible distinct attribute. Secondly, if X_{ij} and Y_{ij} stand for individual conditions in rule paths for the classes X or Y, although X_{ij} and Y_{ij} both refer to the same attribute they may be different conditions . For example `age=50` and `age>55`..

In the MCRDR framework the class is given as a disjunction of rule paths [18]. If m is the number of rule paths for class X, then:

$$\text{class } X = \bigvee_{i=0}^{m} \left(\bigwedge_{j=0}^{n} X_{ij} \right) \quad (\mathbf{1})$$

If Y is also a class, we could define a similarity measure as follows:

$\text{Sim}(X_{ij}, Y_{ij}) = 0$ if X_{ij}, Y_{ij} are different
$\text{Sim}(X_{ij}, Y_{ij}) = 1$ if X_{ij}, Y_{ij} are the same

If α is the set of distinct attributes in rule path X_i and β is the set of distinct attributes in rule path Y_i, then we can define:

$$\text{Similar}(X_i, Y_i) = \frac{\sum_{j=0}^{n} \text{Sim}(X_{ij}, Y_{ij})}{|\alpha \cup \beta|} \quad (\mathbf{2})$$

Table 1. An example of applying $\text{Similar}(X_i, Y_i)$

Measure	rule-path	has engine	move on	passe-nger	weight	speed	public transp.	Value
Similarity	Bus-1	YES		>10				
	Truck				>2000	>30		
		different		different	different	different		0/4
Similarity	Bus-2				>2000	>30	YES	
	Truck				>2000	>30		
					same	same	different	2/3

Table 1 shows the derivation of Similar(Bus-1, Truck) = 0/4 and Similar(Bus-2, Truck) = 2/3. Fig 6 suggests how we can find a similarity measure between 2 classes. Each node in Fig 6 corresponds to a rule path. v stands for the Similar() function relating two rule paths or disjuncts. (In later similar diagrams v stands for the Subsume() and MutualEx() measures.) Fig 6 shows that Class X is the disjunction of nodes 1,2 and 3 and Class Y is the disjunction of nodes 4 can 5. We propose that ClassSimilarity(X,Y) = (v1 + v2 + v3) / 3 ,where we choose the v such that all nodes are covered by at least one edge and the sum of v (eg. v1 + v2 + v3) is maximal (eg. ClassSimilarity(Bus,Truck) = (0/4 + 2/3)/2).

We can similarly define a subsumption measure as follows with X_{ij} and Y_{ij} again standing for individual conditions in a rule path for the relevant class as above. As before:

Sub (X_{ij} ,Y_{ij}) = 0 if X_{ij} does not subsume Y_{ij}
Sub (X_{ij} ,Y_{ij}) = 1 if X_{ij} subsumes or is the same as Y_{ij} (for example A>5 subsumesA>10)

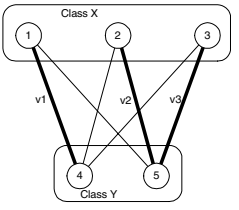


Fig. 6. Similarity

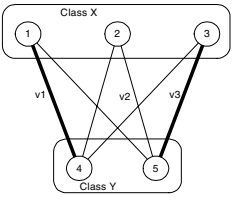


Fig. 7. Subsumptions

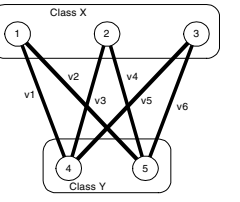


Fig. 8. Mutual-exclusivity

If α is the set of distinct attributes in rule X_i, and β is the set of distinct attributes in rule Y_i, then we can define:

$$\text{Subsume}(X_i, Y_i) = \frac{\sum_{j=0}^{n} \text{Sub}(X_{ij}, Y_{ij})}{|\alpha \cup \beta|}, \text{ if } \alpha \leq \beta \qquad (3)$$

$$\text{Subsume}(X_i, Y_i) = 0, \text{ if } \alpha > \beta$$

Table 2. An example of applying Subsume(X_i,Y_i).

Measure	rule-path	has engine	move on	passe-nger	weight	speed	public transp.	Value
Subsump-tion	Vehicle	YES						
	Bus-1	YES		>10				
		same		sub				2/2
Subsump-tion	Vechicle	YES						
	Bus-2				>2000	>30	YES	
		not			sub	sub	sub	3/4

As before, Table 2 shows the derivation of Subsume(Vehicle,Bus-1) = 2/2, Subsume(Vechicle,Bus-2) = 3/4. Given that function Subsume() measures a degree of confidence that the first rule path subsumes the second rule path, Figure 7 suggests how we can find a subsumption measure between the 2 classes. It shows Class X as a disjunction of node 1,2 and 3 and Class Y as a disjunction of nodes 4 and 5 (each node again represents a rule path). We compute ClassSubsume(X,Y) = (v1 + v3) / 2. We choose the v such that all nodes of class Y are covered by at least one edge and the sum of v (eg. v1 + v3) is maximal. We then compute TotalSubsume (X,Y) =

ClassSubsume(X,Y) - ClassSubsume(Y,X). If the value of TotalSubsume(X,Y) is negative, then we exchange X and Y (see Fig. 7) (eg. ClassSubsume(Vehicle, Bus) = (2/2+3/4) / 2).

We can define a mutual exclusivity measure as follows, with X_{ij} and Y_{ij} again standing for individual conditions in a rule path. As before:

Mut (X_{ij} ,Y_{ij}) = 0 if X_{ij} and Y_{ij} are not mutually exclusive.
Mut (X_{ij} ,Y_{ij}) = 1 if X_{ij} and Y_{ij} are mutually exclusive (for example A>5 and A<2)

We can then define:

$$\text{MutualEx}(X_i, Y_i) = 1 \text{ , if at least one of } \text{Mut}(X_{ij}, Y_{ij})=1$$
$$\text{MutualEx}(X_i, Y_i) = 0 \text{ , otherwise} \quad (4)$$

Table 3. An example of applying MutualEx(X_i, Y_i).

Measure	rule-path	has engine	move on	passe-nger	weight	speed	public transp.	Value
Mutual exclusivity	Bus-1	YES		>10				
	SportCar	YES		=2				
		not me.		mut. Ex				1
Mutual exclusivity	Bus-2				>2000	>30	YES	
	SportCar	YES		=2				
		not me.		not me.	not me.	not me.	not me.	0

As before, Table 3 shows the derivation of MutualEx(Bus-1, SportCar)=1, MutualEx(Bus-2, SportCar)= 0. If the function MutualEx() measures a degree of confidence that the first rule subsumes the second rule, figure 8 suggests how we can find a mutual exclusivity measure between the 2 classes. It shows Class X as a disjunction of nodes 1,2 and 3 and Class Y as a disjunction of nodes 4 and 5. We compute ClassMutualEx(X,Y) = (v1 + v2 +v3 +v4 + v5 + v6) / 6. X and Y are mutually exclusive if and only if all nodes of X and Y are mutually exclusive with respect to each other (see Fig. 8) (eg. ClassMutualEx(Bus, SportCar) = (1+0)/2).

3 Experimental Results

The initial results are from the 9 rules of the toy example in figure 2. As can be seen from these results (Table 4), the measures work reasonably well

The mutual exclusivity and subsumption relations make reasonable sense but nothing should be concluded from the actual values.. Of course all these classes have some similarity as they are all vehicles

Of more interest are the results from the 3710 rules (see Table 5) of a real pathology system. The results shown in each table are the five class pairs with the highest similarity, subsumption, or mutual exclusivity measures. The two columns, Class 1 and Class 2, indicate the classes to which the relation applies. The numbers

correspond to the classes in Table 6. At this stage, we are not in a position to make a medical or pathology assessment of these results, but note the following examples:

Satisfactory lipid profile, unless patient has CHD subsumes *Lipid profile improving If patient has CHD aim for LDL < 2.6 mmol/L. Diabetes and hypertriglyceridaemia noted*, with degree of confidence 1.0.

Raised cholesterol and triglycerides. The mild hypothyroidism may contribute to hyperlipidaemia is similar to *Raised cholesterol and triglycerides. Hypothyroidism may contribute to hyperlipidaemia* with degree of similarity 0.7.

Raised triglycerides with abnormal fasting glucose. Suggest glucose tolerance test is mutually exclusive to *Hypercholesterolaemia. Suggest glucose tolerance test*.

These all seem reasonable. However, a number of the other examples do not make as much sense to a lay reader.

Table 4. Some results from the 9 rule toy KBS.

Similarity >=0.3		
Vechicle	Van	0.500
Vechicle	Sport car	0.500
Van	Sport car	0.500
Bus	Truck	0.330

MutualExclusivity > 0		
Van	Sport car	1.000
Ground vechicle	Glider	1.000
Bus	Sport car	0.500

Subsumption >= 0.875		
Vechicle	Van	1.000
Vechicle	Sport car	1.000
Vechicle	Bus	0.875

Table 5. Some results from the 3710 rule real-world KBS

Sort-by	Class 1	Class 2	1 subsume 2	mutual	similar
subsume	2	17	1.000	0.000	0.400
big-5	12	58	1.000	0.000	0.444
	24	64	1.000	0.667	0.166
	61	64	1.000	0.000	0.286
	24	61	0.939	0.667	0.456

Sort-by	Class 1	Class 2	1 subsume 2	mutual	similar
similar	29	70	0.705	0.000	0.753
big-5	9	31	0.625	0.000	0.625
	43	48	0.136	0.000	0.592
	52	59	0.000	0.392	0.591
	13	40	0.029	0.402	0.580

Sort-by	Class 1	Class 2	1 subsume 2	mutual	similar
mutual-ex	4	68	0.036	1.000	0.145
big-5	6	18	0.118	1.000	0.235
	6	25	0.108	1.000	0.212
	6	43	0.082	1.000	0.189
	6	44	0.000	1.000	0.210

Table 6. Some classes (clinical interpretations of results) from the 3710 rule Pathology KBS.

#Id	Interpretations or classes
2	Satisfactory lipid profile, unless patient has CHD.
4	Raised triglycerides with abnormal fasting glucose. Suggest glucose tolerance test.
6	Raised cholesterol and triglycerides. Suggest 2 hr post-prandial glucose and TSH if not previously done.
9	Raised cholesterol and triglycerides. Suggest TSH if not previously done. Normal glucose levels suggest diabetes has been excluded.
12	Borderline cholesterol level. Suggest glucose tolerance test in view of raised fasting glucose (if not a known diabetic).
13	Raised cholesterol and triglycerides with low HDL. Suggest repeat fasting glucose in 12 months.
17	Lipid profile improving. If patient has CHD aim for LDL < 2.6 mmol/L. Diabetes and hypertriglyceridaemia noted.
18	Hypercholesterolaemia. Diabetes and hypothyroidism are excluded.
24	Mild hypercholesterolaemia with raised fasting glucose. Suggest repeat glucose tolerance test in 12 months.
25	Hypercholesterolaemia. Aim for LDL < 3.4 mmol/L (<2.6 if patient has CHD). Suggest 2 hr post-prandial glucose to exclude diabetes.
29	Raised cholesterol and triglycerides. The mild hypothyroidism may contribute to hyperlipidaemia.
31	Marked hypertriglyceridaemia. Suggest glucose tolerance test.
40	Raised cholesterol and triglycerides. Suggest repeat fasting glucose in 12 months.
43	Borderline HDL. Otherwise satisfactory lipid profile.
44	Satisfactory lipid profile, unless patient has CHD (aim for LDL < 2.6).
48	Low HDL. Otherwise satisfactory lipid profile.
52	Raised cholesterol and triglycerides. Glucose level to follow.
58	Raised cholesterol and triglycerides with raised fasting glucose. Suggest glucose tolerance test.
59	Raised cholesterol and triglycerides. TSH and glucose to follow.
61	Suggest repeat glucose tolerance test in 12 months.
64	Raised triglycerides. Impaired glucose tolerance noted.
68	Hypercholesterolaemia. Suggest glucose tolerance test.
70	Raised cholesterol and triglycerides. Hypothyroidism may contribute to hyperlipidaemia.

4 Further Work

Firstly, there may be other similar types of measure that may be more useful in relating classes as a whole. We will be considering a number of these. Secondly, although this approach allows classes to be considered as a whole across a knowledge base there are still too many relations to provide a reasonable overview of the domain. One possible approach to make the results more accessible is to consider patterns of

relations rather than individual relations. In the results above, individual relations are ranked or a cut-off can be considered. However, we could also look for patterns where say, A subsumed both B and C and B and C were mutually exclusive and a whole hierarchy with this pattern presented to the user. Different ontological patterns could be considered. Preliminary results using this approach suggest that it may be useful for extracting more interesting information. In a similar vein it will be interesting to form concept clusters based on appropriate cut-offs for the relations, and then merge the classes and re-express the relations. Finally, the present technique only considers the conditions in rule paths in determining the relations. It does not consider any other information about the classes that could be derived from the tree structure. For example: although the refinement structure for RDR does not indicate any ontological refinement, it does indicate that an expert thought a conclusion was inappropriate and so should be replaced by another. We are looking at the possible relations between conclusions that this would allow which can be combined with the idea of relations presented here. Finally, we would suggest again that there may be considerable value in attempting to discover ontologies from knowledge bases that are already developed or undergoing incremental development. Secondly, although this work is still in an early stage, we would claim that the techniques we have presented, or similar techniques, seem likely to be suitable for ontology discovery from large, real-world knowledge bases.

Acknowledgements

This work is supported by the Australian Research Council and an Asian Development Bank scholarship. The knowledge base used has been provided by Pacific Knowledge Systems. The authors would like to thank Jane Brennan and Rex Kwok for helpful discussion.

References

1. Beydoun, G. and A. Hoffmann. NRDR for the Acquisition of Search Knowledge. in 10th Australian Conference on Artificial Intelligence. 1997.
2. Catlett, J. Ripple Donw Rules as a Mediating Representation in Interactive Induction. in Proceedings of the Second Japanese Knowledge Acquisition for Knowledge Based Systems Workshop. 1992. Kobe, Japan.
3. Compton, P. and R. Jansen., A philosophical basis for knowledge acquisition. Knowledge Acquisition, 1990. **2**: p. 241--257.
4. Compton, P., P. Preston, and B. Kang. The Use of Simulated Experts in Evaluating Knowledge Acquisition. in 9th AAAI-sponsored Banff Knowledge Acquisition for Knowledge Base System Workshop. 1995. Canada.
5. Compton, P. and D. Richards. Extending Ripple Down Rules. in Australian Knowledge Acquisition Workshop 99. 1999. UNSW.
6. Gaines, B.R., An Ounce of Knowledge is Worth a Ton of Data: Quantitative Studies of the Trade-off between Expertise and Data based on Statistically Well-founded Empirical
7. Gaines, B.R., Transforming Rules and Trees into Comprehensible Knowledge Structures. 1995: AAAI/MIT Press.

8. Gaines, B.R. and P.J. Compton. Induction of Ripple Down Rules. in Fifth Australian Conference on Artificial Intelligence. 1992. Hobart.
9. Kang, B. and P. Compton. Knowledge acquisition in context: the multiple classification problem. in Proceedings of the Pacific Rim International Conference on Artificial Intelligence. 1992.
10. Kang, B., P. Compton, and P. Preston. Multiple classification ripple down rules: Evaluation and possibilities. in 9th AAAI-sponsored Banff Knowledge Acquisition for Knowledge Based Systems Workshop. 1995.
11. Kang, B., P. Compton, and P. Preston. Simulated Expert Evaluation of Multiple Classification Ripple Down Rules. in 11th Banff knowledge acqusition for knowledge-based systems workshop. 1998. Banff: SRDG Publications, University of Calgary.
12. Kang, B., et al., A help desk system with intelligent interface. Applied Artificial Intelligence, 1997. 11((7-8)): 611-631.
13. Kang, B.H., W. Gambetta, and P. Compton, Verification and Validation with Ripple Down rules. International Journal of Human-Computer Studies, 1996. 44: p. 257-269.
14. Kivinen, J., H. Mannila, and E. Ukkonen. Learning Rules with Local Exceptions. in European Conference on Computational Theory. 1993.
15. Mineau, G., G. Stumme, and R. Wille. Conceptual Structures Represent by Conceptual Graphs and Formal Concept Analysis. in Conceptual Structures: Standards and Practices. 7th International Conference on Conceptual Structures, ICCS'99. 1999. Blacksburg, VA, USA, July 1999: Springer.
16. Ramadan, Z., et al. From Multiple Classification RDR to Configuration RDR. in 11th Banff Knowledge Acquisition for Knowledge Base System Workshop. 1998. Canada.
17. Richards, D. and P. Compton. Knowledge Acquisition First, Modelling Later. in 10th European Knowledge Acquisition Workshop. 1997.
18. Richards, D. and P. Compton. Uncovering the conceptual models in RDR KBS. in International Conference on Conceptual Structures ICCS'97. 1997. Seattle: Springer Verlag.
19. Scheffer, T. Algebraic foundations and improved methods of induction or ripple-down rules. in 2nd Pacific Rim Knowledge Acquisition Workshop. 1996.
20. Schreiber, G., B. Wielinga, and J. Breuker, KADS A Principled Approach to Knowledge-Based System Development. 1993: Academic Press.
21. Siromoney, A. and R. Siromoney, Variations and Local Exception in Inductive Logic Programming, in Machine Intelligence - Applied Machine Intelligence, K. Furukawa, D. Michie, and S. Muggleton, Editors. 1993. p. 213 - 234.
22. Suryanto, H., D. Richards, and P. Compton. The automatic compression of Multiple Classification Ripple Down Rule Knowledged Based Systems: Preliminary Experiments. in Knowledge-based Intelligence Information Engineering Systems. 1999. Adelaide.
23. Wille, R., Knowledge Acquisition by Methods of Formal Concept Analysis, in Data Analysis, Learning Symbolic and Numeric Knowledge, E. Diday, Editor. 1989, Nova Science: New York. p. 365 - 380.
24. William E. Grosso, H.E., Ray W Fergeson. Knowledge Modeling at the Millenium (The Design and Evolution of Protege-2000). in Twelfth Workshop on Knowledge Acquisition, Modeling and Management. 1999. Banff, Alberta, Canada.

CGs Applications : Where Are We 7 Years after the First ICCS?

Michel Chein and David Genest

LIRMM (CNRS and Université Montpellier II)
161, rue Ada, 34392 Montpellier cedex 5 - France
{chein, genest}@lirmm.fr

Abstract. The traditional distinction between theories (developed to tackle theoretical problems) and applications (based on theories and realized to help a user) is blurred in computer science. To experiment their theories, computer scientists often write programs. This paper focuses on the features that make such a program an application (also in its software engineering meaning). The discussion is more specifically aimed at artificial intelligence applications and especially conceptual graphs applications presented in ICCS papers, and the importance of applications for a scientific domain.

1 Introduction

In the traditional distinction between scientists, technologists, and engineers, engineers design and realize applications, technologists design and realize tools from which applications are built, and from theories developed by scientists tools and applications are built. This is a very simplified view of how things are going in informatics. As in any young science, the three aspects are closely tied, and, often, the cycle is short: scientists develop theories, and sometimes tools, in order to tackle actual problems (sometimes a same person realizes the whole work: e.g. Knuth is a brilliant example). Usually, as scientists, we present things from theory to applications: scientific hypotheses, theoretical results, tools and applications [6].

In this paper, we only consider the final goal: applications. Application is understood in its usual software engineering meaning, that is to say an application is a program of a specific type. Use of CGs for research purposes in another scientific domain is not considered here, even if this kind of research work could be called, and are sometimes called, CGs applications.

The plan of the paper is the following: in section 2 we propose rough distinctions between several types of internal and external features of an application. We discuss in section 3 some external features of an application, and we distinguish between an application and not yet finalized programs such as prototypes, or mock-ups, or experiments, etc. In section 4 some specific aspects of an AI application are presented. Section 5 is a brief survey of CGs applications presented from ICCS'93 to ICCS'99, and our own situation, quite similar to other CG research groups involved in application development, is presented in section 6. Section 7 is a (some little pessimistic) conclusion.

B. Ganter and G.W. Mineau (Eds.): ICCS 2000, LNAI 1867, pp. 127–139, 2000.
© Springer-Verlag Berlin Heidelberg 2000

2 Internal and External Characteristics

Roughly speaking, an application has internal characteristics and external characteristics. Two types of internal characteristics can be distinguished. A first type of internal characteristics is related to software engineering. An application must have precise specifications, its conditions of use must be described, it must be reliable and robust, it must be validated, documentation must be available, its evolution must be anticipated, and so on. Obviously, this does not always correspond to reality, and many applications, even with million copies sold, do not fulfill these conditions. These not finalized applications eventually sold can be explained by marketing choices of some companies, especially their will to occupy the market. Also, this illustrates the fact that it is difficult to provide an application perfectly robust and running on a large variety of different platforms. We will not consider here software engineering criteria. There are two main reasons for that. Firstly, we do not have the information needed for analyzing, according to the software engineering viewpoint, "applications" which interest us here, that is to say CGs applications. Secondly, there are too few "real"(in traditional software engineering meaning) CGs applications.

The second type of internal characteristics is related to the scientific field that is the basis of the software construction. We specify in section 4, what we understand by an AI software.

A program may be presented in a session "applications" of an artificial intelligence conference, or a CGs conference, but it does not become, ipso facto, an AI or a CGs application. In the same way, any "running" program, respecting software engineering state of the art, is not necessarily an application. External characteristics, those which consider the software from outside, like a black box, are essential for a software which claims to be an application.

3 External View of an Application

In this paragraph a set of general characteristics of an application is proposed. These characteristics can be used as a filter for separating applications from other programs such as: prototypes, tools, toy applications, mock-ups, ... These features also show that qualifying a software as an application is more a job for ergonomists and specialists in the application domain rather than for informaticians.

3.1 An Application Should Aid Solution of Actual Problems

Building a diagnostic system in medicine, or a document retrieval system, are clearly actual problems which can lead to applications. It is not always so easy to say what an actual problem is. The following excerpt from the editorial of the AI special issue "40 years later" [5] does not help very much : *"The first set of papers provide a look at how AI is being used in "applications", problems embedded in and defined by a real world environment"*. Many pure scientific problems are

embedded in and defined by real world environments, for instance SAT is such a problem. If somebody provides an efficient algorithm for solving large boolean equation sets, it will be immediately implemented in tools that will be used for designing optimal digital circuits: a today very real world problem. Numerous scientific and technical problems, in informatics as in many other domains, are embedded in and defined by real world environment. Real world problems can not be opposed to theoretical problems but rather to gratuitous problems. An application should be useful.

3.2 An Application Is Useful for People Other than Designers of This Application

Generally, an application is worth to socio-economic or cultural domains. When there is a large gap between the application domain – for instance medicine – and informatics, it is quite easy to say what an application is. But, computer science is itself a source for real problems, and computer technology and engineering use numerous applications. Often, there exists a large hierarchy of programs upon which an application is built, but, and this is characteristic for an application, it must have an usage value for the so-called end users (although it does not necessarily have an exchange value: see free software).

3.3 Tool and Application Layers

An operating system is a useful software, and it is usually called a tool and not an application. Above an OS there are assemblers, interpreters, compilers, programming languages, ..., a hierarchy of layers. Some tools at level i can be considered as applications for the *(i-1)*-ith level. As in the OSI model, the term application could be reserved for the last layer, the one of end users. For any layer there is also an historical aspect (its life-cycle) : experiment, toy application, mock-up, prototype, alpha version, beta version, release candidate, and release, even beta release candidate exists !

Possible aims of an experiment can be to show that an application project is worthwhile, or to prove that some tools are operational, or to confirm a scientific hypothesis, etc. Sometimes a mock-up does not implement any computing mechanisms but only the external behaviour of a program. Furthermore, generally, it is not at the final scale, and transition from a toy version to pertinent scale version often reserves unpleasant surprises. Normally, a prototype is at correct scale, but some functionalities are lacking. Building them can lead to serious difficulties. Experiments, mock-ups, prototypes should not be given to end users.

Thus, “what are the end users ?” and “what are the end user tasks which could be aided by the program ?” are the very first questions to ask a program which aims to be an application.

3.4 Different Sorts of Usefulness

Not all programs dealing with a real world problem are applications, even if they satisfy "buyers". Let us consider for instance a therapy adviser in medicine. The program can be worth to disease specialists, not because they follow the therapeutic advices given by the program, but because during the program design process they have acquired a better understanding of how they make therapeutic decisions.

Possible objectives of a toy application are to confirm or reinforce a scientific hypothesis, or to show that a tool is operational, such a program should be better called an experiment than an application. Usefulness of a mock-up may be to convince decision-makers to allow development of a "true" application. In the same way, a prototype can be useful for different persons (designers, decision-makers, developers ...). Thus, even if toy applications, mock-ups, and prototypes are important steps in an application construction process, they are not applications. Most of them do not lead to applications.

3.5 An Application Should Be Used by Several Users

The number of users of an application can vary from a handful (e.g. for a highly specialized design aided molecular biology system) to million users (e.g. for a web navigator). A great number of users can considerably increase the difficulty of the final step of an application constructing process. Indeed, even if the design has been very carefully made, it has been realized by a small team which can be unaware of some needs or behaviors of million people. The alternative cycle between tests and modifications or new functionalities may take a long time before convergence, and this time is needed not only for fixing bugs but also for obtaining a satisfactory version for a well chosen end user sample. Furthermore, usually, an unmaintained program does not live a very long time. All that implies that software engineering internal characteristics are fundamental for a program in order to have a chance to become an application.

3.6 Assessment of an Application

An assessment protocol should be designed at the very beginning of design process of an application. Choices for assessing an application are not always easy to make: how to choose user samples ? for what tasks ? what criteria must be satisfied ? what parameters can be measured ? ...

Many parts of an assessment process are specific to the application under consideration. In any case, assessment does not mainly fall within the competence of informaticians, but falls within the competence of ergonomists and application domain specialists.

3.7 Application Cost

It can be easily deduced from the preceding lines that the initial development costs to obtain the first working program are just a fraction of the investments

necessary for the final application release. “Say what is your project cost and I will tell you if there is any chance you will construct an application !”

4 Artificial Intelligence Applications

In order to specify what are the specific internal and external features of an AI application, one must very succinctly present our viewpoint concerning AI.

4.1 Some Words about AI

AI is the part of informatics dealing with the specific domain of human intelligence considered in a restrictive meaning: the domain of reasoning intelligence. That is to say the capability needed for grasping and processing concepts and ideas, for acquiring and using knowledge, for solving problems, for learning , ... significantly, the two problems proposed by Turing into his foundational paper are: playing chess, and learning a natural language. There are other kinds of intelligence, for instance the practical intelligence of homo faber, which are not directly relevant to AI which is mainly concerned by the essential homo sapiens sapiens faculty. One can consider three facets in informatics: computing science (the study of algorithms), computer science (the study of computing hardware and software systems and devices), information processing science (the study of operational informatic models of different real world entities). For AI, the computing science facet is mainly specialized in computational logics, and the computer science facet is specialized in *LISP*, *Prolog*, massively parallel systems, etc. From the viewpoint of information processing, the main goal of AI is to construct operational informatic models of intelligent human activities, or, more modestly, to build programs which can aid people for intellectual tasks. Reasoning capabilities – for instance inference reasonings – on large knowledge bases are needed for such systems. Knowledge and reasoning aspects exist in any information processing problem, but there are different sorts of knowledge. For instance, physical-mathematical knowledge are different from phenomenological knowledge. The latter is relevant to AI. It can not be represented in the same way as, for example, a physical law. Generally, phenomenological knowledge is of a discursive nature, so it is vague, ambiguous, contextual, incomplete, and it must be elicited, elaborated, limited, specified, before any automatic processing. These characteristics of AI relevant knowledge entail that assessment of AI applications is a specifically difficult problem.

4.2 Components of an AI Application

One can roughly distinguish four internal basic components in an AI application.

– The deeper component is a *knowledge base management system* (KBMS), having itself three main parts: a knowledge representation language, an inference engine, and a control mechanism. This fundamental part, which constitutes the kernel of an AI application, may be more or less generic, it can

be a tool (but there do not exist many generic KBMS comparable to Data Base Management Systems). Specific AI difficulties are due to the fact that interesting AI problems are usually undecidable (or semi-decidable), and NP-complete or NP-hard problems are simple AI problems !

- *Interfaces* are the more external component. Developers, system managers and end users need specific interfaces. Due to the peculiar nature of involved knowledge, knowledge elicitation and acquisition are made easier if there is a strong correspondence between interface objects and their internal representation. In the same way, if the application is intended to be an aid to an intellectual work, it is important that the system is able to represent its inferences in an understandable manner for the end user. End users have to be able to compare their reasonings with the system reasonings.
- The *programming environment* is very important for the engineering qualities of the software built. All usual software qualities (portability, robustness, reusability, efficiency, ...) are required for an AI application, but always for the same reasons due to the peculiarities of knowledge considered in AI, evolutivity is a fundamental feature, and every programmer knows that it is more difficult to construct a program with loose specifications than a very constrained one. During the early years of AI the situation was quite simple: *LISP* was dominating. Then came logic programming and *Prolog*, object paradigm and object *LISPs*, *C++*, and, more recently *Java*. Programming paradigms embedded in these languages have important consequences on the resulting application. Even if declarative languages are wished by AI developers, today they widely use object programming languages. This point will not be discussed in this paper. It is an important topic and it deserves a whole study. Let us only mention two general reasons explaining the success of object oriented language: the existence of class libraries such as *STL*[1] or *LEDA* [18] for *C++*, or the *Java API*, and the fact that object oriented languages are effective for the production of applications that can be (relatively) easily maintained and extended. It is necessary to also choose tools (compilers, libraries, etc.) easily available, on many platforms, and maintained.
- The *networking environment* is also crucial, because many applications need distributed data in various forms. Client-server applications become more and more popular, specifically because operating systems integrate facilities for accessing networks. Even if client stations become more powerful, applications need more processing power, which has to be done on consequently dimensioned servers. It is worth to take into account distributed information and/or distributed processing, at the beginning of the development of tools or applications. The choice of the development environment is important because some of them are not very well adapted to communications between processes or computers. Communication facilities explain the success of *C* and *C++* – closely tied to *Unix*, one of the first system integrating mech-

[1] The "Standard Template Library" is a library of data structures and algorithms integrated in the ANSI standard for C++.

anisms of communication – and of *Java*, because the *Java API* contains classes facilitating the access to shared resources.

4.3 Assessment of Applications

One of the distinctive features of an AI application is that it must have reasoning capabilities. Evaluation of such an application is then more difficult than evaluation of an application that only manages data or makes numeric calculus. For instance, the evaluation of a database management system (DBMS) focuses mainly on the run time of queries and the behavior of the application on large bases. Evaluation of DB applications does not take into account relevance of the given answers, because such an application must provide complete answers, otherwise the application is buggy.

In the case of an information retrieval application, a difficult problem is to know what information is relevant for a given question. Time of physical access to the data is less significant than for a DBMS. Indeed, an information retrieval system (IRS) provides an "intellectual access" to the base, in opposition to a DBMS which provides a physical access, and an evaluation of the interest of data accessed (and how it satisfies a user) requires measurements more difficult than measurement of physical access time. When we talk about speed of an IRS, it is necessary to take into account the number of decisions that have to be done by a user for obtaining an answer appropriate to him rather than the number of milliseconds necessary for processing a query [3]. The quality of the results provided to a query is thus essential, because a system slower than another one (because it performs more reasoning) but which provides more satisfactory results, may allow a user to obtain more quickly (with less operations) information he (she) is looking for.

Precision and *recall* are the two parameters usually used in order to assess the quality of results provided by an IRS. Precision is the percentage of relevant retrieved documents, whereas recall is the percentage of the total potentially relevant documents effectively retrieved. To estimate these rates, it is thus necessary to ask a user if he (she) estimates that given documents are relevant or not. For precision, one "just" has to ask him (her) for each retrieved document if that document is relevant or not. On the other hand, recall depends on the quantity of relevant documents in the base, then it is necessary to ask the user about the relevance of all documents, and this is unrealistic because a serious evaluation can be done only on large bases. Thus, the recall parameter can only be estimated.

Moreover, this is only an evaluation of *one* given query for *one* given user, which can vary over time, and generalizing such a result for a system evaluation is hazardous. A complete evaluation of an IRS requires to collect the satisfaction of many users on many queries on a large base. With these conditions, it should not be astonished that reliable evaluations of IRS are so rare [4,20]. Difficulty of evaluation of such systems, and thus difficulty of comparison of various approaches, explains proliferation of approaches and systems.

However, IR seems to be one of the simpler problems of AI because an IRS "just" have to return a list of information relevant to a given query. For more complex applications, such as industrial design or natural language processing, evaluation is undoubtedly more complicated.

5 CGs Applications

Many different program sorts are described in ICCS papers. One can mention:

- implementation of algorithms or methods (constrained maximal join, graph or subgraph isomorphism, CGs matching, classification and clustering, generalization, inference or learning algorithms, ...);
- programs implementing new capabilities added to already existing tools;
- numerous CG-editors, more or less integrated in existing tool;
- general tools;
- and, finally, programs that interest us here, those close to applications (in the specific meaning used in this paper).

The classifications adopted, for program sorts as well as for domains, are certainly questionable. Only titles and ICCS ref. are given. Reader in a hurry can directly jump to the end of this section (discussion).

Natural Language Processing

This domain is very active since the first ICCS. Papers in NLP sessions are often purely descriptive, sometimes programs are mentioned, but they can hardly be called prototypes (of applications), generally, they only illustrate how CGs are used for modeling natural language notions. Among the implemented approaches one can find:

- studies of specific linguistic notions. e.g. Computational Processing of Verbal Polysemy with Conceptual Structures [16], Analysis of Task Oriented conversation into CG structures [17];
- modules to be added in existing system. e.g. Representing natural language causality in CGs [11] to be included in a general system Kalipsos now abandoned, Italian linguistic analysis module to be included in NOMOS [12], Natural Language Text processing and the Maximal Join Operator to be included in CoKEMan [14], Using the CG Operations for Natural Language Generation in Medicine [13] to be included in the GALEN project;
- independent modules. e.g. Extraction of Implicit and Explicit Knowledge from NL Texts [12], Implementing a Semantic Lexicon [17];
- complete systems. e.g. RECIT a Multilingual Analyser of Medical Texts [12], DB-MAT: Knowledge Acquisition, Processing and NL Generation Using CGs [14].

Information (or Document or Knowledge) Retrieval Systems

Many experiments have been realized in this domain. Most of the following works are, or could lead to, applications: Linguistic Processing of Text for a Large Scale Conceptual IRS [12], Conceptual structures and Structured Documents [14], An experiment in document retrieval using CGs [15], a prototype based on CoGITo has been developed, CGKAT for organizing and retrieving knowledge, this system is using tools such as Wordnet, Thot, and CoGITo [14,15], CGs for Corporate Knowledge Repositories [15], The CG Mars Lander [15], Using Viewpoints and CG for the Representation and Management of a Corporate Memory in Concurrent Engineering [16], CG for Representing Business Processes in Corporate Memories [16], Embedding Knowledge in Web Documents: CGs versus XML-based Metadata Languages [17].

Knowledge Acquisition and Representation

Ten or so papers, more or less far from an application, can be gathered in this class: Acquiring temporal knowledge from schedules [11], Modelling and Simulating Human Behaviours with CGs [15], Menu based interfaces to CG: the CGLex approach [15], Knowledge extractor [15], Complex Modelling Constructs in MODEL-ECS [15], A semantic validation of CGs [16], Knowledge Querying in the CG Model: the RAP Module [16], Multikat a Tool for Comparing Knowledge of Multiple Experts [16], A CG based behavior system extraction [17].

Only five papers have answered the interesting SCG-1 initiative [17]. Among them, only two have built a program for solving the problem. Carrying on the SCG initiative was not possible this year, though we could expect more answers than last year.

Requirements

This seems to be an important potential application domain for CGs. Let us mention some papers: Automatic Integration of Digital System Requirements Using Schemata [13], Service trading using conceptual structures [13], Generic Trading Service in Telecommunication Platforms [15], Applying CG Theory to the User-Driven Specification of Network Information Systems [15], A CG-Based Behavior Extraction System [17].

Miscellaneous

For sake of completeness, let us mention two dedicated systems: Using CGs as a common Representation for Data and Configuration in an Active Image Processing System [17], On Developing Case-Based Tutorial Systems with CGs [17].

Discussion

Most of the papers presented at ICCS (even during application sessions organised almost each year) are aiming at testing ideas, approaches, algorithms, or techniques. Generally, they are experiments and there are only a few prototypes: after all this is quite normal for scientific conferences. Nevertheless, system evaluations are often too superficial, even if assessing an AI system is rather difficult, and, what is more annoying, many systems are not sufficiently described, they are not easily available, and they appear and disappear very fastly.

Distinction between applications and tools are not always easy to do, but in order to rapidly program experiments generic tools are needed. Besides Peirce workbench, the following systems aim to be general CGs tools: CGP [11], UDS [12], CoGITo and CoGITANT [14,16], Prolog++ and CG-Prolog [12], CGKAT [14,15], WebKB [15], WebKB-GE [16] , Deakin Toolset and EGP and CGKEE [15], Notio [17], Synergy [17]. These tools are more or less finished, more or less re-usable, rarely maintained. How many of these tools are still maintained ? The web page on conceptual graphs[2] presents only five of them. The "Conceptual Graphs Homepage"[3] only adds one to the list. Whereas the old page "CG Tools"[4] presented twenty six tools ! What did thes e tools become ?

If there is no more tool session, we can worry for forthcoming applications. Without good tools (generic, reliable, available on various systems, documented, maintained, efficient, etc.) it is difficult to develop applications. Let us insist on the importance of the SCG-1 initiative [17]. One needs benchmarks. And it is not astonishing that the two groups which have presented an automatic solution of the problem are developing general CGs tools since a long time.

6 And Us, Wherein Are We ?

In this section we describe situation in our CG research group, with respect to the "application" facet.

We are working on a document retrieval project, called *Mogador*, with ABES (agency of french university libraries). The goal of this work is to produce experimental tools, based on CGs and *Rameau*[5], for indexing and retrieving documents by their subject, and to evaluate, in collaboration with librarians, results provided by this approach. If the evaluation is positive, i.e. if the experiment is sufficiently convincing for decision-makers, we hope that our ideas will be taken into account by the team that will design the next generation of the french document retrieval system. *Mogador* is an illustration of the distinction made in introduction, it is clearly not an application (in the software engineering meaning we have adopted in this paper), but it is an experiment applying CGs to document retrieval.

[2] `http://www.bestweb.net/~sowa/cg`

[3] `http://www.cs.uah.edu/~delugach/CG`

[4] `http://www.cs.uah.edu/~delugach/CG/CGTools.html`

[5] A thesaurus used for the indexing of documents in almost all french public libraries.

We are just beginning to work, with other academic and industrial teams, on a project called *OPALES*. This project aims to build a website prototype for cooperative work on audiovisual documents. INA (french audiovisual national institute) is project manager. If the prototype is successful different applications are planned, for instance a site dedicated to teachers and another one to researchers. Industrial teams of the *OPALES* project will be in charge of the development of these applications. *OPALES* and *Mogador* CGs components are quite the same, and the four internal basic components presented in paragraph 4.2 are the following.

- *Knowledge base management system.* The CGs model used is based on nested positive graphs with coreferent links [7,8] and CG rules [19]. This language is generic and it is used in every project of our research group. The inference engine is based on projection. The control mechanism is not generic and we are defining an ad hoc mechanism for *Mogador*.
- *Interfaces.* We have decided to use the CG model at all levels: theoretical model provides internal representation, reasonings are carried out by graph operations , and end users also handle CGs through GUI. We believe that it is an essential feature for building intelligent cooperative systems, i.e. for knowledge programming. Indeed there must be an "isomorphism" between what is seen by the user and the formal model, to enable faithful modeling of the actual data and problems, and to correctly interpret the results and the computation. There must also be an "isomorphism" between what is seen by the user and how objects and operations are implemented, in order to understand why and how results have been obtained. The most secure way to do this is to use a homogeneous model: the same kind of objects occur at each level [6].
- *Programming environment.* The *Java* language, has been chosen for portability reasons, and it is used to develop the user interface. On the other hand, to carry out the processing and to handle large quantities of data, we use *C++* for performance reasons, as well as CoGITaNT [9], the library of *C++* classes developed within our team since 1994 (CoGITaNT is an evolution of CoGITo [10]). This library is a general platform for handling CGs.
- *Networking environment.* We have chosen to develop various modules according to a client-server architecture. For instance, in *Mogador*, the server, based on the CoGITaNT platform, manages ontological knowledge (more than 400000 types of concepts) and graphs, and performs graph operations. Clients are able to edit the graphs in graphic form, to execute operations and consult results of operations. Extensions we have developed have been designed in a generic way in order to be re-usable in other works. Thus, for responding to the SCG-1 initiative and for representing the *Sisyphus-I* problem [2,1], we used our graph editor developed originally for *Mogador*, and we have benefitted from the extensions recently applied to CoGITaNT for the inference leading to the resolution itself.

Our research group is clearly in the same position as other CG groups, we have realized many experiments, and we hope to be, right now, in position for constructing a prototype.

7 Conclusion

For its durability, any AI domain needs successful applications. There are at least two different and important reasons for that. Firstly, from a scientific point of view, AI is an empirical science (two main reasons have been sketchily presented in this paper: nature of AI problems which often are undecidable, so only incomplete solutions can be looked for, and the phenomenological nature of the knowledge AI is concerned by). AI needs experiments in order to confirm scientific hypotheses. Secondly, AI community has made many promises since its beginning, but only some of them have been honored. "Buyers" of AI want to see effective realizations and not only great ideas or sophisticated theories.

As far as we know, if there exist operational CGs applications there probably exist only a handful. Nevertheless, we believe that there exist mean term potential application domains, especially: corporate memory, information retrieval, specifications of complex systems. In order to build applications, efficient tools are needed, and, what is rather distressing, we could deliver exactly the same discourse that Bob Levinson and Gerard Ellis had done when they launched the Peirce project in 1992 ! (Furthermore, efficient tools are based on efficient algorithms, but this domain is not currently active in the CG community).

We, CG community, have difficulties for analyzing, synthesizing, accumulating our knowledge (may be should we come back to dedicated workshops ?). Honestly answering title question of this paper could be: we have made numerous experiments, most of the time without drawing serious conclusions, we have put our feet on a large area, but the distance from our starting point is rather short. Optimistics could say that it is only because our domain is still in its infancy.

References

1. Jean-François Baget, David Genest, and Marie-Laure Mugnier. Knowledge acquisition with a pure graph-based knowledge representation model - Application to the Sisyphus-I case study. In *Proceedings of the 12th Workshop on Knowledge Acquisition, Modeling and Management (KAW '99), Banff, Canada*, volume 2, pages 7.1.1–7.1.20, 1999.
2. Jean-François Baget, David Genest, and Marie-Laure Mugnier. A pure graph-based solution to the SCG-1 initiative. In ICCS'99 [17], pages 355–376.
3. David C. Blair. *Language and Representation in Information Retrieval.* Elsevier Science Publishers, 1990.
4. David C. Blair and M. E. Maron. An evaluation of retrieval effectiveness for a full-text document-retrieval system. *Communications of the ACM*, 28(3):289–299, 1985.

5. Daniel J. Bobrow and J. Michael Brady. Artificial intelligence 40 years after. *Artificial Intelligence*, 103(1-2):1–4, 1998.
6. Michel Chein. The CORALI project: From conceptual graphs to conceptual graphs via labelled graphs. In ICCS'97 [15], pages 65–79.
7. Michel Chein and Marie-Laure Mugnier. Positive nested conceptual graphs. In ICCS'97 [15], pages 95–109.
8. Michel Chein, Marie-Laure Mugnier, and Geneviève Simonet. Nested graphs: A graph-based knowledge representation model with FOL semantics. In *Proceedings of the 6th International Conference "Principles of Knowledge Representation and Reasoning" (KR'98), Trento, Italy*, pages 524–534. Morgan Kaufmann Publishers, 1998.
9. David Genest and Éric Salvat. A platform allowing typed nested graphs: How CoGITo became CoGITaNT. In ICCS'98 [16], pages 154–161.
10. Ollivier Haemmerlé. *CoGITo : une plateforme de développement de logiciels sur les graphes conceptuels*. PhD thesis, Université Montpellier II, France, 1995.
11. *Conceptual Graphs for Knowledge Representation, proceedings of the first International Conference on Conceptual Structures (ICCS'93), Quebec City, Canada*, volume 699 of *LNAI*. Springer-Verlag, 1993.
12. *Conceptual Structures: Current Practices, proceedings of the second International Conference on Conceptual Structures (ICCS'94), College Park, USA*, volume 835 of *LNAI*. Springer-Verlag, 1994.
13. *Conceptual Structures: Applications, Implementation and Theory, proceedings of the third International Conference on Conceptual Structures (ICCS'95), Santa Cruz, USA*, volume 964 of *LNAI*. Springer, 1995.
14. *Conceptual Structures: Knowledge Representation as Interlingua, proceedings of the fourth International Conference on Conceptual Structures (ICCS '96), Sydney, Australia*, volume 1115 of *LNAI*. Springer, 1996.
15. *Conceptual Structures: Fulfilling Peirce's Dream, proceedings of the fifth International Conference on Conceptual Structures (ICCS '97), Seattle, USA*, volume 1257 of *LNAI*. Springer, 1997.
16. *Conceptual Structures: Theory, Tools and Applications, proceedings of the sixth International Conference on Conceptual Structures (ICCS '98), Montpellier, France*, volume 1453 of *LNAI*. Springer, 1998.
17. *Conceptual Structures: Standards and Practices, proceedings of the seventh International Conference on Conceptual Structures (ICCS'99), Blacksburg, USA*, volume 1640 of *LNAI*. Springer, 1999.
18. Kurt Mehlhorn and Stefan N her. *LEDA - A Platform for Combinatorial and Geometric Computing*. Cambridge University Press, 1999. http://www.mpi-sb.mpg.de/LEDA/leda.html.
19. Éric Salvat and Marie-Laure Mugnier. Sound and complete forward and backward chaining of graph rules. In ICCS'96 [14], pages 248–262.
20. E. M. Voorhees and D. K. Harman, editors. *Proceedings of the seventh Text REtrieval Conference (TREC-7), Gaithersburg, USA*, NIST Special Publication 500-242, 1998. http://trec.nist.gov/.

The Engineering of a CG-Based System: Fundamental Issues

Guy W. Mineau

Université Laval
Quebec City, Quebec
Canada, G1K 7P4
mineau@ift.ulaval.ca

Abstract. This paper presents some important issues that a knowledge engineer must consider when developing a conceptual graph (CG) based system, particularly in the context of a deductive data/knowledge base. These issues entail fundamental representation choices that must be made prior to the development of the system. Of course, inevitable consequences follow and delimit the scope of the system in terms of its representational and inferential capabilities. However, for industrial development, a less ambitious but realistic approach is often more suited to the kind of constraints usually imposed by the market place: feasibility, scalability, simplicity, interpretability, portability, partial knowledge, time to market, etc. This paper presents and discusses some of these issues and sets the stage for an in-depth discussion pertaining to the development of CG-based systems for industrial applications, particularly for applications where a CG system provides the conceptual level of knowledge organization functionalities required by an information system.

1 Looking at the Heart of CG Systems

Any system is developed in three layers: external, conceptual and implementation. The external layer meets the needs of the end-user by allowing him to see only what is appropriate and relevant to his needs, i.e., usually a subset of the functionalities for which the system was built. It provides him with an adapted interface to the system. To do so, it uses a carefully defined mapping onto the conceptual layer. The conceptual layer is the conceptual space in which the knowledge engineer designs a solution space within a generic problem-solving framework, for a particular problem space. Issues like what information is relevant to reach a solution state, how should that information be represented, what operators are required to process it and in the end provide the necessary support to implement the sought-for functionalities, must be dealt with at this level. It describes how a domain is modeled so that related problems can be solved using some problem-solving framework. Finally, the implementation layer maps the conceptual layer onto operational technology. This three-layer architecture allows the mappings between the layers to provide

B. Ganter and G.W. Mineau (Eds.): ICCS 2000, LNAI 1867, pp. 140-156, 2000.
© Springer-Verlag Berlin Heidelberg 2000

independence of each layer, allowing their updating without causing a global revision of the whole system. Each layer can thus evolve at its own pace.

Any conceptual graphs (CG) system may also be seen as having three layers, although not as obvious as with database systems. The knowledge engineer is the link between all three layers. That sometimes causes confusion in CG related research. At times, the knowledge engineer is seen as a "super" user for which a particular interface to the system is required, one that allows the editing of the conceptual layer. In this case, that means CG editing facilities.

This research community at times proposed ways to express CGs at this level. For instance, let us remind the community of the notation introduced by [1] to represent functions in the CG formalism (see Figure 1) using a double-lined arrow to express the functional dependency of the last argument of a conceptual relation on its preceding arguments. The graph of Figure 1 states that a person has only one SIN number (Social Insurance Number).

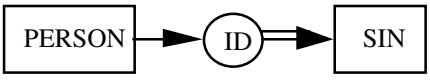


Fig. 1. A functional dependency expressed using a function symbol.

However, to be thorough, the introduction of the function symbol entailed the update of the conceptual layer so that the integration of this new symbol to the previously defined representation and inferential mechanisms of the CG notation would be complete. So [1] also proposed to modify the join operation to take into account the functional dependency that this new "function" symbol introduces. In brief, when all arguments of a function symbol f_1 are joined to some other set of concepts also arguments of a function symbol f_2 of the same type as f_1, then the output concepts of these two functions must also be joined. With our example, if we are to join the PERSON concept of Figure 1 to some PERSON concept in some other graph to which an ID function is also connected, then the two SIN concepts (that of the graph of Figure 1 and of the other graph) must be joined as well (see Figure 2).

Without that modification of the conceptual layer, the introduction of the function symbol would only have been an editing feature that would have had no impact on the enforcement of the constraint that it represents. Like many features in other conceptual modeling languages (like in UML for instance), it may have been useful to the knowledge engineer as a reminder of the functional dependency between some concepts, but it could have been written down on a piece of paper and that would have had the same usefulness to the knowledge engineer. To my opinion, the CG formalism offers much more than modeling constructs to build conceptual models, it provides a system of logic easy to use to describe conceptual models, plus it offers executable specification capabilities during the development cycle of a system. It would be vital that any researcher in this community that introduces some editing

facility to the CG theory also provides the updated conceptual layer so that its semantics is fully integrated to the operations of the system. Therefore, the angle that I chose to look at a CG system in this paper is through its conceptual layer, the layer in which its processing capabilities are defined. Unfortunately, the conceptual layer is often the hardest to grasp. The external layer offers an interface to the system and the implementation layer is based on technology. Thus they are both concrete, while the conceptual layer is used for conceptualization and can remain very abstract. Also, the conceptual layer is mostly used in modeling applications for information and database systems. Therefore, this paper views a CG system as providing the knowledge organization and management core for large industrial data intensive applications, pretty much like the core representation engine of a deductive database. Therefore, the recommendations made in this paper may not apply to other types of CG applications. Nevertheless, I hope that this paper will help trigger transfers to and from other uses of CG technology as encountered in the different research projects reported in the literature.

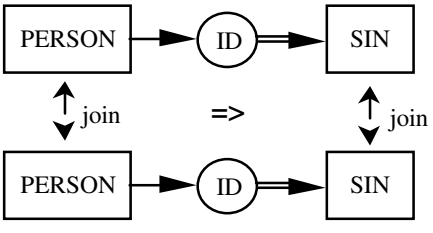


Fig. 2. Joining graphs containing function symbols.

From the point of view depicted in this paper, the conceptual layer of any CG based system is its main core. It forms a system of logic with its own representation constructs and operators. It provides the knowledge engineer with a unique way of designing a solution space for a particular problem space. With that regard, though a subset of the CG theory may be mapped onto first-order predicate logic (FOL), its approach to problem solving, mainly based on existentially quantified concepts interconnected in a semantic net and on a projection operation, is unique. In comparison, with FOL clausal formulae, the conceptual space that a knowledge engineer considers in order to provide a solution space to some problem deals with quantification, unification, and rule chaining and ordering. This conceptual space (for providing a problem solving system based on production rules) is totally different than that of CG systems. The knowledge manipulation operators being different, the approach used to look for a solution space is therefore also different than with other representation schemes. That surely has an impact on the cognitive process that the knowledge engineer undergoes.

For instance, [2] later introduced a representation framework for constraints based on the projection operation. Because the projection operation is at the heart of any CG

system, this approach provided a general and unified representation framework for all semantic constraints found in database literature today. In brief, a constraint graph is stated; it is known to be false. In order for all other graphs to be valid, there must be no projection from the constraint graph onto any graph in the CG system. Figure 3 shows a constraint graph stating that it is false that a person may have two distinct SIN numbers, i.e., there must not be any graph in the system on which the graph of Figure 3 could project itself. Here the reader should notice that we use the normal form assumption [3]: *concepts known to represent the same individual are joined*. The framework proposed in [3] also allows exceptions to constraint graphs to be represented. By providing a general mechanism to represent, among other things, functional dependencies, the modification done to the join operation in [1] now becomes obsolete because any operation that would result in a graph that would violate a constraint would be stopped. However, it is probably more convenient to the knowledge engineer to edit functional dependency constraints using the function symbol of [1] rather than producing graphs like the one in Figure 3. Again, a mapping between the external layer (which could use the function symbol) onto the conceptual layer (which would use constraint graphs) could be defined.

In the example above, it is obvious that the projection operation guided the engineering of a constraint representation mechanism adapted to the CG theory. This approach is totally different then the one adapted with clausal systems for instance.

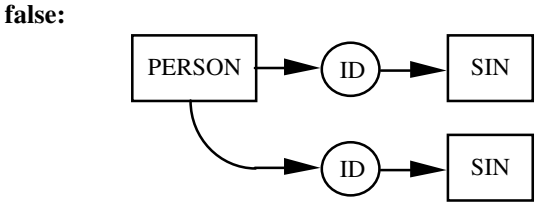


Fig. 3. A constraint graph enforcing a functional dependency.

2 Unicity of Representation: Simple Issues

At the conceptual layer, a CG system is a system of logic. At this point we would like to remind the reader that different logical representations of the same system are possible under the CG operators. As examples, let us cite the expansion and contraction operators that allow the granularity of representation to vary, adding more or less details into a graph. Let us mention also that through the type lattice, multiple representations of an individual concept are possible through the use of different concept types. All these different representations are logically equivalent. Unfortunately, logical equivalence may be a barrier to the engineering of a CG system instead of being its facilitator. The operators of the conceptual layer will have

to be mapped onto some implementation layer, i.e., on operational technology. That constraint requires that the conceptual layer be chosen with care: the mapping between the conceptual and implementation layers may not be as easy to achieve under one representation as it would be under another one. Though there is not much work today in the CG literature on the implementation layer of CG systems, it seems obvious that the representation of the conceptual layer must facilitate its mapping onto the implementation layer (for obvious feasibility and efficiency reasons). For instance, since the projection operation is at the heart of any CG system, any implementation layer must optimize its use.

2.1 The Normal Form Assumption

One way to facilitate the implementation of the projection operation is to have the graphs of the system represented under the normal form assumption (NFA) mentioned above. So we now introduce a first recommendation.

> R1. The conceptual layer of a CG based system should be represented under the Normal Form Assumption.

That will prevent the projection operation to be preconditioned on previous join operations that could arbitrarily have been done or not. That recommendation entails the most specific representation assumption (MSRA) below.

> R2. Any concept should be represented under its most specific form.

That is, if two concepts $[t_1: i_1]$ and $[t_2: i_1]$ exist with $t_1 < t_2$, then only the first concept should be used at the conceptual layer so that all identical concepts can automatically be joined under the NFA. Since we now deal with the representation of the conceptual layer, nothing is said about how a graph should be presented to a user through some interface. It may be the case that the user prefers to see $[t_2: i_1]$ instead of $[t_1: i_1]$. The independence of layers allows different representations to coexist, one for each layer if that is necessary, though it may introduce redundancy.

2.2 Standardization and Interpretability

In the same line of thought, the referent of a concept should be substituted by the object that it represents. That is, there are different ways to represent concepts. For instance, we could use [INTEGER: 8] or [INTEGER: #14] to represent the mathematical entity 8 (an integer number), if #14 is the referent to the object 8 stored somewhere in memory. Even though both concepts represent the integer 8, one should try to minimize the use of referents whenever possible, as they introduce a level of indirection and impact on interpretability. The reader is referred to [4] for a complete coverage of this topic.

R3. Whenever possible, the representation of an individual concept should use the object directly instead of using its referent.

Finally, in terms of syntax, any system should worry about its interconnection to the outside world and to other systems. There was a strong effort these past years toward the standardization of the CG notation that should be taken into account. Even though it may be more useful to the users or the knowledge engineer to be provided with some other syntactical format (which can be done at the external layer), the conceptual layer should stay closely tied to the CG standard.

R4. The conceptual layer should be represented in standardized CG notation as much as possible.

2.3 The Type Hierarchy

As for the description of the type hierarchy, though it is tempting to hide the complexity of representation of some concept in very specialized types, the modeling of the domain should be as close as possible to the natural categories (types) of the objects found in the domain. This is to facilitate the conceptualization of the solution space pertaining to that domain, and to facilitate the interpretability of the conceptual layer.

R5. The types used to describe the domain should be as close as possible to the natural categories found in the domain.

Furthermore, it has been noted in the past that the engineering process of an industrial system is often based on partial knowledge. It has also been noted that a lattice may be hard to conceptualize as the support structure for the type hierarchy. For these reasons, we may want to have the knowledge engineer describe the type hierarchy in any shape that seems natural to the task (at the external layer), and have a lattice completion algorithm to produce the type lattice required at the conceptual layer. In effect, every CGer knows that a lattice structure is compulsory to a CG system since the join operation must verify the most general common subtype of two types before considering joining concepts that conform to these two types.

R6. The type hierarchy of a CG system must be a lattice.

Also, all types, either for concepts or relations, should be seen as primitive. That is, the expansion and contraction operators should be used only in cases where the granularity of representation must be changed for some particular applications. As an example, let us cite natural language processing applications: text analysis or summarization. At the application level, case grammar relation types could be used to help analyze or produce natural language sentences. At the conceptual layer, only those types relevant to the description of the domain should remain. A

contraction/expansion operation would then be needed to change the granularity of representation to accommodate that change of representation.

> R7. At the conceptual layer, all types should be considered primitive, unless specialized applications require a change of representation.

2.4 Conformity of Individuals to Types

As mentioned in 2.3 above, in order to facilitate the acquisition of individuals, the external layer of a CG system may allow *multi-classification*, i.e., the assignment of the same individual to different types. Since recommendation R6 states that the type hierarchy should be a lattice (because of the join operation) and since R2 requires that an individual be stored under its most specific form, a lattice completion algorithm could complete the type hierarchy so that a common subtype is introduced whenever an individual is assigned to two different types t_1 and t_2 with $t_1 \bullet t_2$. Again, the independence of the external layer with regard to the conceptual layer may allow a user to see an individual classified under different types, though at the conceptual layer, it is assigned to only one type.

> R8. At the conceptual layer, there should be no multi-classification of individuals.

2.5 Conceptual Relations

When a binary relation is defined, there is no need to define its inverse relation at the conceptual layer, as it is implicit. However, natural language applications based on CG may require it. Then it could either be defined at the external layer or could be defined at the conceptual layer through a metamodel that would link the conceptual relation with its inverse. We will not get into more details about metamodeling using CG at this point; a forth-coming paper should introduce a theory on the subject. For more details see [5]. For now, we could suggest, also for interpretability purposes, to choose the name of a conceptual relation in terms of its role with regard to its arguments but also so that reading it from its input to its output arguments would come close to some natural language interpretation of its associated semantics.

> R9. The name of a conceptual relation should be chosen so that its purpose becomes clear when reading a graph that contains it, from its input toward its output arguments.

> R10. Its inverse relation should be omitted from the conceptual layer.

Binary relations may be reflexive, symmetrical or transitive. If reflexive or symmetrical, a relation of the same type should be added to the same concepts, but in the inverse order of the parameters.

R11. Reflexivity and symmetry completion should be achieved.

As for the transitivity property, a transitivity completion rule should be produced for that relation, and treated as a production rule (see Section 3.3 below).

2.6 Assertions and Negations

For simplicity reasons, only true graphs should be kept in the system so that the truth-value of retrieved information is never questioned and the inference mechanisms are kept simple, provided that the inference operators are sound and truth-preserving.

R12. Asserted graphs are assumed to be true.

All graphs in a CG system form a generalization hierarchy [6]. Constraint graphs represent empty subspaces of this generalization hierarchy, i.e., subspaces made of all graphs onto which they could project themselves (which are specializations of them). Any graph classified in such a subspace would automatically be known to be false since it violates a constraint. Therefore, when a graph whose truth-value is known to be false is encountered, it should be transformed into a constraint graph so that all the other graphs within the system are validated against it and the consistency of the system upholds. Making constraint graphs out of false graphs is actually a way to ensure the consistency of the system while providing the basis for explicitly handling negated information.

R13. False graphs should be represented as constraint graphs.

Again, in order to keep only true graphs in the system, the application of the specialization operators of the CG theory: the restrict and join operators, should be truth-preserving.

R14. A model checking procedure should be embedded in the application of any specialization operator.

This model checking procedure will prove to be useful also for answering queries, as a negative answer to some query could be produced as soon as the query itself is proven to have no model that supports it (see [7]).

3 Unicity of Representation: Complex Issues

3.1 Embedded Graphs, Memory Structure and Coreferencing

An additional computation problem with the CG formalism arises when embedded graphs are needed to model a domain. That adds to the complexity of the projection algorithm. Should projections be computed for all graphs without consideration of their embedding? Should all graphs be asserted on the universal sheet of assertion or could they be divided into micro-worlds? [8] proposed the introduction of micro-worlds as a way to classify relevant knowledge together mainly for efficiency purposes. Can we determine how conceptual graphs can be divided into different sheets of assertion? One of the purposes of what we proposed in [9] was to ensure that each asserted graph would be flattened out. If so, however, the CG system would then handle a *set* of sheets of assertion instead of only one, where each asserted graph would not embed any other graph. In brief, in [7] we proposed to extract the most embedded graph in a CG, called extent graph, and associate it with the graph from which it was extracted, called its intent graph. We would then replace the extent graph in its intent graph by a generic marker. Each intent graph identifies a context, each extent graph is asserted only in that context. The extent graph is said to be flat since it does not contain any embedded graph. If the intent graph is not flat, then the procedure is applied iteratively until all graphs have been flattened out. Figure 5 shows a context where the intent and extent graphs were produced from the graph of Figure 4.

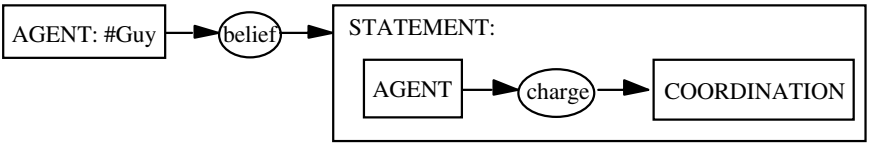


Fig. 4. An example of an embedded graph used to produce the context of Figure 5.

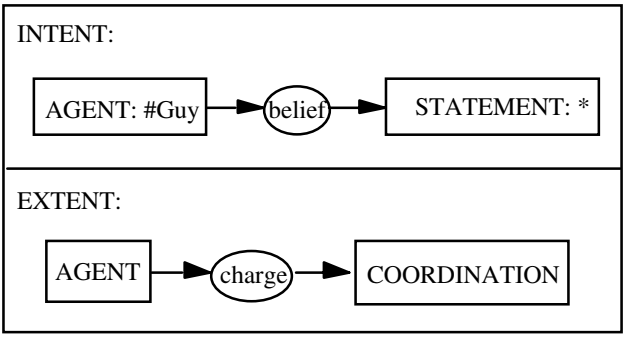


Fig. 5. A context with its intent and extent graphs in the top and bottom rectangles resp.

Finally, in order to avoid duplicating graphs and to speed up the computation of the projection operation, we proposed in [7] to compute a lattice structure based on formal concept analysis [10] from all contexts produced by the procedure described above. Independently of the procedure applied, we can state the following recommendation.

> R15. Whenever possible, a graph flattening procedure should be applied in a way that the semantics of the graphs would not be altered, but that the projection operation would be facilitated.

In any case, a single sheet of assertion for large size CG systems may bring the whole system into a state of brittleness. There is a need to provide fast access to the relevant information both at the implementation (see Section 4 below) and conceptual layer. At the conceptual layer, a memory organization must be designed if it impacts the processing capabilities of the system. For instance, in the design of a multi-agent system, each agent must have its private memory and may share information with other agents. For that application, a particular memory structure must be devised at the conceptual layer (see [11] for an example involving multi-agent systems). If background knowledge on the tasks for which the system is being developed is available, it may be possible to determine a memory structure that would benefit them.

> R16. The conceptual organization of the memory structure should be designed according to the tasks that the system must perform.

And as a consequence**,** like any CG system developer knows, and that is especially true for systems having more than one sheet of assertion, the coreferencing of objects should be done over all sheets of assertion, that is, in a totally global manner.

> R17. The coreferencing mechanism of a CG system should be global.

3.2 Rule Completion[1]

Another source of worry when dealing with CG systems where the projection operation is so important, is the rule completion dilemma. With production rules as described in [12] and as an example is given in Figure 6 below, should we apply the rules to the relevant graphs prior to the computation of the projection operation? How do we handle transitive rules like transitivity completion rules for instance? Rules may increase the size of a graph and of the overall CG base tremendously, even up to infinity.

In order to remedy this problem, each asserted graph should be represented in its most restricted format, that is, it should not include any information that could have

[1] Here we assume that the if-then rules described in this section only use the join operator and can not delete any information. Therefore, they are logical entailment rules.

been derived from the application of production rules. That way, only its *kernel* information would need to be explicitly represented, that is, the information that can not be derived from the application of production rules. Consequently, a CG base would only contain kernel graphs, acting as axiomatic information, and production rules from which information may be inferred.

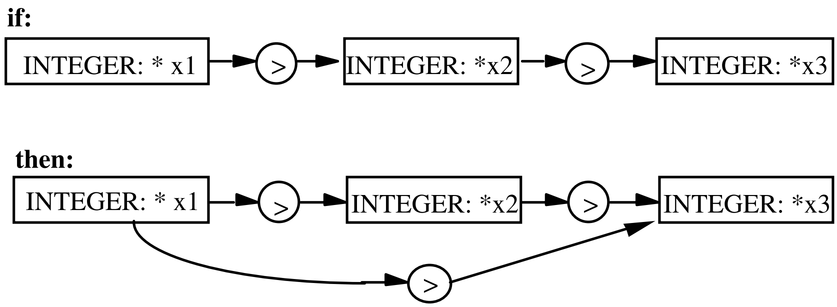


Fig. 6. An example of a production rule under the CG formalism.

Of course, before asserting any graph, a kernel extraction process must be run on the asserted graph, then called *kernel graphs*. Kernel graphs can be found by the application of the production rules in backward chaining mode. Unfortunately, it may be the case that many kernel graphs may arise from this kernel extraction mechanism. This is the price to pay for the unicity of representation that we seek to achieve here. Also, subsequent queries would have to undergo the same kernel extraction processing before being processed, so that the projection operation easily finds all the projections of a query graph onto the relevant kernel graphs.

> R18. When production rules are part of the system, a CG system should keep all asserted graphs under their kernel representation.

3.3 Procedural Attachments: Actors and Processes

Actors [6] and processes [13] may be useful to CG systems as they allow the inclusion of update procedures that are triggered automatically when some assertions are made. In brief, an actor instantiates its generic output concepts based on some computation from its input arguments (when all instantiated). As example, let Figure 7 below define the > actor. This actor computes a boolean value based on the mathematical relation > between two integers.

This actor would be useful to enforce the semantics of the > relation when used in a CG. From recommendation R12 above all asserted graphs are assumed to be true, but there is no way to ensure that the semantics of the > relation in Figure 8 below is not violated. Since the set of integer numbers is infinite, symbolic computation would need the explicit representation of the > relation for an infinite set of pairs of integer numbers in order to compute the truth-value of this relation for particular arguments.

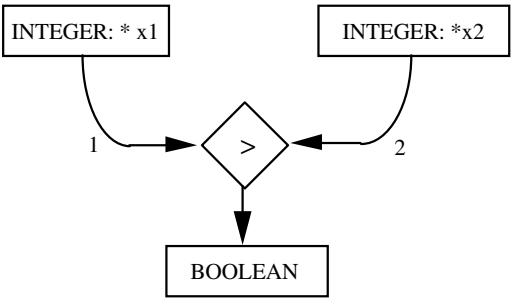


Fig. 7. The definition of the > actor, called *actor graph*.

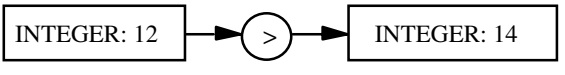


Fig. 8. A CG that uses the > relation.

However, whenever the > relation is used in a graph g, a rule could be triggered so that the actor graph is added to the graph of Figure 8. That would result in a call to the > actor and the subsequent instantiation of the Boolean concept (see Figure 9). The Boolean concept being instantiated to False would provide the required information to alert the system's administrator of this state of fact.

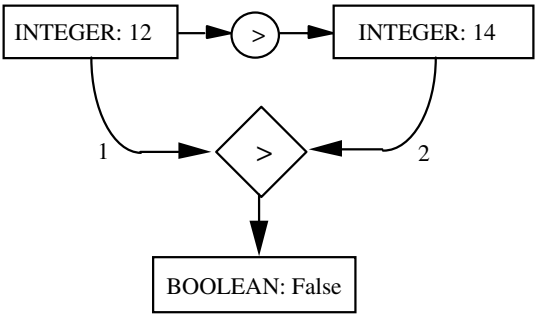


Fig. 9. Joining the graphs of Figure 7 and 8 together.

In this example, the semantics of the > relation was encoded in a procedural form in the > actor (because it would not have been captured in a finite number of assertions using the > relation alone). Actors are thus useful to represent domain constraints for infinite or very large domains. They are also useful to automatically query a database for the benefit of some decision making process that the CG system is achieving.

Processes have been defined in various ways in the literature [14]. For instance, [13] defines them as actors whose input and output arguments are assertions, that is, concepts of type STATEMENT with CG as their referents. For instance, through the use of a process that recognizes a bad use of the > relation (see Figure 10), some action can be undertaken to alert the system administrator and ensure that the system consistency is upheld.

Actors provide gateways to procedural computation allowing CG systems to benefit from both procedural and declarative programming paradigms. Processes implement state changes in a system. Contrarily to if-then rules, they may trigger any type of state change, rendering a non-monotonic evolution of the system. Therefore, they may be used with great care. Non-monotonic systems provide a level of complexity that is beyond that of first-order logic. For instance, kernel graphs would need to be recomputed for each state of the system.

4 The Implementation Layer: A Few Tips

4.1 Indexes

In the same line of thought, when designing the implementation layer, indexes on the asserted graphs will need to be built so that the computation of the projection operation is as fast as can be. These indexes could be supported by a relational database system where an indexing mechanism is provided. But, as a first filter, an indexing mechanism built from the asserted graphs themselves [15] could be beneficial to the efficiency of a CG system. Also, feature selection algorithms could preprocess the asserted graphs so that the most useful *subgraphs* in terms of their discrimination power are identified and used as indexes. However, the type of index that is the most suited for a given application remains to be investigated.

> R19. The investigation of proper indexing mechanisms should be conducted so that an appropriate implementation platform is chosen in light of its capability to retrieve graphs efficiently whenever needed.

4.2 The Technological Platform

I must add also that the industry is usually not too keen on in-house solutions. One key element of choosing a technological platform (either software or hardware) as a

solution to a particular problem relies heavily on the number of year-person put in its development and testing, on the stability of the platform, on its expectation of longevity (will it be supported), on its portability, on market trends, and on its development, implementation and maintenance cost.

> R20. The implementation layer should rely on a well-supported industrial-type platform (like a relational database management system for instance).

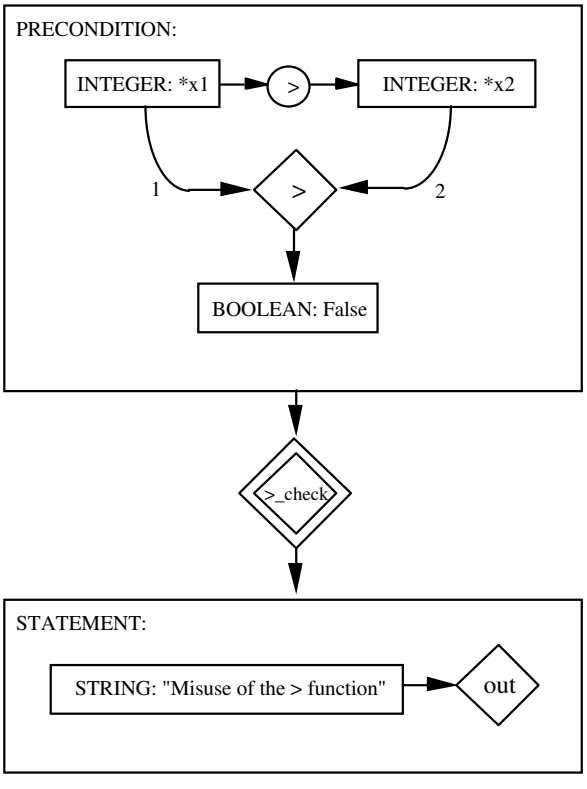


Fig. 10. A simple process (rule) that detects bad uses of the > relation.

Though maybe not sufficient, the starting point of the implementation layer should be some heavily tested and broadly used industrial product. Additional add-ons may be needed to fit the specificity of CG based systems. The implementation layer should permit to add extensions to the basic platform.

> R21. The technological platform chosen as the basis for the implementation layer should be based on an open architecture, allowing gateways to other systems and programming environments.

In the CG literature, the implementation layer has been neglected so far. Considering the risk associated with the selection of a technology over the next nowadays, there will be no large scale use of CG systems until a well-tested platform is available. Consequently, CG researchers should now tackle the problems associated with the implementation layer of a CG system. Though past efforts have not always been successful for various reasons (lack of proper specifications, coordination and funds), I believe that individual efforts may provide enough background knowledge on the associated pitfalls of this implementation layer so that financial opportunities may eventually arise for targeted applications.

> R22. The implementation layer of a CG system should be devoted attention from the CG community if a large scale CG tool platform is to be developed, which would help disseminate the theory.

5 Conclusion

This paper tried to demonstrate that a CG system can be looked at under a three-layer architecture. To my opinion, this architecture is of extreme importance to the understanding and planning of new technological improvements in the CG theory. As a research community, we should relate our work to this architecture, providing the pieces of a big puzzle, pieces that will eventually fit. More emphasis should be put on the impact of our work onto these layers and thus on the mappings between layers.

All three layers are important to the dissemination of the CG theory. The external layer introduces potential users to CG systems. It should be designed so that a "super" user (the knowledge modeler) is provided with simple editing facilities through a graphical interface. As of now, only Charger [16] seems to provide a GUI and be widely available (written in Java).

Of course editing CG is far from enough. Knowledge processing under the CG formalism should also be available. Towards that goal, theoretical results are now abundant [17-23]; CG systems are well understood by the community. Some prototypes and specialized tools exist [16]. However, any industrial tool should be based on a widely supported platform where the implementation layer was carefully designed to meet industrial constraints.

As a whole, a CG-based tool for industrial applications should deal with the following issues in order to obtain the necessary credibility in the market place: efficiency, scalability, portability, flexibility, simplicity of development, use and maintenance, and interpretability. Based on my personal experience, the recommendations put forth in this paper aim at facilitating the development of CG systems with regard to these issues. They are by no means exhaustive nor absolute; specific needs may require other ways of dealing with these issues. However, they could be considered as a first set of guidelines when having to develop large scale CG based systems which, in the end, is our common goal.

Acknowledgement

I wish to thank Prof. Rudolf Wille for inviting me to give this talk at the 8th International Conference on Conceptual Structures (ICCS-2000) in Darmstadt, Germany. Having been active in the CG community for over 12 years now, it is a great opportunity for me to sit back and reflect upon the lessons that research and development in this field have taught me over the years, particularly in the context of industrial development and use of CG based systems.

References

1. Mineau, G.W.: Views, Mappings and Functions: Essential Definitions to the Conceptual Graph Theory. In: Tepfenhart, W.M., Dick, J P., Sowa, J.F. (eds): Lecture Notes in Artificial Intelligence, vol. 835. Springer-Verlag, (1994). 160-174
2. Mineau, G.W. & Missaoui, R.: The Representation of Semantic Constraints in CG Systems. In: Lukose, D., Delugach, H., Keeler, M., Searle, L. & Sowa, J.F. (eds): Lecture Notes in Artificial Intelligence, vol. 1257. Springer-Verlag, (1997) 138-152
3. Chein, M. & Mugnier, M.L.: Specialization: Where Do the Difficulties Occur? In: Pfeiffer, H.D., Nagle, T.E. (eds): Lecture Notes in Artificial Intelligence, vol. 754. Springer-Verlag, (1993) 229-238
4. Gerbé, O., Mineau, G.W., Keller, R.K.: Conceptual Graphs, Referents and Metamodeling. Submitted for publication
5. Gerbé, O.: A Uniform Representation for Modeling and Meta-Modeling Activities in the Development of a Corporate Memory. Université de Montréal: Ph.D. thesis in Computer Science, (2000)
6. Sowa, J. F.: Conceptual Structures: Information Processing in Mind and Machine. Addison-Wesley, (1984)
7. Mineau, G.W.: The Extensional Semantics of the Conceptual Graph Formalism. Submitted for publication.
8. Lenat, D. B. & Guha, R. V.: Building Large Knowledge-Based Systems, Representation and Inference in the Cyc Project. Addison-Wesley, (1990)
9. Mineau, G.W. & Gerbé, O.: Contexts: A Formal Definition of Worlds of Assertions. In: Lukose, D., Delugach, H., Keeler, M., Searle, L. & Sowa, J.F. (eds): Lecture Notes in Artificial Intelligence, vol. 1257. Springer-Verlag, (1997) 80-94
10. Wille, R., Restructuring Lattice Theory: an Approach Based on Hierarchies of Concepts. In: I. Rival (ed.): Ordered Sets. (1982) 445-470
11. Mineau, G.W.: Constraints on Processes: Essential Elements for the Validation and Execution of Processes. In: Tepfenhart, W., & Cyre, W. (eds): Lecture Notes in Artificial Intelligence, vol. 1640. Springer-Verlag, (1999) 66-82

12. Baget, J.F., Genest, D. Mugnier, M.L.: A Pure Graph-Based Solution to the SCG-1 Initiative. In: Tepfenhart, W., & Cyre, W. (eds): Lecture Notes in Artificial Intelligence, vol. 1640. Springer-Verlag, (1999) 335-376
13. Mineau, G.W.: From Actors to Processes: The Representation of Dynamic Knowledge Using Conceptual Graphs. In: Mugnier, M.-L., Chein, M. (eds): Lecture Notes in Artificial Intelligence, vol. 1453. Springer-Verlag, (1998) 65-79
14. Lukose, D. & Mineau, G.W.: A Comparative Study of Dynamic Conceptual Graphs. In: Gaines, B. & Musen, M. (eds): Proceedings of the 11th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop (KAW-98), University of Calgary, Calgary, Alberta, Canada. Section VKM-7, (1998) 20 pages
15. Levinson, R. A. & Ellis, G.: Multi-level hierarchical retrieval. *Knowledge Based Systems*, **5**(3), 233-244, (1992)
16. http://www.cs.uah.edu/~delugach/CG/home-body.html#Tools
17. Lecture Notes in Artificial Intelligence, vol. 699. Springer-Verlag. (1993)
18. Lecture Notes in Artificial Intelligence, vol. 835. Springer-Verlag. (1994)
19. Lecture Notes in Artificial Intelligence, vol. 954. Springer-Verlag. (1995)
20. Lecture Notes in Artificial Intelligence, vol. 1115. Springer-Verlag. (1996)
21. Lecture Notes in Artificial Intelligence, vol. 1257. Springer-Verlag. (1997)
22. Lecture Notes in Artificial Intelligence, vol. 1453. Springer-Verlag. (1998)
23. Lecture Notes in Artificial Intelligence, vol. 1640. Springer-Verlag. (1999)

Conceptual Graphs, Metamodeling, and Notation of Concepts

Olivier Gerbé[1], Guy W. Mineau[2], and Rudolf K. Keller[3]

[1] HEC Montreal.
3000, chemin de la Côte-Sainte-Catherine, Montréal, Québec, Canada H3T 2A7
Olivier.Gerbe@hec.ca
[2] Université Laval
Québec, Québec, Canada G1K 7P4
mineau@ift.ulaval.ca
[3] Université de Montréal
C.P. 6128 Succursale Centre-Ville, Montréal, Québec, Canada H3C 3J7
keller@IRO.UMontreal.ca

Abstract. Knowledge management, in particular corporate knowledge management, is a challenge companies and researchers have to meet. The conceptual graph formalism is a good candidate for the representation of corporate knowledge, and for the development of knowledge management systems. But many of the issues concerning the use of conceptual graphs as a metalanguage have not been worked out in detail. By introducing a function that maps higher level to lower level, this paper clarifies the metalevel semantics, notation and manipulation of concepts in the conceptual graph formalism. In addition, this function allows metamodeling activities to take place using the CG notation.

1 Introduction

Knowledge management, especially corporate knowledge management, is a challenge companies and researchers have to meet. In a previous work, we compared conceptual graphs with other formalisms [3, 5] and we concluded that the conceptual graph formalism was a good candidate for the representation of corporate knowledge, and for the development of knowledge management systems.

Conceptual graphs are a knowledge representation formalism introduced by John Sowa [7] where objects of the universe of discourse are modeled by concepts and conceptual relations that associate concepts. Conceptual graphs have been extensively used and studied by a large scientific community. Using conceptual graph formalism, a corporate memory has been developed at the Research & Development Department of DMR Consulting Group Inc in order to memorize the methods, know-how and expertise of its consultants [4]. This corporate memory, called Method Repository, is a complete authoring environment used to edit, store and display the methods used by the consultants of DMR. The core of the environment a knowledge engineering system based on conceptual graphs. Four methods are commercially delivered and their documentation in paper and

B. Ganter and G.W. Mineau (Eds.): ICCS 2000, LNAI 1867, pp. 157–171, 2000.
© Springer-Verlag Berlin Heidelberg 2000

in hypertext format is generated from conceptual graphs. About two hundred business processes have been modeled and from about 80,000 conceptual graphs, we generated more than 100,000 HTML pages in both English and French that can be browsed using commercial Web browsers.

However some fundamental aspects of conceptual graphs remains ambiguous and not formally specified. For example, the notation of concepts is syntactically defined in the new CG standard [1] but its use and its semantics seems ambiguous. Figure 1 illustrates this problem of notation. How could one represent the notion (concept) of the person John using conceptual graphs?



Fig. 1. Which notation?

Another example is the manipulation of embedded graphs. Concepts may have conceptual graphs in the referent field. Figure 2 illustrates this problem of manipulation. How could a CG system access the conceptual graph that is in the referent field of this graph?

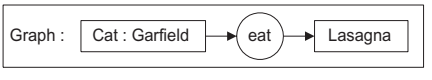


Fig. 2. How to access the embedded graph?

We propose in this paper a metamodeling approach which will semantically define the notation and manipulation of concepts in the CG language. Little work has been done on metamodeling and conceptual graphs. John Esch in [2] introduces two relationships: Kind that links a concept to its type and Subt that links two concept types that are in a subtype relationship. He defines, using the relationship Subt, a type hierarchy for each higher order level and he links, using the relationship Kind, types from different levels. But Esch does not deal with conceptual relations and relation types. In [8] Michel Wermelinguer defines more formally higher order types and proposes a translation to first order logic. He defines one hierarchy for all the concept types and one hierarchy for all the relation types and organizes them in regard of their nature and their order. But none proposes a complete metamodel of the conceptual graph language itself.

These two approaches are compatible and our work extends the notions introduced by these authors. We formally define the conceptual graph language using conceptual graphs and we propose in this paper a notation for referents based on the metamodel. In addition, we unify the two operators τ and ρ [6] in

one general operator ω that maps a higher order level to a lower order level and so allows the manipulation of concepts at the lower levels.

This paper[1] is organized as follows. Section 2 presents an overview of the metamodel of the conceptual graph formalism and Section 3 details five basic components of this metamodel: element, concept, referent, individual concept and generic concept. Based on this metamodel, Section 4 formalizes the notation of referents in concepts and Section 5 introduces the function ω that links a concept to the entity it represents and illustrates the use of this function in metamodeling, laying the foundation for a CG theory of metamodeling. Section 6 concludes and presents future work.

2 CG Metamodel Overview

This section gives an overview of the conceptual graph metamodel. The metamodel defines the basic components needed to represent knowledge. Figure 3 presents the concept type hierarchy of the CG language metamodel.

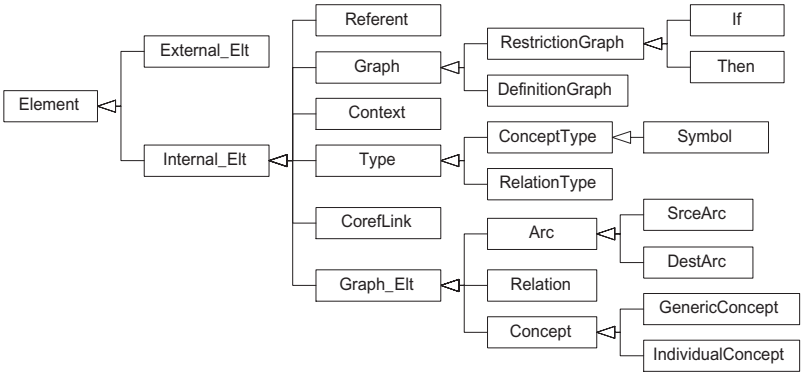


Fig. 3. Metamodel Concept Type Hierarchy.

At the highest level, we have external elements (ExternalElt) that are part of the real world to be represented and internal elements (InternalElt) that are building blocks of the CG language.

External elements represent entities of the Universe of Discourse that are outside of the system but can be referenced by internal elements.

Internal elements are categorized under six types: Referent, Graph, Context, Type, CorefLink, and GraphElement. Referents are the proxies that stand for entities of the Universe of Discourse; Graphs are the sentences of the CG language; Contexts help to cluster knowledge; Types are categories to classify entities; Coreference Links associate elements that represent the same entities; and Graph Elements are concepts, relations and arcs.

[1] This work is part of a research project supported by HEC Montreal.

Among concepts we distinguish individual concepts (IndividualConcept) that represent identified entities and generic concepts (GenericConcept) that represent unidentified entities.

Graph has two specialized subtypes: DefinitionGraph and RestrictionGraph. Definition graphs are used to define concept types and relation types. Restriction graphs are graphs that must always be false. They are used to state constraints on types. If and Then are special cases of restriction graphs where the conditions they represent must already be present or be acquired simultaneously.

3 CG Metamodel Components

This section details the core components of the conceptual graph formalism that are relevant to the notation and manipulation of concepts. We introduce a formal definition using conceptual graphs of the five basic concept types related to concept: Element, Concept, Referent, Individual Concept and Generic Concept. But before, in order to understand specification graphs in formal definitions, we illustrate on an example how concept types are specified.

3.1 Concept Type Specification

Concept Types are defined by three kinds of graphs : definition graph, restriction graph, and rule graph as illustrated in Figure 4. This figure shows the graph that specifies the concept type Driver. Definition, restriction and rule graphs give the necessary and sufficient conditions to recognize instances of Driver.

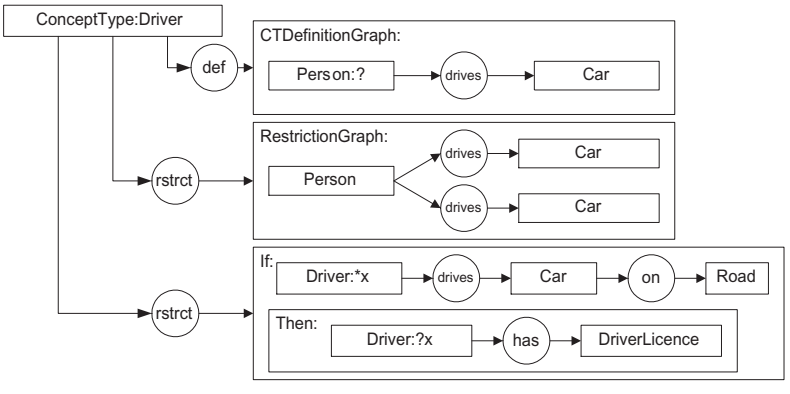


Fig. 4. Type definition of Driver.

Definition Graph. The definition graph shows the conceptual relations that a concept must have. In the example a person that drives a car is a driver under the conditions stated by restriction graphs and rule graphs.

Formally, in [7], a concept type is defined by a lambda expression, the example in Figure 4 states the following equation :

`Driver = [Person : λ]→(drives)→[Car].`

The symbol λ shows that the concept Person is the formal parameter. In the graphic form and in the linear form the symbol λ is replaced by a question mark.

Restriction Graph. Restriction graphs specify supplementary and necessary conditions; these graphs are forbidden graphs or subgraphs. They show particular topologies that must not exist. In our example, the restriction graph states that a driver cannot drive two cars[2].

Rule Graph. Rule graphs specify other supplementary and necessary conditions that are expressed more easily with rules[3]. In our example, the rule graph states that if a driver drives a car on a road then he has a driver license.

3.2 Element

The conceptual graph language is made of elements that are combined to represent knowledge. Figure 5 shows the specification graph of type Element. An element is symbolized by one or more symbols but one symbol may not be linked to two different elements. Symbols are unique identifiers in the system.

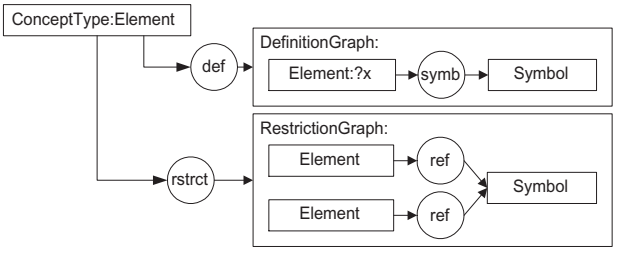


Fig. 5. Specification graph of type Element.

3.3 Concept

The notion of concept is the fundamental notion of the conceptual graph theory. A concept is the representation of an object, an idea or any notion one can perceive and express.

[2] Two different boxes represent two different concepts.
[3] Any rule graph can be rewritten as a restriction graph.

Definition 1. *A concept is the representation of an object of the Universe of Discourse. It is the assembly of two parts: a referent that identifies the represented object and the type that classifies it.*

Figure 6 shows the specification graph of type Concept. The definition graph states that a concept is an internal element that has a type, a referent, and is element of a graph. Restriction graphs clarify building rules, a concept has one and only one type and one and only one referent.

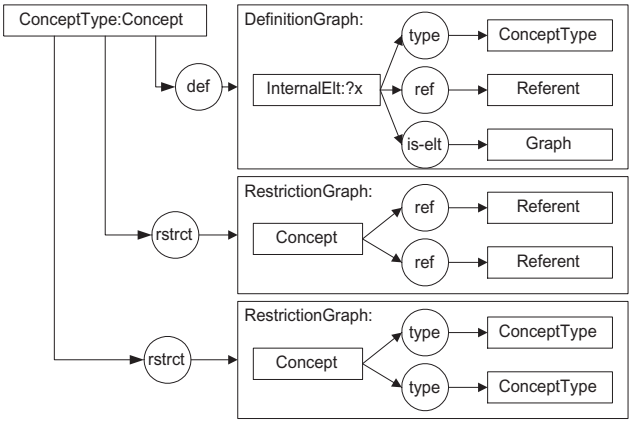


Fig. 6. Specification graphs of type Concept.

One can notice that one type may be associated to several different referents and one referent may be associated to different types and make up different concepts. This later mechanism provides the representation of point of views, one object may be perceived in different ways.

Examples of concepts are :

```
[Person], [TaxiDriver], [Concept]
[Person : #Tom], [Concept : #624], [ConceptType : Person]
```

Examples of concepts whose concept type is Concept are:

```
[Concept : #624], [Concept : [ConceptType : Person] ],
[Concept : [Referent : #624] ]
```

3.4 Referent

The referent is the part of a concept that represents and identifies the object of the universe of discourse for which the concept is an interpretation.

Definition 2. *A referent is a proxy for an object of the universe of discourse in the knowledge base. A referent is made up of a quantifier and a designator that refers to the object.*

Figure 7 shows the specification graph of type Referent. A referent is a part of a concept and represents an element (external or internal). Restriction graphs state that one referent represents one and only one element and two referents cannot represent the same element. There exists a special referent written `#blank` that 'represents' an element which exists but is not identified (3.6).

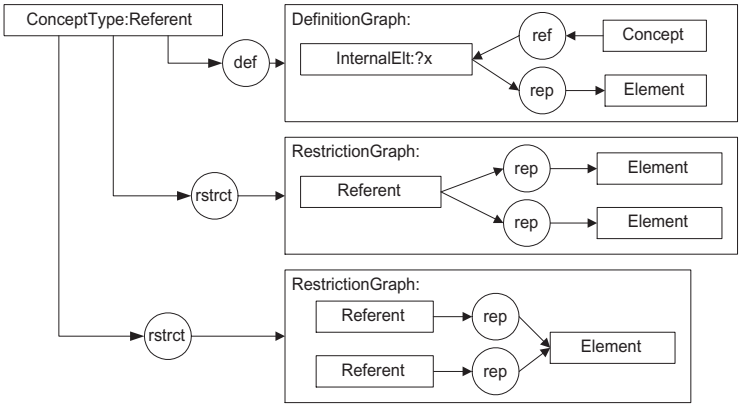


Fig. 7. Specification graph of type Referent.

Examples of referents are :

```
#12, #blank, Tom, Person, ∀*x.
```

3.5 Individual Concept

Definition 3. *An individual concept is a concept whose represented entity is known. The concept is an interpretation of an object on the universe of discourse that is identified and represented in the knowledge base by a referent other than* `#blank`.

Figure 8 presents the specification graph of type IndividualConcept. An individual concept has a referent that represents an element. The referent of an individual concept is different from the blank referent.

Examples of individual concepts are :

```
[Person : Tom], [Referent : #624], [ConceptType : Person]
```

3.6 Generic Concept

There exist two kinds of concepts depending whether the entity represented by the concept is known or unknown.

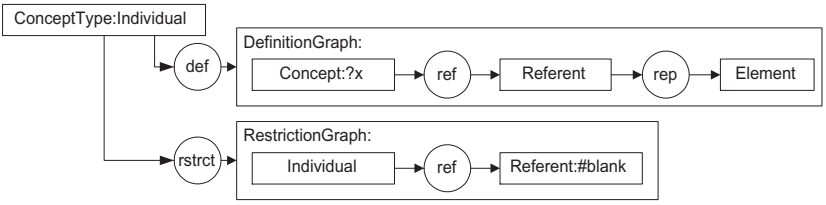


Fig. 8. Specification graphs of type IndividualConcept.

Definition 4. *A generic concept is a concept that represents an unknown entity. The concept is an interpretation of an object of the universe of discourse that exists but is not identified.*

Figure 9 presents the specification graph of type GenericConcept. A generic concept has the special referent `#blank` that represents an unidentified element. In practice this referent is omitted.

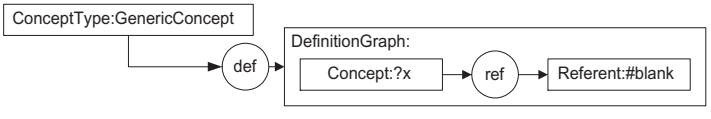


Fig. 9. Specification graphs of type GenericConcept.

Examples of generic concepts are :

```
[Person], [TaxiDriver:#blank], [Concept]
```

4 Notation

This section formalizes the notation of concepts in the conceptual graph formalism. Individual concepts are the basic components of the CG language. Individual concepts are concepts that are abstractions of well identified entities. The referent slot of an individual concept is symbolized by a unique literal preceded by the symbol `#`. The referent represents the identified entity in the system. The entity of the universe of discourse itself may be symbolized. Figure 10 presents the underlying metamodel. A concept has a referent. This referent is symbolized by a symbol and represents an element that may also be symbolized.

An alternative for the notation of the concept is to replace its symbol by any symbol of the represented element. This mechanism may be formalized by a metalevel rule as illustrated in Figure 11. If an element represented by the referent of a concept is itself symbolized by symbols then the referent may be symbolized by the same symbols.

We illustrate this rule on three different examples: notation of concept, notation of concept type and notation of graph.

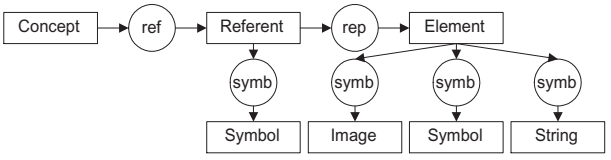


Fig. 10. The metamodel of symbolization.

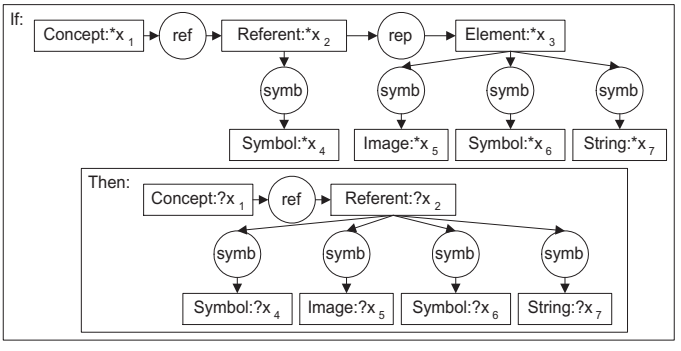


Fig. 11. The notation rule.

4.1 Notation of Concept

A lot of different notation has been used in the literature to denote individual concepts. We give here the example of the representation of the person John. According to the metamodel of Figure 10, the metalevel conceptual graph describing the concept that stands for the person John is presented in Figure 12. The concept `[Person:#34]` has a referent that is symbolized by `#34`. This referent represents the element John (an external element), which can be symbolized by different symbols: a string, a symbol[4] and an image.

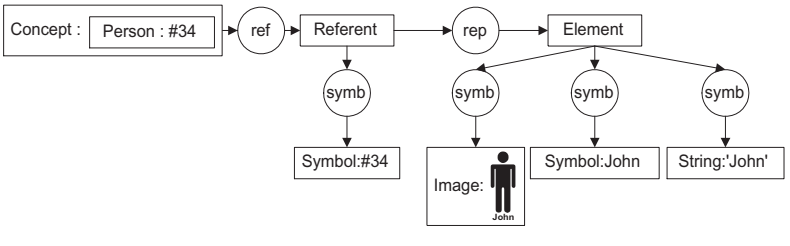


Fig. 12. The meta conceptual graph for notation of John.

[4] A symbol is different from a string. It is a whole and cannot be edited

By applying the notation rule defined above the referent `#34` may be replaced by any of the symbols that stands for the person John. Figure 13 shows the possible notation of the same concept.



Fig. 13. Different and equivalent notations of John.

4.2 Notation of Concept Type

In the literature concept types, when used as concepts, are denoted with the type label in the referent field. Figure 14 presents the meta level conceptual graph describing the concept that stands for the concept type Person. The concept `[ConceptType:#15]` has a referent symbolized by `#15` The referent represents the type Person that is symbolized by the symbol `Person`.

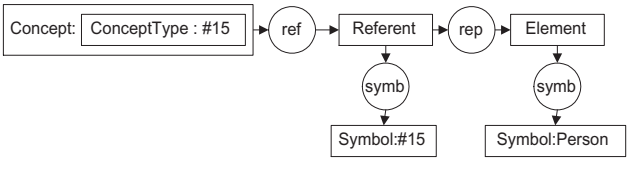


Fig. 14. The meta conceptual graph for notation of the type Person.

By applying the notation rule the referent `#15` may be replaced by the symbol that stands for the type. Figure 15 shows the two possible notations of the concept type.

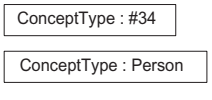


Fig. 15. Different equivalent notations of the type Person.

4.3 Notation of Graph

Concepts that represent conceptual graphs are denoted with the type Graph in the type field and the graph itself in the referent field. Figure 16 presents the meta

level conceptual graph describing the concept that stands for the graph `[Cat: Garfield]→(eat)→[Lasagna]`. The concept `[Graph:#72]` has a referent symbolized by `#72`. The referent represents the graph that is symbolized by a symbol `GarfieldMeal`, a linear form and a first order logic form of the graph.

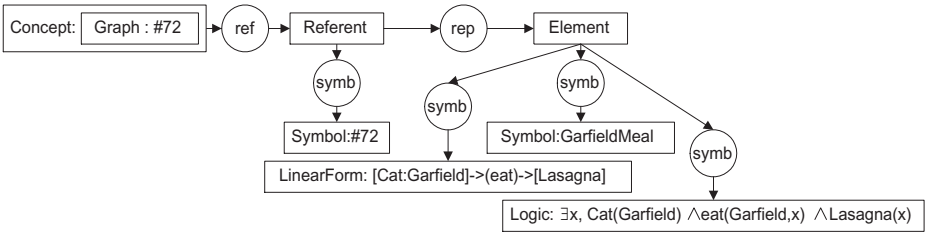


Fig. 16. The meta conceptual graph for notation of the Garfield meal.

By applying the notation rule the referent `#72` may be replaced by any of the symbols that stands for the graph. Figure **??** shows possible notations of the same graph.

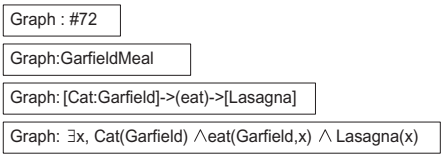


Fig. 17. Different equivalent notations of the graph Garfield Meal.

Using this notation mechanism, we can define named elements. We will write `[Graph : GarfieldMeal [Cat: Garfield]→(eat)→[Lasagna]]` to state that `GarfieldMeal` and `[Cat: Garfield]→(eat)→[Lasagna]` are two symbols of the graph. Any reference to the graph may be subsequently done by the concept `[Graph : GarfieldMeal]`. For example, this mechanism may be apply with situations; it allows to name situation and avoid to repeat every time the graph that describes the situation when we make a reference to this situation.

5 Meta Level to Data Level and Data Level to Entities

This section introduces the function that links a concept to the entity or entities it represents.

In [6] Sowa describes a mapping between the meta level and the data level. To translate a meta level statement into a data level statement, Sowa introduces two functions τ and ρ. The function τ translates a referent name into a type

label. The example below illustrates the use of the function τ. The meta level statement: t_1 is a subtype of t_2 is transformed into a data level statement: every t_1 is a t_2 .

```
IF : [Type:*t1]->(subt)->[Type:*t2]
THEN : [τt1 : ∀*x] [τt2 :?x]
```

The function ρ has the same behavior as τ on relation types and relations; it translates the name of a relation into a relation type label. The example above presents the translation rule from the meta level to the data level in Entity-Relationship diagram.

```
IF : [Type:*t1]->(arg1)->[Relation:*r]<-(arg2)<-[Type:*t2]
THEN : [τt1]->(ρr)->[τt2]
```

5.1 Function ω

More generally, we need a function to access the entity represented by a concept in order to use it. For example, we would like to access the image represented by the concept `[Drawing : BeautifulLandscape]` or we would like to be able to manipulate the graph represented by the concept `[Graph:[Cat]->(on)->[Mat]]`.

Let us define ω as such a function.

Definition 5. *The function ω is defined over $\mathcal{C} \rightarrow \mathcal{E}$ where $\mathcal{C}$ is the set of concepts that represent entities of the system and $\mathcal{E}$ is the set of all referenced elements (internal and external elements).*

Applied on a concept the function ω returns the entity represented by the concept. Obviously, the function is defined on the set of concepts that represent entities of the system.

```
ω([Drawing : BeautifulLandscape]) =
ω( [Graph : [Cat]->(on)->[Mat] ]) = [Cat]->(on)->[Mat]
```



5.2 ω versus τ and ρ

Using the function ω, the above example using τ may be rewritten as follows:

```
IF : [Type:*t1]->(subt)->[Type:*t2]
THEN : [ω([Type:*t1]):∀*x] [ω([Type:*t2]):?x]
```

where `ω([Type:*t1])` returns the entity represented by the concept `[Type:*t1]` that is the type represented by `t1` and `ω([Type:*t2])` returns the entity represented by the concept `[Type:*t2]` that is the type represented by `t2`.

To show the equivalence with τ and ρ if we replace `ω([Type:*t1])` by `ωt1`. The rule becomes:

```
IF : [Type:*t1]->(subt)->[Type:*t2]
```

`THEN : [`ω`t`$_1$`:`$\forall$`*x] [`ω`t`$_2$`:?x]`

In a same way, the translation rule for an E-R diagram becomes:

`IF : [Type:*t`$_1$`]->(arg1)->[Relation:*r]<-(arg2)<-[Type:*t`$_2$`]`
`THEN : [`ω`t`$_1$`]->(`ω`r)->[`ω`t`$_2$`]`

5.3 Function ω and Metamodeling

The function ω maps a higher level to a lower level. To show the power and the use of the function ω, we give below four examples using the function. Using ω we give the definition of the transitivity, symmetry and anti-symmetry properties of a relation. Such definition uses generic relation types at the higher and lower level in specification graphs. So before giving the associated definitions, we present the notation of co-referenced types that are mapped from a higher level to a lower level.

Concepts that represent co-referenced concepts are denoted with the same symbol in the referent field preceded by an asterisk and by a question mark. Figure 18 presents the meta level conceptual graph describing two co-referenced concepts that stand for an unidentified Concept Type[5].

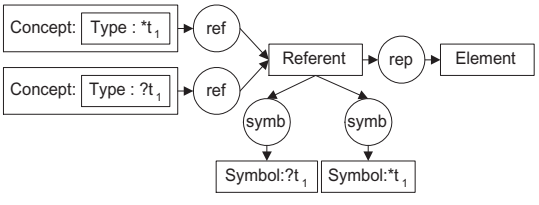


Fig. 18. The meta conceptual graph for notation of coreferenced concepts.

Two concepts that are linked by a coreference link are abstractions of the same element. Therefore they have the same referent that represents the unidentified element. Applied on these concepts the function ω returns the unidentified element that is represented by the referent. For simplicity reasons, ω`([Type:?t`$_1$`])` will be denoted `?t`$_1$ and ω`([Type:*t`$_1$`])` will be denoted `*t`$_1$.

Transitivity. A relationship $\mathcal{R}$ is transitive if and only if :

$$x\mathcal{R}y \wedge y\mathcal{R}z \Rightarrow x\mathcal{R}z \tag{1}$$

Figure 19 presents the graph that defines a transitive relationship at the meta level.

[5] In a coreference link the defining concept is denoted with an asterisk and the bound concept is denoted with a question mark.

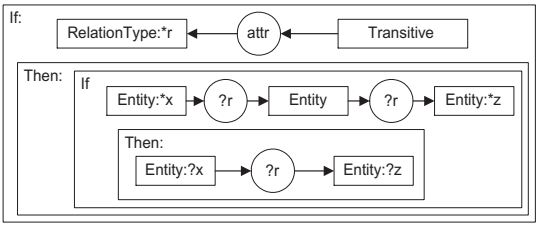


Fig. 19. Definition rule of a transitive relationship.

Symmetry. A relationship $\mathcal{R}$ is symmetrical if and only if :

$$x\mathcal{R}y \Rightarrow y\mathcal{R}x \tag{2}$$

Figure 20 presents the graph that defines a symmetrical relationship.

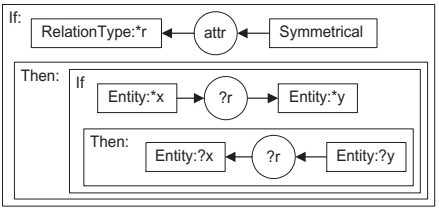


Fig. 20. Definition rule of a symmetrical relationship.

Anti-symmetry. A relationship $\mathcal{R}$ is anti-symmetrical if and only if :

$$x\mathcal{R}y \wedge y\mathcal{R}x \Rightarrow x = y \tag{3}$$

which is equivalent to

$$\neg(x\mathcal{R}y \wedge y\mathcal{R}x \wedge x \neq y) \tag{4}$$

Figure 21 presents the graph that defines an anti-symmetrical relationship[6].

6 Conclusion

In this paper, we introduced an approach to clarify the semantics, notation, and manipulation of concepts in CG language. This approach uses metamodeling constructs based on CG language. This demonstrates that conceptual graphs may be used in metamodeling activities. The function ω that maps a higher level to a lower level allows the manipulation of concepts from different levels.

[6] Note: two different boxes represent two different concepts

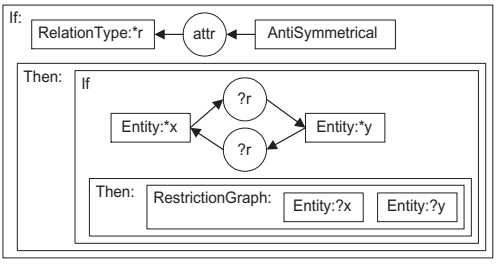


Fig. 21. Definition rule of an antisymmetrical relationship.

With a simple problem, the representation of concepts, we showed that through the formal definition of a meta level and of mapping functions from one level to the next, we can represent in a uniform way higher level and lower level and navigate between them. There is an obvious need for a complete theory of metamodeling in the CG formalism. We are currently developing such a theory.

References

[1] NCITS.T2 Committee. *Conceptual Graph Standard - draft proposed American National Standard*, 1999.

[2] J. Esch. Contexts, Canons and Coreferent Types. In W. Tepfenhart, J. Dick, and J. Sowa, editors, *Proceedings of the Second International Conference on Conceptual Structures, ICCS'94*, pages 185–195. Springer-Verlag, 1994.

[3] O. Gerbé. Conceptual Graphs for Corporate Knowledge Repositories. In *Proceedings of 5th International Conference on Conceptual Structures*, pages 474–488, 1997.

[4] O. Gerbé and al. *Macroscope Architecture: Architecture of DMR Repository.* DMR Consulting Group Inc., 1994.

[5] O. Gerbé, R. Keller, and G. Mineau. Conceptual Graphs for Representing Business Processes in Corporate Memories. In *Proceedings of the sixth international Conference on Conceptual Structures (ICCS'98).* Springer-Verlag, 1998.

[6] J. Sowa. Relating Diagrams to Logic. In John F. Sowa (Eds.) Guy W. Mineau, Bernard Moulin, editor, *Conceptual Graphs for Knowledge Representation, ICCS '93*, volume 699, pages 1–35. Springer-Verlag, 1993.

[7] J. F. Sowa. *Conceptual Structures: Information Processing in Mind and Machine.* Addison-Wesley, 1984.

[8] M. Wermelinger. Conceptual Graphs and First-Order Logic. In G. Ellis, R. Levinson, W. Rich, and J. Sowa, editors, *Proceedings of the Third International Conference on Conceptual Structures, ICCS'95*, pages 323–337. Springler-Verlag, 1995.

Knowledge Representation and Reasonings Based on Graph Homomorphism

Marie-Laure Mugnier

LIRMM
161, rue Ada, 34392 Montpellier - France
mugnier@lirmm.fr

Abstract. The main conceptual contribution in this paper is to present an approach to knowledge representation and reasonings based on labeled graphs and labeled graph homomorphism. Strengths and weaknesses of this graph-based approach are discussed. Main technical contributions are the followings. Fundamental results about the kernel of this approach, the so-called *simple graphs model* are synthesized. It is then shown that the basic deduction problem on simple graphs is essentially the same problem as conjunctive query containment in databases and constraint satisfaction; polynomial parcimonious transformations between these problems are exhibited. Grounded on the simple graphs model, a knowledge representation and reasoning model allowing to deal with facts, production rules, transformation rules, and constraints is presented, as an illustration of the graph-based approach.

Introduction

The main conceptual contribution in this paper is to present an approach to knowledge representation and reasonings based on labeled graphs and labeled graphs homomorphism. The kernel of this approach is the so-called *simple graphs model*, which has three essential characteristics: objects are *labeled graphs*; reasonings are based on graph operations and basically on labeled graph *homomorphism*; the model is *logically* founded. This basic model has been extended in several ways, keeping these fundamental properties.

Technical contributions in this paper are mainly the followings.

- Fundamental results about the simple graphs model are synthesized and revisited.
- It is shown that the basic deduction problem on simple graphs (comparing two graphs by subsumption) is essentially the same problem as conjunctive query containment in databases and constraint satisfaction, despite their very different formulation; polynomial parcimonious transformations between these problems are exhibited.
- Grounded on the simple graphs model, the *constrained derivation model*, which allows to deal with facts, production rules, transformation rules, and constraints, is presented.

B. Ganter and G.W. Mineau (Eds.): ICCS 2000, LNAI 1867, pp. 172–192, 2000.
© Springer-Verlag Berlin Heidelberg 2000

A part of technical results provided in this paper are new, or have been published in French only. The paper is organized as follows. Next section is devoted to basic notions and results about the simple graphs model. Then, correspondences between the simple graph deduction problem and, on one hand the conjunctive query containment problem in databases and, on the other hand, the constraint satisfaction problem, are exhibited. Section 3 introduces the constrained derivation model. We finally discuss benefits and limitations of our approach.

1 The Simple Graphs Model

We assume the reader is familiar with the very basic notions about conceptual graphs, namely simple graphs (conceptual graphs without negation), their logical translation by Φ, the projection operator and co-reference links [Sow84].

1.1 Simple Graphs

Basic ontological knowledge is encoded in a structure we call a **support**. We consider here a primitive version of a support with 3 components, say $S = (T_C, T_R, I)$. T_C and T_R are finite partially ordered sets and respectively denote the relation and concept type sets. T_C possesses a greatest element, noted $\top$. Relation types may have any arity greater or equal to 1, and two comparable relation types must have the same arity. I is a set of individual markers. The following partial order is defined on the set of markers $I \cup \{*\}$, where $*$ denotes the generic marker: $*$ is the greatest element and elements of I are pairwise noncomparable. $G = (R,\ C,\ E,\ l)$ denotes a **simple graph**, where R and C are the relation and concept node sets, E is the set of edges (with a total order on edges incident to a relation node), and l is the node labeling mapping. A simple graph is not necessarily connected. Also note that it may be empty. Relation node labels are partially ordered by the partial order on T_R. Concept node labels are partially ordered by the product of the partial order on T_C and the partial order on the set of markers, $I \cup \{*\}$ (a label $(t,\ m)$ is less or equal to a label (t', m') if $t \leq t'$ and $m \leq m'$).

For simplicity reasons and because they do not add expressive power to simple graphs, we first exclude co-reference links from the basic model. However, we will show at the end of this section that all results extend to simple graphs with co-reference links.

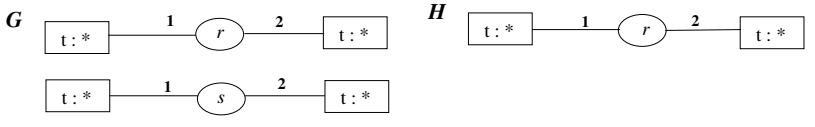
1.2 The $\leq$ Relation

The *specialization/generalization* relation (denoted by $\leq$ or $\geq$) — or *subsumption* relation — is the fundamental notion for reasoning with simple graphs. This relation can be defined in terms of a sequence of elementary graph operations: given two simple graphs G and H, $H \leq G$ (G subsumes H) if H can be obtained from G by a sequence of such operations. It can also be computed by a global operation, the so-called *projection*, which essentially is a graph homorphism.

In the original presentation of conceptual graphs [Sow84], simple graphs are *connected* graphs. Specialization operations (called canonical formation rules) are defined and it is proven that, given two (connected) simple graphs G and H, $H \leq G$ implies the existence of a projection from G to H. [CM92] proves the reciprocal. See also [MC93] or [Mug95] for a precise definition and a study of the notion of a specialization derivation.

In the proposed standard for conceptual graphs [Sow99], simple graphs are no more necessarily connected graphs (and may include co-reference links, but this aspect will be taken into account later). A new set of canonical formation rules is proposed: two equivalence rules (copy and its inverse, simplify), two specialization rules (restrict and join) and two generalization rules (unrestrict and detach), which are the inverse of specialization rules. These rules are said to be the foundation for all logical operations on CGs.

Indeed, the set of equivalence and specialization rules (or the set of equivalence and generalization rules) is logically *sound*, i.e. if H is obtained from G using these rules (say $H \leq G$) then $\Phi(H)$ implies $\Phi(G)$ — keeping into account $\Phi(S)$, the logical interpretation of partial orders on types in the support S. But it is *not complete*, in the sense that it does not allow to compute all logical deductions (see Figure 1).



$\Phi(G)$ implies $\Phi(H)$, but there is no mean with the proposed canonical formation rules of deducing one graph from the other one.

Fig. 1. H should subsume G

Let us propose another set of rules, which ensures the desirable soundness and completeness properties (this set has originally been defined in [CM95]; here, some operation names have been changed in order to avoid confusion with [Sow99]). There are four specialization operations and four inverse operations, called generalization operations. In both cases, the first operation produces a graph equivalent to the original graph (where "equivalent" can be understood in the sense of the $\leq$ relation or in a logical meaning). For all operations, the resulting graph is disjoint from the operand graph(s).

Specialization operations. Let G be a simple graph; a simple graph H can be obtained from G by:

- **Relation simplify**: delete a duplicate relation node of G (i.e. a relation node r such that there is another relation node with the same type and exactly the same i-th neighbors, for all i).

- **Restrict**: decrease the label of a relation node (its type) or the label of a concept node (its type and/or its marker).
- **Join**: merge two concept nodes of G with same label.
- **Disjoint sum**: Let K be a simple graph disjoint from G; H is obtained by juxtaposing G and K.

Generalization operations. Let G be a simple graph, a simple graph H can be obtained from G by:

- **Relation duplicate**: duplicate a relation node of G (i.e. adds a relation node with the same type and exactly the same i-th neighbors, for all i, as an existing relation node).
- **Unrestrict**: increase the label of a relation or concept of G.
- **Detach**: split a concept node c into nodes c_1 and c_2, with the same label as c, such that the union of the sets of edges incident to c_1 and c_2 is the edge set of c —one of the two sets may be empty.
- **Substract**: delete some (possibly all) connected components of G.

H is a specialization of G ($H \leq G$) if H can be derived from G by specialization operations. Since disjoint sum is a binary operation, the derivation of H from G may have other sources than H. G is a generalization of H ($G \geq H$) if G can be derived from H by a sequence of generalization operations. Note that if $G \geq H$, then G can be obtained from H by considering H only (in terms of [MC93], the derivation from G to H is a path, and not a graph with several sources). It is straightforward to prove that $G \leq H$ *iff* $G \geq H$.

Notice both sets of specialization and generalisation operations are *minimal* (no rule can be simulated with the other rules) and correspond to very *elementary* operations.

1.3 Projection

Projection [Sow84] is the fundamental operation on simple graphs since it allows the *effective* computation of the $\leq$ relation. This operation is essentially a labeled graph homomorphism. Let us recall that a *graph homorphism* (or morphism) Π from a graph $G = (X_G,\ E_G)$ to a graph $H = (X_H,\ E_H)$, where X and E respectively denote the node set and the edge set of a graph, is a mapping from X_G to X_H which preserves node adjacency, i.e. if xy is an edge of E_G, then $\Pi(x)\Pi(y)$ is an edge of E_H. A *labeled graph homomorphism* is a graph homomorphism which in addition preserves labels (on nodes, and edges if the graph is labeled on both). A *projection* is a generalization of a labeled graph morphism; a partial order on node labels is added and the following additional condition has to be satisfied: for any node x, the label of x is greater or equal to the label of $\Pi(x)$. When the partial order is trivial (all labels are noncomparable), one comes back to a classical labeled graph homomorphism. Note that this definition ensures that the node bipartition and the ordering on edges incident to a relation node are preserved.

Theorem 1. *Let G and H be two simple graphs; $H \leq G$* iff *there is a projection from G to H.*

Proof. The proofs of [Sow84] and [CM92] — which apply to connected simple graphs — are trivially adapted.

Finally, let us recall some results about the *structure* of the specialization relation:

1. $\leq$ is a preorder relation (and not an order) (e.g. [Jac88])
2. Let us say that two simple graphs G and H are *equivalent* if they subsume each other; an *irredundant* graph is a graph that is not equivalent to one of its subgraphs. Then any equivalence class for $\leq$ contains a *unique* irredundant graph (up to isomorphism); moreover this graph is the *smallest* graph of the equivalence class
3. The restriction of $\leq$ to irredundant graphs is a *lattice*.

Proof. See [CM92] for property 2 and [CM95] for property 3.

1.4 Soundness and Completeness of Graph Operations

The semantic Φ assigns a first-order formula $\Phi(G)$ to each graph G [Sow84]. $\Phi(G)$ is a positive, conjunctive and existentially closed formula. And to a support S is assigned a set of formulas, $\Phi(S)$, which corresponds to the interpretation of the partial orderings of T_R and T_C. For all types t_1 and t_2 such that t_1 covers t_2, one has the formula $\forall x_1 ... x_p (t_2(x_1, \ ..., x_p) \rightarrow t_1(x_1, \ ..., x_p))$, where $p = 1$ for concept types, and p is otherwise the arity of the relation.

Theorem 2. *(Soundness) Given two simple graphs G and H defined on S, if G projects to H then $\Phi(S), \Phi(H) \models \Phi(G)$.*

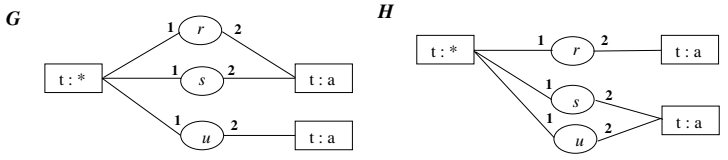
Proof. Similar to the proof of [Sow84] for connected graphs.

Theorem 3. *(Completeness) If H is in normal form (i.e. there are no two nodes with the same individual marker) and $\Phi(S), \Phi(H) \models \Phi(G)$ then G projects to H.*

Proof. See [CM95]. The proof is similar to that of [CM92] for connected graphs. Notice that if H is not in normal form the result is false: see Figure 2.

The normal form of a graph is easily computable (merge all individual nodes with same marker). However, depending on constraints on types associated with individual markers, it may be impossible to compute a type corresponding to the merging of two nodes. In the simplest case, when individual markers always appear with the same type in a concept node, any graph possesses a normal form, and the logical formula assigned to the normal form of a graph is trivially equivalent to that of the graph.

Two alternative logical semantics have been proposed in [PMC98] and [CMS98] [Sim98]. [PMC98] defines a new logic, in which deduction is equivalent to projection. [CMS98] [Sim98] proposes a variant of Φ, called Ψ, for which projection is sound and complete w.r.t. FOL deduction without any restriction.



$\Phi(G) = \Phi(H) = \exists x t(x) \wedge t(a) \wedge t(a) \wedge r(x,\ a) \wedge s(x,\ a) \wedge u(x,\ a)$

G and H have same logical translation but they are noncomparable by projection.

Fig. 2. The need for normality

These semantics rely on the same intuitive idea: if several nodes represent the same entity, they *a priori* correspond to several different viewpoints on this entity; thus, the graph obtained by merging these nodes is not considered as equivalent to the original one.

1.5 Co-reference Links

Co-reference links do not add expressive power to the simple graph model. Indeed, there are two ways of translating them in FOL. If co-reference links allowed between *generic* concept nodes only, the same variable can be associated to co-referent nodes. If co-reference links can link generic and individual nodes, the equality operator is needed to express that terms associated with these nodes are equal. In both cases, the logical formula assigned to a graph with co-reference links is equivalent to that of the graph obtained by merging all co-referent nodes (provided that these nodes have the same type)[1].

More generally, let us say that two nodes are *co-identical* if they refer to the same entity, i.e. they are generic nodes related by a co-reference link or individual nodes with the same individual marker. Formally, co-identity is an equivalence relation on concept node set of a graph, denoted by *co-ident*. The normal form of a simple graph with co-reference links is obtained by merging co-identical nodes. In other words, a graph is in normal form if the co-identity relation on its concept nodes is the identity. The logical formula associated with a graph is equivalent to that associated to its normal form.

Let us extend previous canonical formation rules in order to deal with co-reference links. The set of specialization rules (Relation simplify, Restrict, Join, Disjoint Sum) is added with a new rule (Co-reference addition), and, since a part of the Join operation becomes redundant, Join is replaced with a rule merging co-identical nodes only (Co-identical join); from a logical viewpoint, it becomes an equivalence rule.

[1] Do nodes refering to the same entity necessarily have the same type? Since we do not want to tackle this question here, we consider simple ways of avoiding it: (1) an individual marker always appears with the same type; (2) coreferent nodes have the same type; (3) coreferent nodes are generic nodes (note however we could allow co-reference between individual and generic nodes with very slight changes in definitions).

- **Co-reference addition:** add a co-reference link between two generic concept nodes with same type.
- **Co-identical join:** merge two co-identical nodes of the same graph, i.e. merge two co-referent nodes or two nodes with the same individual marker.

The set of generalization rules is modified in the same way. A new rule is added (Co-reference deletion) and Detach is replaced by an equivalence rule (Co-identical split), splitting a node into two co-identical nodes.

- **Co-reference deletion:** delete a co-reference link.
- **Co-identical split:** split a node into two co-identical nodes with same label, i.e. split an individual node into two nodes with the same label, or split a generic node into two nodes with the same label related by a co-reference link. The union of the sets of edges incident to the two nodes is the edge set of the original node —one of the two sets may be empty.

Now, projection has to respect co-identity, i.e. two coreferent nodes have co-identical images (either the same image, or distinct images which are coreferent nodes or nodes with the same individual marker). Note that previous results about simple graphs (theorems 1, 2 and 3) extend to simple graphs with coreference links.

1.6 Note about Projection and Logical Completeness

Our approach gives greater importance to projection than to the set of canonical formation rules; the price to pay is that graphs have to be put in normal form for ensuring logical completeness (w.r.t. Φ; let us recall that in [Sim98] an alternative semantic Ψ is proposed for which the completeness results hold without restriction). Suppose one rather aims at providing complete formation rules with respect to Φ, without any restriction on the form of the graphs. Such a result can be achieved with the rules we propose in the following manner.

Property 1. *Consider the set of specialization rules (co-reference addition, relation simplify, restrict, co-identical join, disjoint sum) added with co-identical split. Then, given two simple graphs (possibly with coreference links) G and H, H can be derived from G if and only if $\Phi(S), \Phi(H) \vDash \Phi(G)$.*

A similar property is obtained with the set of generalization rules added with co-identical join. These properties can be explained as follows. Let us consider the structuration of the simple graph space by the projection operation (a graph G is said to be more general than a graph H if G projects onto H). Let us consider two extremal transformations of a graph: the *normal* form (no co-identical join can be done) and the *anti-normal* form (see [GW95]: the graph is split into star graphs, composed of one relation and its neighbors, all distinct, related with co-reference links; in other words, no co-identical split can be done). Following properties are immediate:

- The anti-normal form and the normal form of a graph are unique (up to isomorphism).

- If two graphs have the same normal form, they also have the same anti-normal form; to one normal form corresponds one anti-normal form, and reciprocally.
- G projects onto its normal form and the anti-normal form of G projects onto G
- All graphs with same anti-normal (or normal form) are logically equivalent by Φ.

If we possess both rules "co-identical join" and "co-identical split", we can search the whole space of graphs bounded by an anti-normal graph and the corresponding normal form. Note that the projection completeness theorem (theorem 3) could also be rewritten by replacing the normal form condition on the target graph H with an anti-normal form condition on the source graph G:

Theorem 4. *(Completeness) If G is in anti-normal form (i.e. no co-identical split can be done on G) and $\Phi(S), \Phi(H) \vDash \Phi(G)$ then G projects to H.*

To conclude this note, either one favours projection, and logical completeness of graphs operations (projection or an equivalent set of formation rules) is obtained with (minor) restrictions on the form of graphs; or one favours logical completeness of the set of formation rules, and the correspondence with projection is lost.

1.7 Note about the Proposed Formation Rules in [Sow99]

Previous rules present two advantages w.r.t. rules proposed in [Sow99]: logical completeness and simplicity. Specially, the equivalence rules of [Sow99], *copy* and *simplify*, could be decomposed into more elementary equivalence operations, respectively relation duplicate and co-identical split, and co-identical join and relation simplify (see Figure 3).

Let us consider for instance the *copy* rule definition (*simplify* is the inverse of copy): "the copy rule makes a copy of any subgraph v of u and adds it to u to form w. If c is any concept of v that has been copied from a concept d in u, then c must be a member of exactly the same coreference sets as d. Some conceptual relations of v may be linked to concepts of u that are not in v; the copies of those relations must be linked to exactly the same concepts of u". Copy can be simulated by a sequence of relation duplicate (for all copied relations) followed by a sequence of co-identical splits (for all copied concepts).

Let us now come back to simple graphs without co-reference links, and to the basic deduction problem: given two simple graphs G and H, is there a projection from G to H? The next section shows that this problem is essentially the same as two other basic problems in databases and artificial intelligence.

2 Projection, Conjunctive-Query Containment and Constraint Satisfaction

Conjunctive queries constitute a broad class of frequently used queries in databases because their expressive power is equivalent to that of the Select-Join-

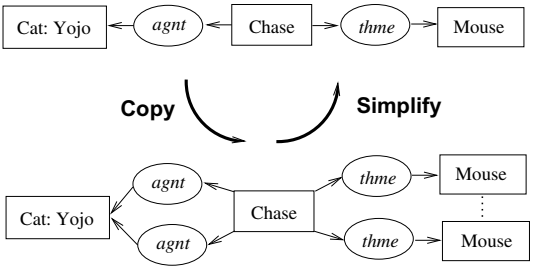


Fig. 3. Proposed equivalence rules in [Sow99]

Project queries in relational databases [AHV95]. And **conjunctive query containment** is a fundamental problem in database query evaluation and optimization. **Constraint satisfaction problems** are a class of combinatorial problems widely investigated in artificial intelligence. What do these problems have in common, and what do they have in common with projection between simple conceptual graphs? This section shows that the three problems are essentially the same problem, even if they are formulated in very different ways.

The key notion explaining correspondences between these problems is that of a labeled graph homomorphism. Let us recall that a labeled graph homomorphism from a labeled graph G to a labeled graph H is a mapping from G nodes to H nodes that preserves edges and labels. We will see that the three problems can be formulated in terms of a labeled graph homomorphism problem. Such a result is not surprising in fact because the homomorphism notion is the fundamental notion between relational structures (which are the more general algebra structures). The fundamental algebraic problem is the following: given two finite relational structures A and B is there a homomorphism from A to B? This problem can be immediately recast as a labeled graph homomorphism problem since a finite relational structure can be naturally formulated as a labeled graph of the same size (which in fact can be seen as a simple conceptual graph).

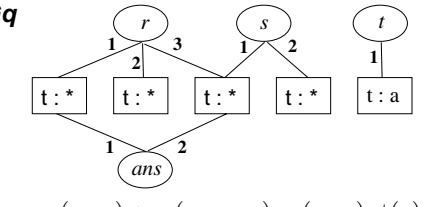
Let us call a **simplified support** a support $S = (T_C, T_R, I)$ where T_C and T_R are provided with the trivial partial order, i.e. all elements of T_C and T_R are non comparable. For establishing equivalencies between problems we will consider simplified supports; at the end of this section, we will reconsider the introduction of partial orders on types. The **Projection** problem is given by a simplified support S, and two simple graphs in normal form G and H relative to S, and asks if there is a projection from G to H. As already noticed, since S is a simplified support, the question comes down to ask wether there is a labeled graph homomorphism from G to H.

This section first presents polynomial transformations between Projection and Conjunctive query containment, then polynomial transformations between Projection and Constraint satisfaction. All mappings are parcimonious transformations (they preserve the set of solutions). Thus, algorithms for building a

solution or for enumerating all solutions can be directly transferred from one domain to the other one.

2.1 Projection and Conjunctive-Query Containment

Let us first recall some basic database definitions about conjunctive queries. We follow here notations of [AHV95]. Let R denote a set of n-adic relation names ($n \geq 1$) and let dom denote a countably infinite set of constants. A (positive and non-recursive) conjunctive query q is usually written as a rule, $q = ans(u) \leftarrow r_1(u_1),\ ...\ r_n(u_n),\ n \geq 1$, where r_1 ... r_n belong to R, ans is not in R (ans stands for "answer"), u and u_1 ... u_n are tuples of terms (variables or constants of dom), and each variable of u occurs at least once in u_1 ... u_n. Notice that u may be empty. A database instance α on R maps each p-ary relation of R to a finite subset of dom^p. Given a query $q = ans(u) \leftarrow r_1(u_1),\ ...\ r_n(u_n)$ and an instance α on R, $q(\alpha)$ denotes the set of answers to q in α; $q(\alpha)$ is the set of tuples $\mu(u)$ where μ is a substitution of the variables of q by dom constants such that for any j in $\{1, ..., n\}$, $\mu(u_j) \in \alpha(r_j)$. When the arity of ans is 0, $q(\alpha)$ is the set $\{()\}$ if there is a substitution μ such that for any j in $\{1, ..., n\}$, $\mu(u_j) \in \alpha(r_j)$ and otherwise $\emptyset$. A query q is said to contain a query q' ($q' \subseteq q$) if, for any instance α on R, $q'(\alpha) \subseteq q(\alpha)$. The **Query containment** problem is the associated problem: given two queries q and q', does q contain q'? It is well-known in databases that the query containment problem can be reformulated as a homomorphism problem, where a homomorphism is defined as follows. A **homomorphism** from $q = ans(u) \leftarrow r_1(u_1),\ ...\ r_n(u_n)$ to $q' = ans'(u') \leftarrow r'_1(u'_1),\ ...\ r'_{n'}(u'_{n'})$ is a substitution θ of the variables of q by terms (variables or constants of dom) such that $\theta(u) = u'$ and for any j in $\{1, ..., n\}$, there is i in $\{1, ..., n'\}$ such that $\theta(r_j(u_j)) = r'_i(u'_i)$. The homomorphism theorem [AHV95] proves that, given two queries q and q', q contains q' iff there is a homomorphism from q to q'.



$q = ans(x,\ z) \leftarrow r(x,\ y,\ z),\ s(z,\ v)$, t(a)

Fig. 4. Transformation from a query to a graph

Transformation from Query Contaiment to Projection (Q2P). Let $S = (T_C,\ T_R,\ I)$ be a simplified support built as follows. T_C is restricted to a single

type. T_R is equal to $R \cup Ans$, where $Ans = \{ans_k |\ k \geq 1\}$ and k stands for the arity of ans. I is equal to dom. Elements of T_R are noncomparable. Then a simple graph (in normal form) $G_q = (R_q, C_q, E_q, l_q)$ can be assigned to q as follows (see Figure 4 for an example). R_q is a set of (n+1) nodes labeled by $r_1,\ ...,\ r_n,\ ans_k$. C_q is in bijection with the set of terms occurring in $u,\ u_1,\ ...,\ u_n$. The node of C_q assigned to a term e is generic if e is a variable and otherwise individual with the referent e. And if e is the i-th argument of r_j (or ans), then the concept node assigned to e is the i-th neighbor of the relation node assigned to r_j (or ans). Then, one can immediately check that, given two queries, q and q', every homomorphism from q to q' is a projection from G_q to $G_{q'}$, and reciprocally.

Moreover, it can be proven that the query homomorphism theorem ([AHV95], 6.2.3) is equivalent to the soundness and completeness theorem for projection between simple graphs (in normal form) [CM92], the minimization theorem ([AHV95], 6.2.6) is equivalent to the theorem about irredundant simple graphs [CM92], and the complexity theorem for query decision problems ([AHV95], 6.2.10) is equivalent to the complexity theorem for simple graphs decision problems [CM92]. See [SCM98] for more detail about these equivalencies.

Transformation from Projection to Query Containment (P2Q). A simplified support $S = (T_C,\ T_R,\ I)$ naturally induces a database; the database relation names are given by the union of T_C and T_R (the arity of an element of T_C is 1), and dom is a countable infinite set which contains the individual marker set I. A query q_G is built from a normal simple graph G in a manner similar to $\Phi(G)$; more specifically, $q_G = ans() \leftarrow r_1(u_1)\ ...\ r_n(u_n)$, where $r_1(u_1)\ ...\ r_n(u_n)$ are the atoms of $\Phi(G)$. Given two normal simple graphs G and H, it can be easily checked that any homomorphism from q_G to q_H is a projection from G onto H, and reciprocally.

Theorem 5. *The problems* Conjunctive query containment *and* Projection *are polynomially equivalent; furthermore by both transformations Q2P and P2Q, every solution to one problem is a solution to the other one.*

2.2 Projection and Constraint Satisfaction

The input of a constraint satisfaction problem (CSP) is a set of variables, a set of possible values for the variables and a set of constraints between the variables. The question is to determine whether there is an assignment of values to the variables that satisfies the constraints. More specifically, a *constraint network* $P = (X,\ C,\ D,\ R)$ consists of (see Figure 5):

- a set of variables, $X = \{x_1,\ ...\ , x_n\}$
- a set of constraints, $C = \{C_1,\ ...\ , C_p\} \subset \mathcal{P}(X)$,
  When constraints are binary, i.e. involve two variables, the pair $(X,\ C)$ is generally seen as a non directed graph, whose nodes are the variables, and edges are the constraints. In the general case, $(X,\ C)$ is considered as an hypergraph.

- a set of variable domains, $D = D_1 \cup ... \cup D_n$, where D_i is the domain of variable x_i.
- a set of constraint definitions, $R = \{R_1, ..., R_p\}$; for every constraint C_i, let $(x_{i_1} ... x_{i_q})$ be any ordering of C_i variables, then $R_i \subset D_{i_1} \times ... \times D_{i_q}$.

A *solution* to P is an assignment of values to the n variables such that all constraints are satisfied, more formally:

a mapping $\mathcal{I}$: $X \to D$
$x_i \mapsto a \in D_i$,
and, for every constraint $C_j = (x_{j_1} ... x_{j_q})$, $(\mathcal{I}(x_{j_1}) ... \mathcal{I}(x_{j_q})) \in R_j$.

A *constraint satisfaction problem (CSP)* is the associated problem: given a constraint network P, is there a solution to P?

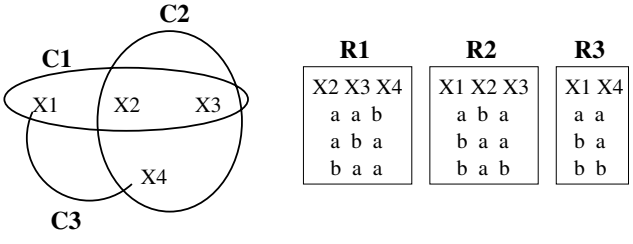
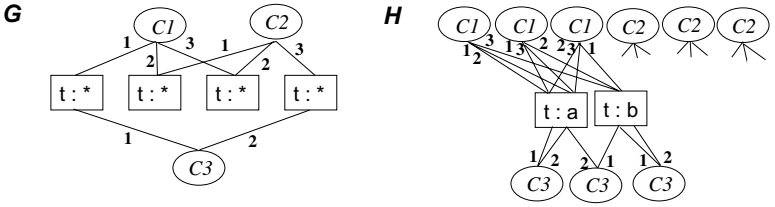


Fig. 5. An instance of a CSP



G and H are obtained from the constraint satisfaction network of Figure 5 by the transformation C2P.

Fig. 6. Transformation from CSP to Projection

Transformation from Constraint Satisfaction to Projection (C2P). A CSP can be recast as a Projection problem by the following transformation, say $C2P$ (see Figure 6). Consider a constraint network $P = (X, C, D, R)$. Let us build a simplified support $S = (T_C, T_R, I)$ from P as follows: T_C is restricted

to one element, say $\top$, T_R is equal to C, each relation type having the same arity as the corresponding constraint, and I is equal to D. G and H are then defined as follows. The concept node set of G is equal to X. Each concept node is labeled by $(\top, *)$. There is one relation node for each constraint, labeled by the corresponding relation type. The i-th neighbor of a relation node is the concept node corresponding to the i-th variable of the constraint. The concept node set of H is D. Each concept node is labeled by $(\top, a)$, where a is the associated value of D. There is one relation node for each tuple of a constraint definition; this relation node is labeled by the name of the constraint, and its i-th neighbor is the concept node corresponding to the i-th value. Notice that both graphs G and H are in normal form. It can be easily checked that any solution to the CSP is a projection from G to H, and reciprocally.

Transformation from Projection to Constraint Satisfaction (P2C) In turn, a Projection problem can be recast as a CSP by the following transformation, say $P2C$. There is one variable for each concept node of G, and one constraint for each relation node of G. The set D of domain values is built from the concept node labels of H: there is one value for each distinct label in H. The domain D_i of a variable x_i is the subset of D to *a priori* possible images for the concept node c_i by a projection from G to H, i.e. $D_i = \{l = (t, m)\ s.t.\ label(c_i) \geq l\}$. The definition of a constraint C_i coming from a relation node r_i is the set of tuples given by the relation nodes of H with same label as r_i, i.e. if C_i is a constraint over variables $(x_{i_1}\ ...\ x_{i_q})$, then its definition is $R_i = \{(l_{i_1}\ ...\ l_{i_q})$ where $l_{i_j} = label(c_{i_j})$ s.t. $c_{i_1}\ ...\ c_{i_q}$ are neighbors, in this order, of a relation node r in H with $label(r) = label(r_i)\}$. It can be easily checked that any projection from G to H is a solution to the CSP, and reciprocally.

Theorem 6. *The problems* CSP *and* Projection *are polynomially equivalent; furthermore by both transformations C2P and P2C, every solution to one problem is a solution to the other one.*

Note about general supports. What if we consider simple conceptual graphs on general supports instead of simplified supports? The translations C2P and Q2P from CSP and Conjunctive query containment to Projection still hold, since a simplified support is a support. The translation P2C can be adapted in a straightforward way: the tuples of the definition of a constraint C_i coming from a relation node r_i of G are given by relation nodes of H with a label *less or equal* to the label of r_i (instead of *equal to*). For P2Q, G and H on a support S can be translated into G' and H' on a simplified support S' such that every projection from G to H is a projection from G' to H', and reciprocally. See for instance [Bag99]; the size of the new input $(S',\ G',\ H')$ is linear in the size of the old input $(S,\ G,\ H)$.

Relationships with published works. The translation Q2P has been published in [CMS98]. A first version of the correspondences C2P and P2C can be found in [CM95] (not published) or in [MC96] (in French). In these previous

versions, correspondences are done between labeled graph morphism and binary CSP, but they rely on the same ideas. Independently, in [KV98] it is shown that conjunctive query containment and CSP are essentially the same problem because they can be recast as a relational structure homomorphism.

3 The Constrained Derivation Model

It would be out of the scope of this paper to present in detail this extension to the simple graph model (see [BGM99]). The aim of the following presentation is rather to illustrate our approach. The constrained derivation model is composed of three kinds of objects: simple graphs, rules and constraints. Simple graphs are the basic constructs from which rules and constraints are defined. Projection is the basic operation from which reasonings with rules and constraints are defined.

- In this model, a *simple graph* is interpreted as a fact or a query.
- A *rule* allows one to add new knowledge. It is composed of an hypothesis and a conclusion, and is used in the following classical way: given a simple graph, if the hypothesis of the rule projects to the graph, then the information contained in the conclusion is added to the graph. Rules are split into *production rules* and *transformation rules*.
- A *constraint* defines conditions for a simple graph to be valid. It is composed of a condition part and a mandatory part. Roughly said, a graph satisfies a constraint if for every projection of its condition part, its mandatory part also projects to the graph.

The deduction problem in this framework can be outlined as follows: given a *knowledge base* (composed of facts, production rules, transformation rules and constraints) and a *query*, is it possible for this knowledge base to *evolve* in such a way that the query is satisfied? More specifically, facts describe an initial world. Production rules complete world descriptions. Transformation rules define possible transitions from a world to other ones. Constraints are used to check validity of worlds. A successor of a valid world is obtained by a single application of a transformation rule on this world. Solving the problem is to find a chain of valid worlds evolving from the initial one such that the query projects to the last one.

All examples of this section are taken from the modelization of the Sysiphus-I case study (a problem of allocation of offices to members of a compagny) with the constrained derivation framework [BGM99].

3.1 Rules

A rule "If H then C" is classically defined as a couple of simple graphs, H and C, respectively called hypothesis and conclusion of the rule, sharing some generic concept nodes (called here frontier nodes). Briefly said, the logical semantics Φ is extended to rules in the following way: let x_1 ... x_p be the variables assigned to frontier nodes; let $\Phi'(H)$ and $\Phi'(C)$ be formulas obtained from $\Phi(H)$ and

$\Phi(C)$ by removing existential quantification of x_1 ... x_p. Then, given a rule R, $\Phi(R) = \forall x_1 \ ... \ \forall x_p \ \Phi'(H) \rightarrow \Phi'(C)$.

[Bag99] provides a graphical visualization of a rule as a simple graph provided with a coloring of its nodes with two colors, say 0 and 1. In drawings, 0-colored nodes are painted in white, and the others in black. The subgraph induced by the color 0 nodes must be a syntactically correct simple graph. Nodes with color 0 are the hypothesis nodes and make up the *hypothesis* of the rule. Concept nodes of color 0 with at least one neighbor outside the hypothesis part are the *frontier* nodes. Nodes with color 1 are the conclusion nodes and, together with the frontier nodes, they make up the *conclusion* of the rule. See in Figure 7 rules R_1 and R_2 expressing that "near" is a reflexive and symmetric relation.

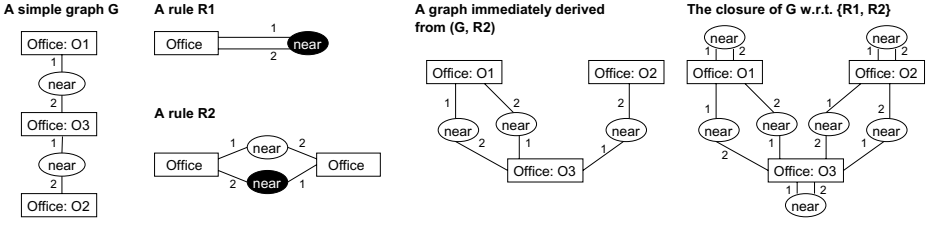


Fig. 7. Rules, derivation and closure

A rule R is *applicable* to a simple graph G if there is a projection, say Π, from the hypothesis of R to G. In this case, the result of the application of R to G following Π is the simple graph G' obtained from G and the conclusion of R by restricting the label of each frontier node c in the conclusion to the label of its image $\Pi(c)$ in G, then joining c to $\Pi(c)$. In this case, we say that G' is *immediately derived* from (G, R) (see Figure 7). Let $\mathcal{R}$ be a set of rules. G' is said to be *immediately derived* from $(G, \mathcal{R})$ if there exists a rule R in $\mathcal{R}$ such that G' is immediately derived from (G, R). A graph G' is said to be *derived* from $(G, \mathcal{R})$ if there exists $G = G_0, G_1, \ldots, G_k = G'$ such that each G_{i+1} is immediately derived from $(G_i, \mathcal{R})$.

Theorem 7. *(Soundness and compleness of forward chaining) Let G and H be simple graphs and $\mathcal{R}$ be a set of rules, all relative to a support S.*

- *if H is derived from $(G, \mathcal{R})$ then $\Phi(S),\ \Phi(\mathcal{R}),\ \Phi(G) \vDash \Phi(H)$*
- *provided that G and all rules of $\mathcal{R}$ are in normal form, and the graph obtained at each derivation step is put into normal form: if $\Phi(S),\ \Phi(\mathcal{R}),\ \Phi(G) \vDash \Phi(H)$ then there exists a simple graph H' that can be derived from $(G, \mathcal{R})$ such that H projects onto H'.*

Proof. See [SM96]. Note that [SM96] also defines backward chaining operations, and proves associated soundness and completeness results.

3.2 Constraints

We consider *positive* and *negative* constraints. A positive constraint expresses that "if information A is present, then information B must also be present". A negative constraint expresses that "if information A is present, then information B must be absent". Formally, a constraint is defined in the same way as a rule, as a simple graph provided with two colors, 0 and 1. Nodes colored in 0 form the condition part of the constraint. A given graph G satisfies a *positive* constraint C if *every* projection from the condition part of C into G can be extended to a projection of the whole graph C into G. And G satisfies a *negative* constraint C if *no* projection from the condition part of C into G can be extended to a projection of the whole graph C into G. A graph G is said to be valid w.r.t. a set of constraints $\mathcal{C}$ if it satisfies all constraints of $\mathcal{C}$.

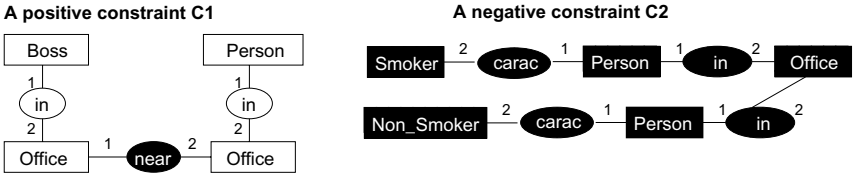


Fig. 8. Positive and negative constraints

Fig. 8 presents two constraints: a positive constraint *C1*, expressing that "a boss office should be near all offices" (more litterally: "if a *boss* is *in* an *office* and a *person* is *in* an *office*, then this latter office must be *near* the *boss office*"), and a negative constraint, *C2*, expressing that "a smoker and a non smoker can not share an office" (the condition part is here empty). Note that more generally we could restrict negative constraints to negative constraints with an empty condition. Indeed, given a constraint C and a graph G, the statement "no projection from the condition part of C to G can be extended to a projection of C into G" is equivalent to "there is no projection from C to G". Also note that a more general form of constraint can be handled, corresponding to the following intuitive semantics "if information A is present and information B is absent then information C must be present and information D must be absent". It is defined as a simple graph provided with four colors, one for each part A, B, C and D.

3.3 Constrained Derivation

Consider that asserted facts (represented by a simple graph F) and production rules (represented by a set of rules $\mathcal{R}$) describe a world, say W. The associated deduction problem (given a query Q and a world W, is Q deducible from W, i.e. does there exist a derivation from $(F, \mathcal{R})$ to a graph F' such that Q can be projected into F'?) is a semi-decidable problem [CS98]. Let us restrict ourselves to decidable cases. Such a case occurs for instance if we restrict production rules

to rules that do not add new concept nodes (the conclusion of a rule is composed of frontier nodes and relation nodes; or in logical terms, the formula assigned to the conclusion does not add new variables). It is then possible to compute the *closure* of F w.r.t. $\mathcal{R}$. Intuitively, a graph G is closed w.r.t. $\mathcal{R}$ if no application of a rule of $\mathcal{R}$ to G adds information. More formally, for any rule R of $\mathcal{R}$, if R can be applied onto G (there is a projection Π from the hypothesis of R into G) then the resulting graph is equivalent to G (Π can be extended to a projection of the whole graph R into G). Given a graph F and a set of rules $\mathcal{R}$, a closure of F w.r.t. $\mathcal{R}$ is a graph derived from $(F, \mathcal{R})$ closed w.r.t. $\mathcal{R}$. All closures are equivalent for projection; we call *the* closure of F by $\mathcal{R}$ the (unique) smallest graph of this set (see Figure 7). Now, a world $W = (F, \mathcal{R})$ is said to be valid w.r.t. a set of constraints $\mathcal{C}$ if the closure of F by $\mathcal{R}$ is valid w.r.t. $\mathcal{C}$.

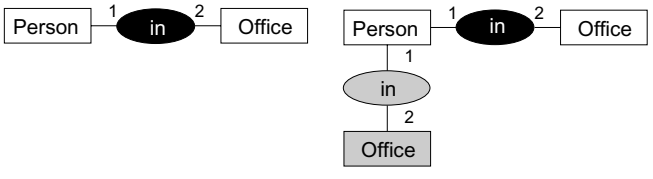


Fig. 9. Transformation rules

Let us add a set of rules, called *transformation rules*, whose role is to generate new worlds. Transformation rules are rules as defined above but the intuitive semantic becomes "if information A is present in a world, then generate a new world by adding information B to this world". See Figure 9: the rule on the left says "if there are a *person* and an *office*, try to put that *person* into that *office*".

A more general form of transformation rule can be handled, with hypothesis split into a positive part ("if information A is present") and a negative part ("and if information B is absent"). Such a rule is encoded by a graph with three colors. It is applicable to a graph if there is a projection from the positive part of the hypothesis onto the graph that cannot be extended to a projection of the whole hypothesis ("A is found but we cannot find B"). See Figure 9: the rule on the right says "if there are a *person* who is not already assigned to an office, and an *office*, try to put that *person* into that *office*". Note that it would not be possible to generalize in the same way production rules because the derivation mechanism would become non monotonic (for instance the "unique closure" property would not hold true anymore).

Let $W = (F, \mathcal{R})$ be a valid world w.r.t. a set of constraints $\mathcal{C}$ and let $\mathcal{T}$ be a set of transformation rules; a *successor* of W by $\mathcal{T}$ is a valid world obtained by appling one rule of $\mathcal{T}$ onto the closure of F by $\mathcal{R}$. A world W' is a *descendant* of a (valid) world W if there exists a sequence $W_0 = W$ ... $W_k = W'$ such that each W_{i+1} is a successor of W_i, $0 \leq i < k$. Such a sequence is called a constrained derivation from W to W'. The global deduction problem can be expressed as follows. The input is a world $W = (F, \mathcal{R})$, a set of constraints $\mathcal{C}$,

a set of transformation rules $\mathcal{T}$ and a query Q. The question is whether there exists a constrained derivation from W to a world W' such that Q projects to W'. Note that every world of a constrained derivation has to be valid, including W.

Finally, let us point out that the constrained derivation model could handle nested graphs [CMS98] instead of simple graphs.

4 Strengths and Weaknesses of This Graph-Based Approach

Let us first answer some questions about our approach: why are we representing knowledge with labeled graphs, and specially with conceptual graphs? Why are we doing reasonings with graph operations instead of logical operations? More specifically, why are we basing reasonings on graph homomorphism and not on diagrammatic rules *a la Peirce*?

Why labeled graphs? Different kinds of labeled graphs have long been used to represent knowledge. One of the more attractive features of labeled graphs in general for knowledge acquisition and representation is their *visual* aspect. Basically a simple conceptual graph is a bipartite labeled graph, where one class of nodes represents entities and the other represents relationships between these entities. It is thus easily editable and interpretable by a person, in particular if it has a small number of nodes and edges or if it has a special structure enhanced by the drawing. As noticed in [BBV97] for instance, in a knowledge modeling context, "Experts appreciate graphic notations. The very simple graphic concepts of Conceptual Graphs (boxes, ovals and arrows) and the use of labels chosen in the expert language make modeled knowledge understandable by the expert with no effort, whereas they were afraid by textual formal languages".

Why conceptual graphs? Several important features distinguish conceptual graphs from other labeled graph representations:

- a clear separation is made between ontological knowledge and factual knowledge;
- they are not limited to binary relations;
- they are provided with a logical semantics.

Why graph-based reasonings? Conceptual graphs are not only a *visual* language but also a *formal* knowledge representation model. They are provided with reasoning operations which are sound and complete with respect to deduction in first order logics (see above for simple graphs, [CMS98] for nested graphs, and [Sow84][Wer95][KS97] for more general graphs equivalent to first order logic).

One commonly held opinion is that conceptual graphs are — only — a graphical interface for logics. Indeed, a justified question is to ask why not simply

translate conceptual graphs into logics, use logical techniques for doing reasonings, and come back to graphs for presenting results. A related question is why base reasonings on graph homomorphism (projection) rather than on elementary graphical rules (canonical formation rules for simple graphs and diagrammatic rules *a la* Peirce for general conceptual graphs). The main argument justifying graph-based reasonings is related to knowledge modeling. Another argument is related to computational aspects.

Benefits from a modeling viewpoint. Let us summarize main qualities of the model we propose from a modeling viewpoint (for a more detailed discussion, see [BGM99]). Objects are easily understandable by an end-user *as well as reasonings*, and this for two reasons: projection is a graph-matching operation easily interpretable, and the same language is used at interface and operational levels. This property can provide a precious help to verify the validity of an expertise modeling for instance. Indeed, it is in general very difficult to ensure that the expert reasoning is correctly modeled [GS95]. Our model can aid by giving the expert the possibility to run the system and to follow reasonings step by step in order to check that what is done is correct from his/her viewpoint.

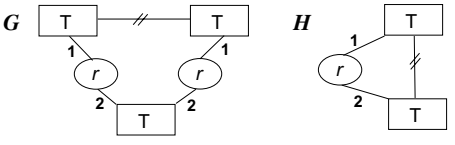
Benefits from a computational viewpoint. Having knowledge represented by graphs and reasonings based on graph theoretic notions allows one to moor to combinatorial algorithms, and to benefit from results in this domain. Also, using graphs whereas the dominating formalism is logic[2] provides a different viewpoint. For instance, just to give some immediate notions, a path, a cycle or a distance are natural notions on graphs and are not in logics. See for instance the algorithm for backward chaining of production rules based on the graph notion of a *piece* in [SM96] or the complexity result in [BMT99] (the deduction problem of a description logic, whose complexity was unknown, is shown to be solvable in polynomial time by a translation into a graph homomorphism problem).

Limitations. One question we have been exploring can be summarized as follows "how far is it possible to go in knowlegde representation and reasonings with labeled graphs and graph matching operations (with graph homomorphism in the foreground)?". Indeed, the main hypothetical limitation of our work is expressivness. At present time we are not able to give a definitive answer to this question, but can provide some partial results.

Simple graphs represent positive and conjunctive knowledge. And projection corresponds to deduction on this knowledge. It seems that, taken alone, this kind of operation (that compares two graphs by a *global* matching, instead of trying to obtain one from another by a derivation sequence) is fundamentally unable to deal with negation or disjunction in a complete way, even limited as shown below. Suppose for instance one wants to deal with *difference links*,

[2] and we do no intend here to question this pre-eminence; logic is naturally the reference formalism as soon as one aims at modelizing reasonings.

expressing that two entities represent distinct entities (the contrary of a co-reference link). A difference link is logically interpreted by the atom $\neg(x = y)$, where x and y are the terms assigned to concept nodes extremities of the difference link. One may represent difference as a relation type, say *dif* with associated production rules giving it the semantics of $\neg equal\ (\forall x \forall y dif(x, y) \rightarrow dif(y, x)$ and $\forall x \forall y \forall z dif(x, y) \wedge (y = z) \rightarrow dif(x, z))$.



$\Phi(G)$ implies $\Phi(H)$, but there is no projection from H to G

Fig. 10. Projection and difference links

In Figure 10 [3], difference links are represented by crossed lines. $\Phi(G)$ implies $\Phi(H)$ but there is no projection from H to G. Indeed, $\Phi(G) = \exists x \exists y \exists z\ r(x,\ z) \wedge r(y,\ z) \wedge \neg(x = y)$ and $\Phi(H) = \exists x \exists y\ r(x,\ y) \wedge \neg(x = y)$, and by adding to $\Phi(G)$ the subformula $((x = z) \vee \neg(x = z)) \wedge ((y = z) \vee \neg(y = z))$, one keeps an equivalent formula from which $\Phi(H)$ can be trivially deduced.

bel r.

One approach for obtaining a model equivalent to FOL (or more), while keeping projection as the basic operation is that of [KS97]. This work proposes a sound and complete reasoning method for general conceptual graphs, based on *tableaux*, which decomposes conceptual graphs into simple graphs, and combines projection onto simple graphs with tableaux rules.

The constrained derivation model outlined in this paper clearly allows some forms of negation and disjunction, while keeping objets and operations easily interpretable. The study of its expressivness is not achieved yet.

References

AHV95. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

Bag99. J.-F. Baget. A simulation of co-identity with rules in simple and nested graphs. In *Proc. ICCS'99, LNAI 1640*, pages 442—455. Springer Verlag, 1999.

BBV97. C. Bos, B. Botella, and P. Vanheeghe. Modeling and simulating human behaviors with conceptual graphs. In *ICCS'97 Proc., LNAI 1257*, pages 275–289. Springer Verlag, 1997.

BGM99. J.-F. Baget, D. Genest, and M.-L. Mugnier. Knowledge acquisition with a pure graph-based knoweldge representation model — application to the sisyphus-i case study. In *Proc. KAW'99*, 1999.

[3] This counter-example is from Geneviève Simonet (personal communication).

BMT99. F. Baader, R. Molitor, and S. Tobies. Tractable and Decidable Fragments of Conceptual Graphs. In *Proc. ICCS'99*, LNAI 1640, pages 480–493. Springer Verlag, 1999.

CM92. M. Chein and M.L. Mugnier. Conceptual Graphs: Fundamental Notions. *Revue d'Intelligence Artificielle*, 6(4):365–406, 1992.

CM95. M. Chein and M.L. Mugnier. Conceptual Graphs are also Graphs. Research Report 95-004, LIRMM, Jan. 1995. 17 pages.

CMS98. M. Chein, M.-L. Mugnier, and G. Simonet. Nested Graphs: a Graph-based Knowledge Representation Model with FOL semantics. In *Proc. KR'98*, pages 524–534. Morgan Kaufmann, 1998.

CS98. S. Coulondre and E. Salvat. Piece Resolution: Towards Larger Perspectives. In *Proc. ICCS'98, LNAI 1453*, pages 179–193. Springer Verlag, 1998.

GS95. B.R. Gaines and M.L. Shaw. Knowledge and requirements engineering. In *Proc. KAW'95*, 1995.

GW95. B.C. Ghosh and V. Wuwongse. A direct Proof Procedure for Definite Conceptual Graph programs. In *LNAI 954, Proceedings of ICCS'95*, pages 158–172. Springer Verlag, 1995.

Jac88. M.K. Jackman. Inference and the Conceptual Graph Knowledge Representation Language. In S. Moralee, editor, *Research and Development in Expert Systems IV*. Cambridge University Press, 1988.

KS97. G. Kerdiles and E. Salvat. A sound and complete proof procedure based on tableaux and projection. In *Proc. of ICCS'97, LNAI 1257*, pages 371–385. Springer Verlag, 1997.

KV98. P.G. Kolaitis and M.Y. Vardi. Conjunctive-Query Containment and Constraint Satisfaction. In *Proceedings of PODS'98*, 1998.

MC93. M.L. Mugnier and M. Chein. Characterization and Algorithmic Recognition of Canonical Conceptual Graphs. In *LNAI 699, Proc. ICCS'93*, pages 294–311. Springer Verlag, 1993.

MC96. M.L. Mugnier and M. Chein. Représenter des connaissances et raisonner avec des graphes. *Revue d'Intelligence Artificielle*, 10(1):7–56, 1996.

Mug95. M.L. Mugnier. On generalization/specialization for conceptual graphs. *J. Expt. Theor. Artif. Intell.*, 7(3):325–344, 1995.

PMC98. A. Preller, M.-L. Mugnier, and M. Chein. Logic for Nested Graphs. *Computational Intelligence*, 14(3), 1998.

SCM98. G. Simonet, M. Chein, and M.-L. Mugnier. Projection in Conceptual Graphs and Query Containment in nr-Datalog. R.R. LIRMM, 1998.

Sim98. G. Simonet. Two FOL Semantics for Simple and Nested Conceptual Graphs. In *Proc. ICCS'98*, LNAI 1453. Springer Verlag, 1998.

SM96. E. Salvat and M.L. Mugnier. Sound and complete forward and backward chainings of graph rules. In *ICCS'96 Proc., LNAI 1115*, pages 248–262. Springer Verlag, 1996.

Sow84. J.F. Sowa. *Conceptual Structures - Information Processing in Mind and Machine*. Addison-Wesley, 1984.

Sow99. J. F. Sowa. Conceptual Graphs: Draft Proposed American National Standard. In *ICCS'99 Proc., LNAI 1640*, pages 1–65. Springer Verlag, 1999.

Wer95. M. Wermelinger. Conceptual Graphs and First-Order Logic. In *Proc. of ICCS'95, LNAI 954*, 1995.

User Modelling Using Conceptual Graphs for Intelligent Agents

James F. Baldwin, Trevor P. Martin, and Aimilia Tzanavari

A.I. Group, Department of Engineering Mathematics,
University of Bristol,
Bristol BS8 1TR. UK
{Jim.Baldwin, Trevor.Martin, A.Tzanavari}@bristol.ac.uk

Abstract. We describe a novel inference method to help us deal with the problem of having sparse information about the user. The information we will have available for the user will form his/her model. That model will be represented in a conceptual graph format. We will also have gathered information about categories of users that share characteristics, preferences and interests. These will form the "prototypes", as we call them and will be represented in graphs as well. Because we will know significantly more about the prototypes, they will be the source where we will try to get the information we want for the user. This method has been implemented in FRIL [2, 3, 10].

1 Introduction

In the past software systems used to be relatively simple because technological capabilities were limited. They tended to be appropriate for performing very specific tasks only. Subsequently, a limited number of people were using them; the ones that had relevant background and needed to use them. However, nowadays things have changed: systems have become comparatively complex and the users that are meant to use them form very diverse user groups. They have different characteristics, needs, abilities, preferences and interests. As a result, software systems have to become more individualised and cater for those differences and not treat all the users in the same manner. Their scope is to make man's life as easy as possible.

User modelling [9, 12, 14] can be identified in general terms as the part of the software system's design that deals with the aforementioned problems. Since nowadays software systems structure is becoming more and more agent-based, we can safely assume that user modelling will be taken care of by either a dedicated agent, or by an agent's component.

Intelligent Agents can be defined as Personalised Assistants that “look over the user’s shoulder” and learn the user’s characteristics in order to act for his interest [13, 16]. Indeed, the user’s characteristics, which of course will be related to the software system’s purpose, form the core element of the individualisation. This information is called the user’s model. The user’s model, in combination with the relevant IA knowledge base will be used by the Intelligent Agent to predict the human responses, the human needs and suggest to the user things to do, help the user utilise his computer more efficiently.

B. Ganter and G.W. Mineau (Eds.): ICCS 2000, LNAI 1867, pp. 193-206, 2000.
© Springer-Verlag Berlin Heidelberg 2000

There are two problems related to this procedure. The first is concerned with the knowledge representation of the user's model, how we build the model, how we use the model with the other IA knowledge bases to make inferences and decisions. The second one is concerned with the fusion problem of individual inferences.

In this paper we will mainly concentrate on the first problem and discuss a form of knowledge representation and inference method, which is novel and new. We will then discuss the fusion problem and give a solution, which is suitable for our case but not unique. Examples will be given throughout the analysis of the method.

2 Knowledge Representation of the User Model and of the Prototypes

It is important to recognize that we will only have sparse information about a particular user and while this will build up over time, we should not expect to be able to use learning methods that require large amounts of data to provide the user model. Essentially, this means that we need to find a way to make inference when we only have sparse data available.

In human terms we do this all the time. We meet someone and make decisions about that person with reference to a set of prototypical persons, which we have built over time. For example, if we knew someone that worked in a software house, we could assume with a certain degree of confidence that he would know how to program. We should collect data for building up clusters of types of users, according to their behaviour, their abilities, needs, etc and use these clusters as prototypes. The way in which the system acquires relevant information to build them is a separate problem. It can produce them by extracting information from a database of individuals, or the user can provide it with his understanding of prototypes or refine existing ones. For our purpose, we will assume that we have a collection of prototypical people.

As for the user's model, the easiest way would be to ask him directly to pick a model (categorise himself) from an existing model database. That would leave everything to the user. Another way would be to ask the user for relevant information, possibly giving him a set of choices for each characteristic we are interested in. Finally, the most advanced method and at the same time the least demanding from the user's point of view, would be creating the model by learning the necessary information.

The next question that we need to answer is how can the system acquire relevant information about the user. By observing the user's real-time interaction with the system, we can learn quite a lot about him, his preferences, his interests, his habits, etc. Additionally, all the systems that use user-profiling techniques, should give their user a certain degree of freedom as well as allow him to get involved with improving the agent's performance. Consequently, the user should be able to give feedback concerning the system's actions and should be able to improve his own profile at any time.

The representation of the clusters/ prototypes we mentioned earlier can be done in different ways. One approach would be to represent each person in the cluster as a vector point and to find the average vector for the cluster and this would represent our

prototype for the cluster. Having a collection of prototypes represented in this way, when presented with a particular user, we would match him/her to the nearest prototype vector. However, we consider this to be a rather simplistic form of representation because first of all it does not allow us to consider relationships between concepts, but only basic attributes. In addition, there is a lack of flexibility as a result of having just a single point representing a prototype. Consequently, we decided to use Conceptual Graphs that satisfy both of the above requirements. In this way, we will be able to represent a cluster as a set of attributes and their relations. Conceptual graphs are described in greater detail in a following section.

Let us give an example to illustrate the difference between the two above forms of representation and identify the advantages of the second one. Supposedly, we had to represent a cluster of almost perfect circles. One way to achieve this would be to take a prototypical circle with radius r and centre c and create the vector (r, c). We would then accept a circle as a member of this cluster, if its average radius and centre lie close to r and c. A second way of representing the same cluster would be by using fuzzy sets [11]. Specifically, by defining the radius as a fuzzy set f and the centre having co-ordinates (gx, gy), where gx and gy are two fuzzy sets. The fuzzy sets f, gx and gy will thus define a family of circles with varying membership. We could describe this family as a fuzzy circle and this will constitute our cluster of acceptable circles. To decide whether a new almost perfect circle belongs to this cluster, we would have to check whether it lies in the aforementioned fuzzy circle. Relations between attributes can be defined as well. In our example, having the attributes minimum diameter and maximum diameter of a circle, we can define the difference between the two, which will have a maximum magnitude for accepted circles. We can then say that for any almost perfect circle, this difference must be g where g is a fuzzy number. Conceptual graphs are suitable for efficiently describing attributes as fuzzy sets and relations between them.

3 The Philosophy Behind Our Approach

We will collect information on users based on their interaction with the computer and will divide them into clusters. The latter will be generic categories of people with similar behaviour and characteristics. Those characteristics will be captured in the cluster's definition. Each cluster will be defined by the relevant fuzzy set attributes and relationships between them and will be represented by a conceptual graph. This will form an individual prototype. A new person will be checked against this prototype to decide how closely it matches it. He/she will also be checked against all the other prototypes that include information that is of interest. By doing this we assume that a user that partially matches a certain prototypical person’s behaviour, will probably possess other features of that specific prototype as well. The computer will be able, when missing some information about its user, to deduce it from prototypical users with similar behaviour. This form of reasoning will be inductive or even analogical and will have no truth guarantee.

4 Conceptual Graphs

Conceptual graphs [15] are finite, connected, bipartite graphs with concept and relation nodes. Concept nodes have a label and a referent field, whereas relation nodes have only labels. The concept node's referent field is the concept's instantiation and can either be a value, a set of values or a fuzzy set [4]. The relation nodes' role is to connect the concept nodes and to represent the relationship between them. Conceptual graphs are related to semantic nets and overcome some of the difficulties such as the "isa" problem, which were found in early uses of the latter. Another major advantage that can be easily identified is that these graphs are closely related to natural language. Every English sentence can be represented with a conceptual graph and every graph corresponds to an English sentence. Here is an example graph that illustrates the aforementioned notions.

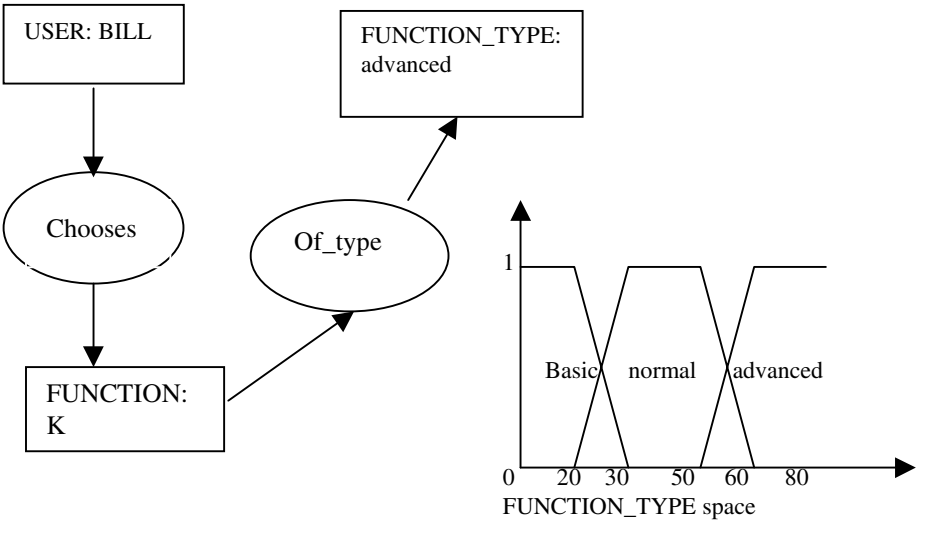


Fig. 1. Example conceptual graph

The graph above is a graph representing the sentence: " Bill chooses advanced functions". We can see that concepts USER and FUNCTION are instantiated to single values "BILL" and "K" respectively, whereas "FUNCTION_TYPE" to the fuzzy set "advanced", which is defined at the "Function type" space. The relation nodes "Chooses" and "Of_type" are there to connect the previously mentioned concepts.

5 Outline of the Inductive Inference Mechanism

In this section we will discuss the preliminary ideas for inference with respect to a certain user based on prototypes represented as conceptual graphs. At this point, we will not refer to the source of the information used. We will assume that after

studying people in a certain context, we derived a set of prototype conceptual graphs. People with similar behaviour or characteristics, depending on the context, were clustered to the same graph. Furthermore, the user's graph is developed after collecting relevant information about him. This information can be acquired in several ways: observation of the real-time interaction between him and the system, some direct user-feedback concerning the system's actions, or by other means which will not be dealt with in this report. In this report we will refer to the prototype graphs as P1, P2, … Pn and to the user's graph as S. Examples of these can be seen in figures 2 and 3:

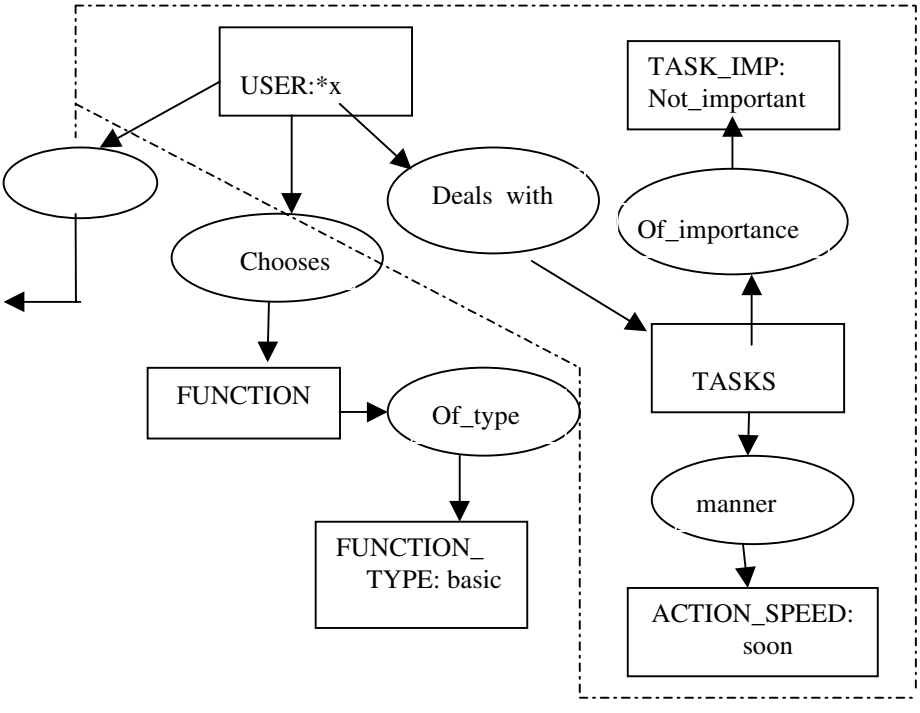


Fig. 2. Prototype graph P1 and P1' (within dotted line)

Prototype graphs P1, P2 will be representative of the prototype graph category. In our example we will only use two graphs for illustrative purposes. However, in reality there will be a significantly greater number.

As we mentioned above, the computer will not always have all the necessary information on a particular user in order to take action on his behalf. Its knowledge will be limited, so it should make inferences from the information it has available. If the user were one of the prototypes, then the answer would be given by part of the respective graph.

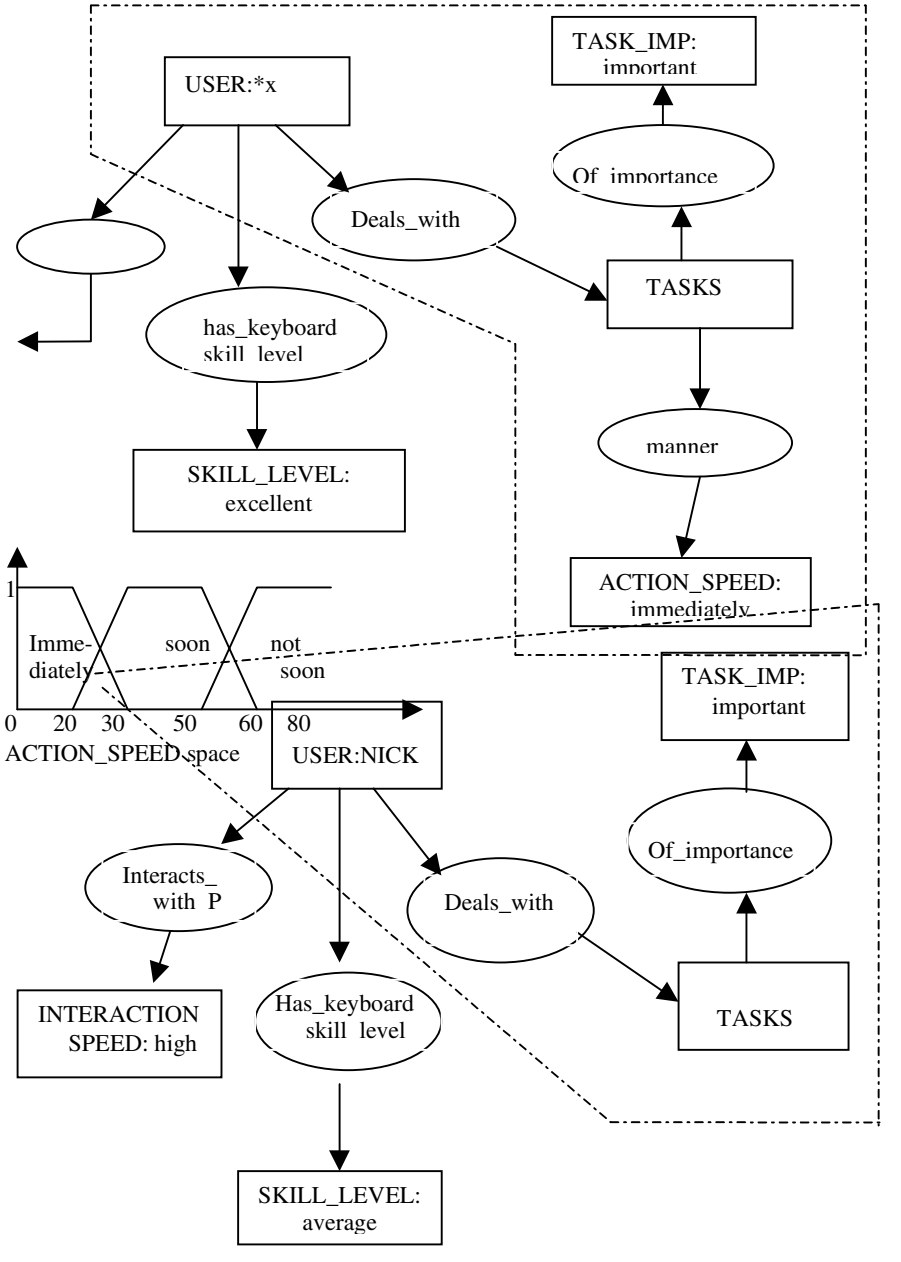


Fig. 3. Prototype Graph P2 and P2' (within dotted line, top) and User Graph S (bottom)

The information that the computer will be looking for, or in other words the information that will be missing from the user's graph, could be expressed in a query/question format when using natural language. For our example, let us suppose we would like to answer the question: "How quickly does the user act when dealing with important tasks?" Since every natural language sentence can be represented with a conceptual graph, we do this for our question and we end up with the graph Q in figure 4. To construct Q, we must use concept and relation node labels that exist in the prototypes as well, in order to achieve some matching. The information required will be concept nodes with empty referent fields (ACTION_SPEED in this case).

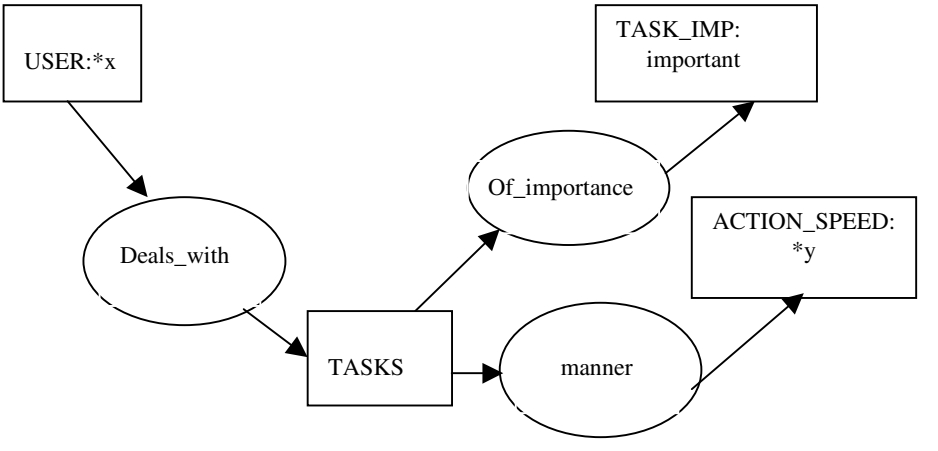


Fig. 4. Question/ Query Graph Q

We will match graph Q to each of P1, P2…Pn – this will correspond to a maximal join operation – all nodes in Q should match corresponding nodes in P1, P2…Pn. If nodes exist in Q that do not find matching nodes in a certain prototype graph, then that graph is not capable of providing us with the information we are looking for and so we do not consider it any further. After the maximal join, in the resulting graphs, the concepts with the empty referent fields will be instantiated. These graphs will look like the sub-graphs defined by the dashed line drawn on P1 and P2 with the appropriate instantiations. For each graph, we identify the sub-graphs that correspond to the answer and strip the remaining concepts and nodes. The new graphs obtained will be called P'1, P'2…P'n. At this point we need to mention that the stripping operation would be more efficient if we kept parts of the graph that might be relevant to our question. Looking at the main concept of the question graph and keeping sub-graphs that directly support it, can possibly identify these parts. This will further be analysed in following reports.

The graph S corresponding to the specific user is also treated in this way. Q is mapped onto S and S is stripped of non-relevant nodes to give S'. This will look like the sub-graph defined by the dashed line in S, with an additional ACTION_SPEED concept with empty referent field. S of course will not contain the sub-graph corresponding to the answer, otherwise we would not be referring to the prototype graphs to obtain this information.

We now take S' and match onto P'1, P'2…P'n by performing maximal join operations. This results to graphs P"1, P"2…P"n. These maximal joins will not be complete joins and some measure of completeness will be used to give a support for how well S' matches each of P'1, P'2… P'n. When we talk about matching two graphs, we mean matching one's relations and concepts against the other's. Two relations match when their type is the same. Two concepts match with a support s=1 when their types and referents are identical and with a support s<1 when their types are the same, but their referents different. Since we consider the case where referents can be fuzzy sets as well [4], in that specific case we perform Point Semantic Unification [1], which is based on the Mass Assignment Theory [5], to obtain a support. Because we get a support from each pair of concepts that is matched, we accept the overall support to be the conjunction of the individual supports. Consequently, we end up with a support for each pair S' and P'n. Let these supports be given by s1, s2…sn. In our example, the information we would get from P"1 is that "Nick deals with non-important tasks soon" with a support s1 and from P"2 that "Nick deals with important tasks immediately" with a support s2. We can now pick out the parts of P"1, P"2…P"n that correspond to our answer. Let these answer graphs be A1, A2…An. We do this by projecting P"1, P"2…P"n on to our query graph and identifying the part of the former that does not project on anything. In our example they will just consist of the ACTION_SPEED concept instantiated to one of the fuzzy sets of the respective space (Fig. 3).

At this stage we have a set of answer graphs A1, A2…An with a support for each one. It is now necessary to fuse these to obtain a final answer graph A. This will be done with a combing scheme, based on the Mass Assignment Theory [5], that will take into account the supports as well. Essentially, our answers will have the form of fuzzy sets with accompanying supports. If we wanted to combine for example f1 and f2 with supports s1 and s2 respectively:

$$ExpLeastpr\ ejudicedDi\ stribution\ (f1) = s1 * LPD(f1) + (1 - s1) * LPD(\overline{f1}) = f1'$$

$$ExpLeastprejudicedDistribution(f2) = s2 * LPD(f2) + (1 - s2) * LPD(\overline{f2}) = f2'$$

$$f_{final} = f1' \cap f2'$$

The answer graph A will contain the information required from the computer to act on the user's behalf. The feedback given by the user can be potentially used to adjust and improve the combining schemes used to match the user's preferred choice.

At this present time, a Conceptual Graph Toolkit software package has been implemented in FRIL [2, 3, 10], which allows graphs to be stored in linear format and us to perform several operations such as maximal join, simplification, restriction, etc. Furthermore, the whole inference mechanism described above has been developed.

6 Application - "The Forum"

Researchers at BT's Adastral Park have developed the "Forum" [11], which is an on-line collaborative virtual working environment [6] aiming to bring people together both informally and formally. It is designed to allow people who should meet each other do so easily and naturally and provides the means for them to have richer on-

line meetings. It is divided in two parts: the Contact Space and the Meeting Space. We will focus on the first. The Contact Space consists of a number of zones that represent subject interests. According to the user's "overall" interest at a particular moment, his avatar is placed in the appropriate zone along with other users' avatars. An avatar can chat to any other on the same zone.

Our aim in this project is not only to categorize a certain user based on his interaction with the computer and subsequently place his avatar on the relevant zone. That can be achieved in several different ways. We would propose one related to the mechanism described previously, however we will not get into it in detail in this paper. Our aim is to make the whole system act a bit more intelligently. A system is not necessarily intelligent simply because it knows e.g. all the symptoms of 1000 diseases. It must be able to make rational diagnoses based on the presence and absence of different combinations of them. In the Contact Space case, one possibility would be to need to make inference about a user not knowing everything about him, e.g. how much is he interested in interest X. Another would be to have to suggest to someone to go and "chat" with person Y because they have a common interest, apart from the "zone's" interest. We will now analyze some ideas on the way our novel inference method can be applied in the "Forum".

7 Applying Intelligent User Modelling to "the Forum"

Our target is to create prototype user models, which will constitute the different user categories; the user groups in other words. After “constructing” those models and a similar user’s model, we will be able to identify similarities between them, which will allow us to draw conclusions. “What should a prototype consist of”, is a question that can be answered in a lot of different ways depending on the information we have available and the scope of our system. It is very crucial to find the right combination of pieces information to form the prototype user models. Two requirements need to be satisfied in every case. To take advantage of the information gathered from different sources, so that important information is not left out, and at the same time not to include too much detail that would make the system too complex and less flexible.

Initially, we need to identify the different sources of information for a particular user. These can be the topics the user works on, the applications he/she uses, the topics he/she searches for on the Internet and his/her interest areas based on Jasper [7, 8]. This kind of information can be easily collected and summarized. It can be generalized in a way that nothing important is omitted and everything that is not of great interest is left out. In any case, the information used should be enough to distinguish one prototype from another.

Each prototype can represent a different zone in Contact Space. That means that we will have a “Software Engineering” prototype, a “Wearable Computing” prototype, a “Distributed Computing”, a “Shared Virtual Worlds”, etc. Each one of them will basically include the characteristics of a person that would definitely belong to that category. For example a “Shared Virtual Worlds” prototype user will be someone whose interests are “Virtual Reality”, “Animation”, “Computer Games”, who works on “VRML” and “C++”, who searches for “avatars” and “navigation in virtual space” on the Internet and who uses applications such as “The Forum” and “Visual C++”.

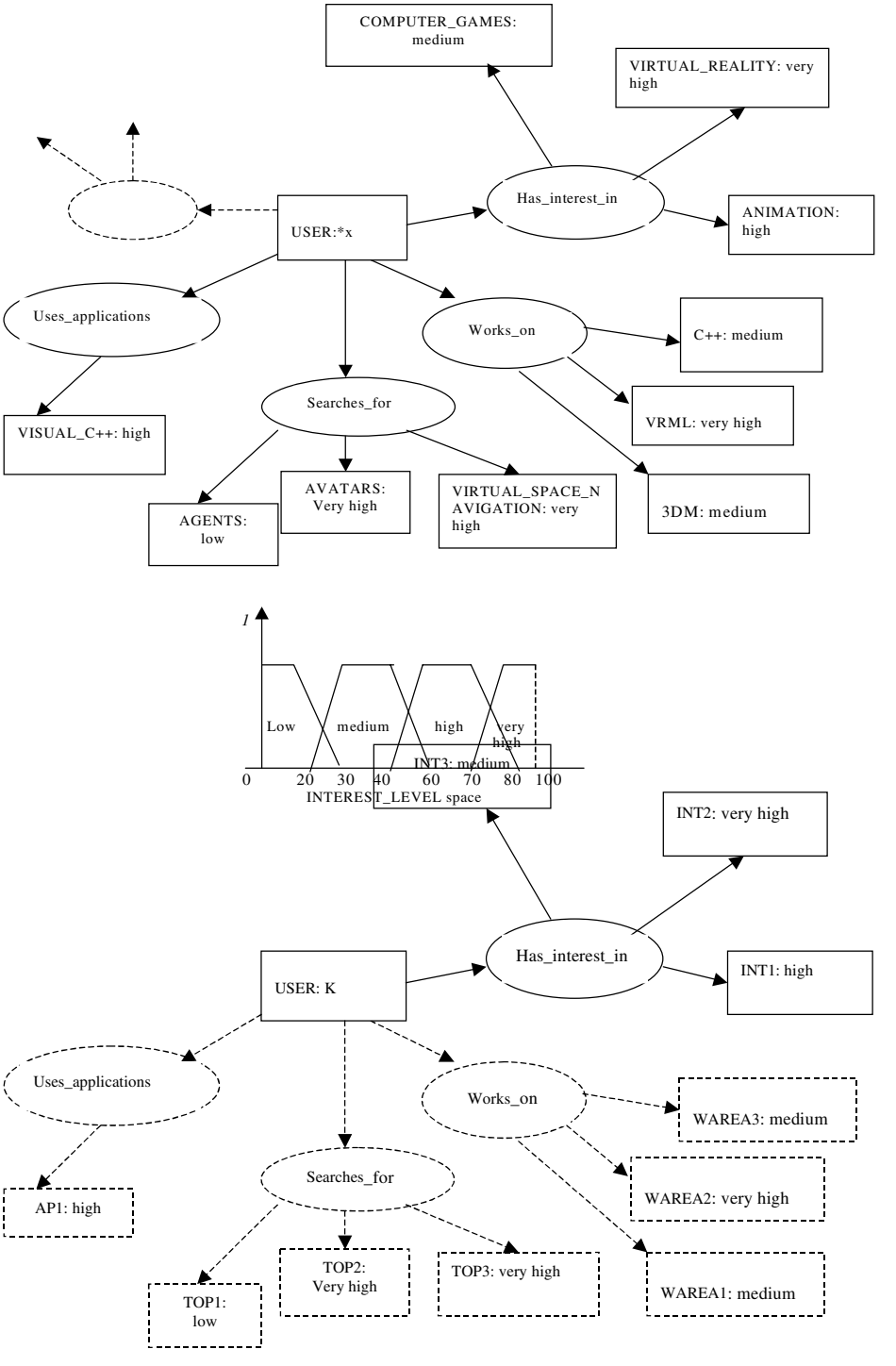


Fig. 5. Example Conceptual Graph (top) and Abstract User's Graph (bottom)

As we specified in the inference method description, we are going to use conceptual graphs as a method of knowledge representation. These graphs will have concepts instantiated to fuzzy sets in order to be able to obtain a support when performing the matching later on. In our case, the concepts nodes will be key phrases as the ones mentioned above, with instantiations to fuzzy sets that will show the level of interest to that particular topic. The relation nodes will represent the source of information. In figure 5 (top) we can see an example conceptual graph and the fuzzy sets defined in LEVEL_OF_INTEREST space.

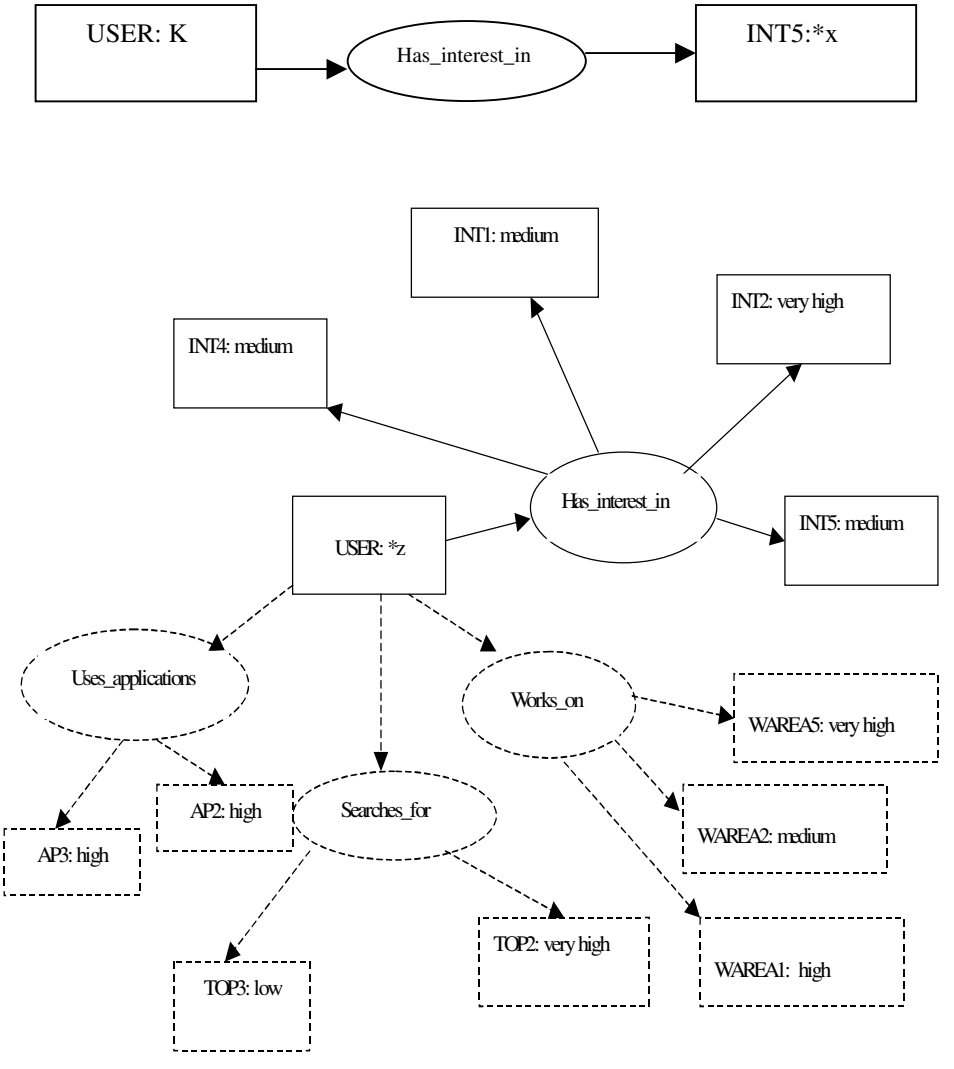


Fig. 6. Query Graph (top) and Prototype Graph 1 (bottom). Concept and relation nodes in dotted lines are stripped

A collection of graphs with a similar architecture to the one of graph in figure 6 would play the role of the different user groups. The user himself will have a graph that will encapsulate his characteristics. At that point, we can actually say that we have all the necessary information in an appropriate for manipulation format. We are now able to perform several operations.

One is to compare the user's graph to all the prototypes and find a degree of matching to each one. That will give us the ability to say in which category and thus in which zone the user belongs to, having his specific attributes in mind. The other probably more useful operation we can perform, as analyzed in the proposed method's description, is to infer things about the user that we don't know. Supposedly, we wanted to find out how interested is user K in interest topic 5 (INT5 in graph). We would hypothetically have graphs that would resemble the ones shown in figure 5 (bottom), 6 and 7.

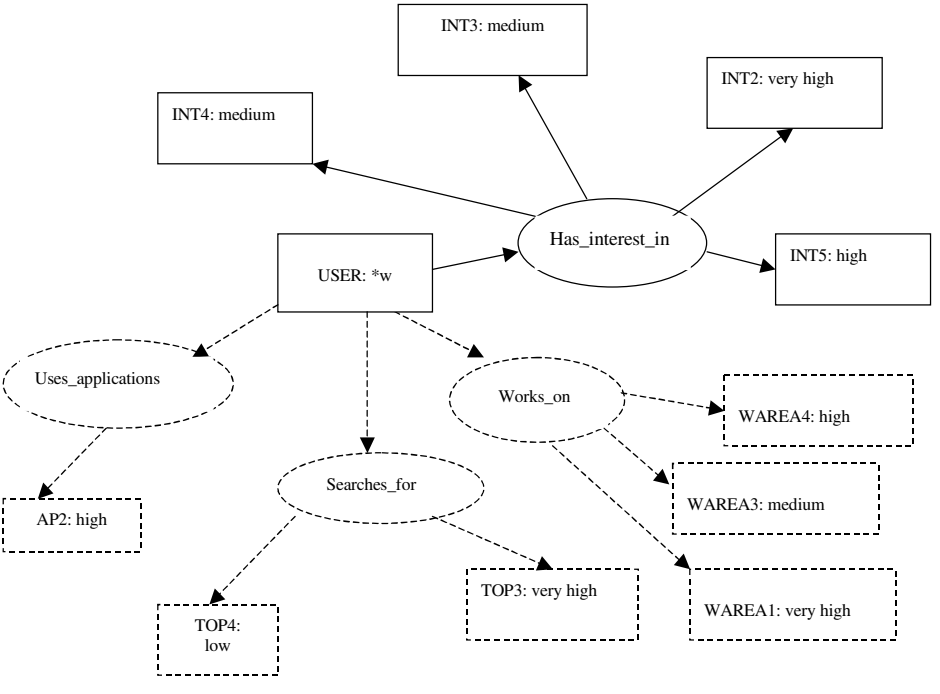


Fig. 7. Prototype Graph 2. Concept and Relation nodes in dotted lines are stripped

By following the procedure described previously, we "strip" both the user graph and the prototype graphs from any irrelevant sub-graphs having the query graph as our guide. The irrelevant parts can be seen in dotted lines. We then match the resulting user's stripped graph on to each stripped prototype graph. Obviously some concepts match and some do not. The ones that match will probably match with a certain degree/support, which will come from combining their fuzzy sets (point

semantic unification [1]). An overall support per prototype-user pair can be obtained by means of conjunction.

The answer, which in our case would be the instantiation of the INT5 concept, can be found by projecting the query graph on the prototypes. Consequently, we end up with a fuzzy set as an answer and a support associated with it, for every prototype we considered. We can combine these using a combination scheme based on Mass Assignment theory that combines probability distributions taking supports into account. In our example, we would get “medium” as an answer from prototype1 with a support S1 e.g. and “high” from prototype2 with a support S2. The combination of these two would result to a fuzzy set defined on INTEREST LEVEL space. Depending on what format we want our answer to have, we can get an exact value in that space, or we can get a linguistic output such as “fairly high” or “rather low”.

8 Summary

We have thoroughly described a novel method that gives us the ability to infer information we do not know about the user, based on the information we already have. The method uses Conceptual Graphs [15] as a form of knowledge representation and employs Fuzzy Set Theory [17] as well as the Mass Assignment Theory [5]. "The Forum" has been introduced as a system to which we can apply our mechanism to obtain advanced results.

References

1. Baldwin, J.F.: Probabilistic, Fuzzy and Evidential Reasoning in Fril. Proc. Two Decades of Fuzzy Control, IEE London (1993) 1-20
2. Baldwin, J.F., Martin, T.P., Pilsworth, B.W.: FRIL - Fuzzy and Evidential Reasoning in AI. Research Studies Press, John Wiley (1995)
3. Baldwin, J.F., Martin, T.P., Pilsworth, B.W.: FRIL Manual. Fril Systems Ltd, Bristol Business Centre, Maggs House, Queens Road, Bristol, BS8 1QX, UK (1988)
4. Baldwin, J.F., Morton, S.K.: Conceptual Graphs and Fuzzy Qualifiers in Natural Language Interfaces. Research Report ITRC-85, University of Bristol (1985)
5. Baldwin, J.F.: A Theory of Mass Assignments for Artificial Intelligence. In: Driankov, D., Eklund, P.W., Ralescu, A.L. (eds): Fuzzy Logic and Fuzzy Control. (1991)
6. Bradley, L., Walker, G., McGrath, A.: Shared Spaces. British Telecommunications Engineering Journal, Vol. 15 (1996)
7. Davies, N.J., Weeks, R., Revett, M.C.: An Information Agent for WWW. Proc. Int. Conf. On WorldWideWeb, Boston, USA (1995)
8. Davies, N.J., Weeks, R., Revett, M.C.: Jasper: Communicating Information Agents for WWW. Proc. Int. Conf. On WorldWideWeb, Boston, USA (1995)
9. Kobsa, A.: User Modeling: Recent work, prospects and hazards. In: Schneider-Hufschmidt, M., Kuhme, T., Malinowski, U. (eds): Adaptive User Interfaces: Principles and Practice. Elsevier Science Publishers B.V., Amsterdam (1993) 111-128
10. Martin, T.P., Arcelli Fontana, F.: Logic Programming and Soft Computing, Research Studies Press/ Wiley (1998)
11. McGrath, A.: The Forum. SIGGROUP Bulletin, Vol. 19, no. 3 (1998)

12. McTear, M.: User modelling for adaptive computer systems: A survey of recent developments. Artificial Intelligence Review, Vol. 7 (1993) 157-184
13. Nwana, H.S.: Software Agents: an overview. The Knowledge Engineering Review, Vol. 11, no. 3 (1996) 205-244
14. Rich, E.: User Modeling via Stereotypes. Cognitive Science, Vol. 3 (1979) 329-354
15. Sowa, J.F.: Conceptual Structures: Information Processing in Mind and Machine. Addison-Wesley, Massachusetts (1984)
16. Wooldridge, M., Jennings, N.: Intelligent Agents: Theory and Practice. The Knowledge Engineering Review, Vol. 10, no. 2 (1995) 115-152
17. Zadeh, L.A.: Fuzzy Sets. Information and Control, Vol. 8 (1965) 338-353

URL References

1. BT Labs - The Forum: http://www.bt.com/innovation/exhibition/forum/index.htm

Towards a Unified Querying System of Both Structured and Semi-structured Imprecise Data Using Fuzzy View

Patrice Buche and Ollivier Haemmerlé

INA-PG, Département OMIP, 16, rue Claude Bernard
F-75231 Paris Cedex 05, France
{Patrice.Buche,Ollivier.Haemmerle}@inapg.inra.fr

Abstract. This work is part of a national project which aims at building a tool for the analysis of microbial risks in food products. As a first step of this work, we propose a unified querying system which simultaneously scans two different bases: a relational database containing structured information and a conceptual graph knowledge base containing semi-structured information. These two bases contain microbiological information. To achieve this, we propose a way of translating a database query expressed in a speci fic language into a query represented by a conceptual graph. This graph is projected into the base. It can also be generalized in order to avoid silent answers.

1 Introduction

Our research work is part of a national project which brings together government institutions and industry in order to build a tool for the analysis of microbial risks in food products. The first step of this project consists in gathering in a "database" all the information available in the scientific bibliography in microbiology that can be useful for risk assessment. In this field of application, information can be either qualitative or quantitative. The information is often imprecise because of the com plexity of the biological processes involved.

But another problem is that this information consists of experimental results in a field where knowledge is growing everyday. The integration of this information is a source of irregularity: similar data are often represented in a different way in independent bibliographical references. The term "semi-structured" is used to qualify this kind of information which is not really structured but presents similarities even if it is implicit. With semi-structured information, it is very difficult to determine a classical database schema in order to store all the useful information. Different approaches have been proposed to solve this kind of problem: (i) the definition of a new kind of database management system especially designed for semi-structured data [1]; (ii) the definition of viewpoints in the object model [2]; (iii) hybrid approaches combining the use of languages designed for semi-structured information representation such as XML for instance and object-oriented DBMS [3] or semi-structured data DBMS [4].

B. Ganter and G.W. Mineau (Eds.): ICCS 2000, LNAI 1867, pp. 207–220, 2000.
© Springer-Verlag Berlin Heidelberg 2000

In this paper, we propose another approach which consists in designing a unified querying system (called UQS) that scans two separate bases simultaneously: (i) a relational database extended to the representation of fuzzy data containing the structured information, processed by the CFQ engine (for Contextual Fuzzy Querying), (ii) a conceptual graph knowledge base containing the semi-structured information, processed by the SSI engine (for Semi-Structured Information).

There were many reasons for us to choose the CG model in order to represent the semi-structured information for many reasons. First, its graph structure is well suited for the representation of weakly structured information. Second, the projection operation is available to perform database scanning. Third, the terminological knowledge can be useful to implement enlarged flexible querying. Fourth, different software platforms are available, allowing one to realize prototypes easily.

The relationships between Relational Databases and CGs have been studied in several works. They generally aim at representing Relational Databases in terms of CGs in order to express all the knowledge of a RDBMS (schema, data, query, view...) in a unified framework [5]. Such a modelization allows one to take advantage of the terminological knowledge of the CG model, in order to introduce flexibility in the process of querying as in [6]. Our approach is rather different: we don't aim at translating a formalism into another. On the contrary, we propose to use uniformly two bases of a different nature; the base using the CG model allows us to relax the mandatory constraint of schema pre-existence in the relational model.

In this paper, we study the architecture of the SSI engine. On the one hand, we will focus on the representation of semi-structured information and on the other hand, on the way of querying this information. We will assume that values used as selection criteria are crisp values (not fuzzy) and that the information stored in the CG knowledge base is precise; the "fuzzification" of this part of the unified querying system will be presented in a future paper.

In the second section, we define the UQS query language and give explanations about the CFQ engine. In the third section, we explain how our application is modelized in terms of a CG knowledge base. In the fourth and fifth sections, we define respectively the notion of couple attribute/value in terms of CGs and the notion of view used by the unified querying system in order to be able to scan the CG knowledge base. In the sixth section, we propose an algorithm allowing one to search for answers to a query o n the CG knowledge base.

2 The UQS User Interface and the CFQ Engine

2.1 UQS Querying Language

In UQS, the queries are expressed in terms of a set of DB-projection[1] attributes and a set of selection criteria using the form attribute/value. These queries are

[1] in order to prevent ambiguities, we use the term of DB-projection when dealing with the notion used in the relational database model

expressed in a given view. A view is a classical concept in databases, e.g. a virtual table in which all the information needed by the user is brought together.

Definition 1 *A query Q in UQS is a set* $\{V, a_1, \ldots, a_n, <a_{n+1}, v_{n+1}>, \ldots, <a_m, v_m>\}$ *where V is the name of the view in which the query is asked;* $a_1, \ldots, a_n$ *are the attributes of the DB-projection and* $<a_{n+1}, v_{n+1}>, \ldots, <a_m, v_m>$ *are the pairs attribute/value of selection criteria. Note that* $\{a_1, \ldots, a_n\} \cap \{a_{n+1}, \ldots, a_m\}$ *is not necessarily empty.*

The result of the execution of a query in UQS is a table which is composed of a number of columns equal to the number of DB-projection attributes. Each line represents a tuple composed of couples <attribute, value>, resulting from the query.

Definition 2 *An answer A to a query Q in UQS is a set of tuples, each of them of the form* $\{<a_1, v_1>, \ldots <a_n, v_n>\}$ *with* $a_1, \ldots, a_n$ *the attributes of the DB-projection and* $v_1, \ldots, v_n$ *the values resulting from the execution of the query associated with each attribute of the DB-projection.*

2.2 The CFQ Engine

An exhaustive presentation of the CFQ engine is given in [7]. Let us now present it briefly. For the relational database scanning, we have introduced the new concept of contextual fuzzy view. This notion is an extension of the classical concept of view, which is fuzzy because the selection criteria are expressed as fuzzy predicates as in [8]. This view is also contextual because it associates the user's preferences expressed in the query with a category of queried information (defined i n the knowledge base of the information retrieval system). The contextual fuzzy view has three advantages. First, the CFQ engine enlarges queries when they are too restrictive because of the actual content of the database: it provides an additional degree of flexibility not available in the previous systems. Second, the view engine makes an estimation of the data sought, in addition to the nearest information retrieved from the database. For this purpose, statistical models have to be incorporated into the categories of queried information. Third, the view engine optimizes the fuzzy matching processing because the user's preferences are only compared with the information associated with the category selected: in the previous systems, all the information stored in the queried tables was compared with the user's preferences. The concept of contextual fuzzy view is implemented in a prototype written in Java language called CFQ (for Contextual Fuzzy Querying). The information is stored in an Oracle DBMS using the imprecision representation model of FSQL [8].

3 Representation of the Semi-structured Data from Our Knowledge Base in Terms of Conceptual Graphs

The underlying application concerns information retrieved from scientific bibliography in microbiology for risk assessment. More precisely, it is information

describing the behaviour of pathogen germs (increase, decrease or stability of pathogen concentration such as Listeria Monocytogenes for instance) in food products during food transformation processes (heating, cutting, storage, mixing, etc.). In this paper, we take the example of information about the behaviour of Listeria in food when it interacts wi th another family of bactery, called Bacteriocin, which can be inoculated in food products in order to inactivate Listeria (that means to obtain a decrease in the concentration of Listeria).

We chose the CG model [9] as a formalism to represent semi-structured data. More precisely, we follow the formalization introduced in [10]. The principal differences with the model introduced by Sowa are that the partially ordered concept type set is not necessarily a lattice, and that CGs can be non-connected graphs (the projection of G into H is then defined in terms of the projection of each connected component of G into H).

3.1 The Terminological Knowledge (The Support)

In the modelization of the support, we represent the main part of the application semantics in the set of concept types. A weaker part of the application semantics is represented in the set of relation types. Relation types must act as "as generic as possible" connectors in order to be as stable as possible. The concept type set thus contains: biological information organized in taxonomy (substrate, germs), different types of data retrieved from scientific papers (bibliographical data, experimental data, model parameters, units of measure,...) and the different actions described in the papers (heating, germs interaction,...) as well as their results in terms of the behaviour of the germs (increase, decrease or stability of pathogen). Part of the concept type set is given in fig. 1.

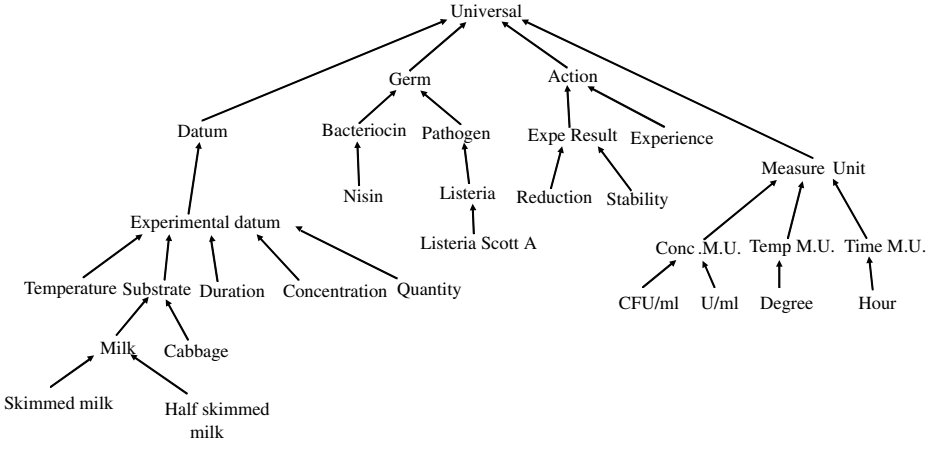


Fig. 1. A part of the concept type set

Remark 1 *It is important to note that all the attributes of the DB tables that can be queried on in UQS have to be inserted in the concept type set.*

The set of relation types contains generic relation such as *Agt(Action, Germ)* (has for agent), *Char(Universal, Experimental Datum)* (is characterized by), *Obj(Action, Universal)* (has for object), *Res(Action, Action)* (has for result), *Unit(Datum, Measure Unit)* (has for unit). Some relation types may be sub-typed in order to obtain a stronger type control. For example, the relation type *Unit* is sub-typed in *Temperature Unit*, *Time Unit* and *Concentration Unit* .

The set of individual markers is used as is usual in the CG model to represent the instances of concept types. Even if our goal is to represent complex numeric values (such as fuzzy intervals for instance), the only markers we use are symbolic ones (individuals). In this first step of our work, the matching of numeric values is limited to exact matching. In other words, a temperature of 20 is represented by the concept vertex [Temperature : twenty], "twenty" being an individual marker (in our examples, fo r a better readability, we use the numerical notation). The extension to numerical fuzzy values will be studied in a future work.

Remark 2 *We consider that the intersection of the set of the concept type labels and the set of the individual marker labels is empty. If* Nisin *is a concept type, sub-type of the concept type* Bacteriocin*, we don't accept an individual marker labeled "Nisin".*

3.2 Assertional Knowledge (The Graphs)

In our knowledge base, each graph represents an elementary piece of information. For instance, all the information relative to a biological experience (conditions of the experience such as temperature, pH, nature of the substrate, and results of the experience...) belongs to the same graph. Of course, a scientific publication can be expressed by several graphs. In fig. 2 and 3, we give the two graphs which correspond to the two experiences presented in the following biological resu lt taken from [11]: *"at a temperature of 37 °C, nisin at a concentration of 50 U/ml is very efficient (reduction after 2 hours) against Listeria Scott A (initial concentration of 6.10^8 CFU/ml) in skimmed milk and very inefficient (stability) in half-skimmed milk".*

We chose to represent the graphs in our knowledge base with connected, possibly cyclic CGs.

Definition 3 *The knowledge base $KB = \{G_1, \ldots, G_p\}$ containing the semi-structured knowledge of our system is a set of connected, possibly cyclic CGs.*

4 An Analogy of the Notion of Attribute/Value in Terms of Conceptual Graphs

The notion of pair attribute/value is used in two different steps of the search for answers in a knowledge base composed of CGs:

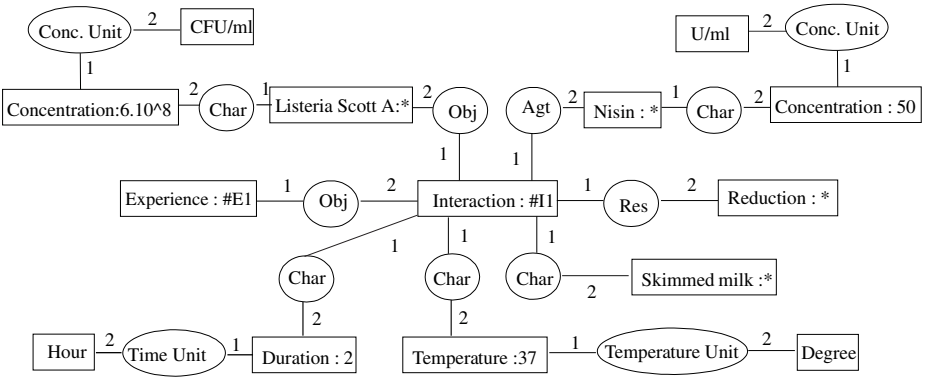


Fig. 2. An example of semi-structured information (experience 1)

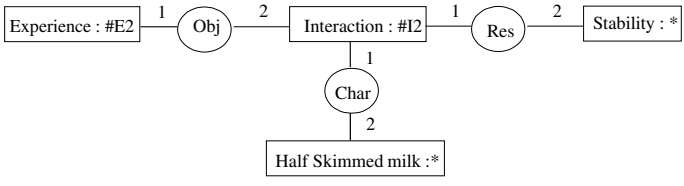


Fig. 3. A second example of semi-structured information (experience 2, only subpart different from experience 1 is shown)

1. during the selection operation, since we have to select information in which an attribute has a given value;
2. during the DB-projection, since we have to associate a value with each attribute of the DB-projection to build a tuple of the answer.

4.1 Simulation of a Selection in Terms of Conceptual Graphs

Assume we want to select a set of CGs from KB which satisfy a single $<a, v>$ pair of selection criteria.

Definition 4 *The concept vertex of selection associated with a pair of selection criteria* $<a, v>$*, noted sel-vertex*$(<a, v>)$ *is a single concept vertex labeled:*

- *[v : *] if v is a sub-type of a in the concept type set;*
- *[a : v] if v is not a sub-type of a in the concept type set.*

For example, if the pair of selection criteria is <Bacteriocin, Nisin>, the concept vertex of selection is [Nisin : *] because Nisin is a sub-type of Bacteriocin. If the pair of selection criteria is <Temperature, 20>, the concept vertex of selection is [Temperature : 20] because 20 is not a sub-type of Temperature.

Definition 5 *A CG G satisfies a selection criteria* $<a, v>$ *iff the graph limited to the single concept vertex sel-vertex*$(<a, v>)$ *can be projected into G.*

4.2 Simulation of a DB-Projection in Terms of Conceptual Graphs

Assume we want to compute a DB-projection in a CG, with an attribute of projection a.

Definition 6 *The concept vertex of DB-projection associated with an attribute of DB-projection a, noted proj-vertex(a), is a single concept vertex labeled [a : *].*

Definition 7 *The result of the DB-projection of an attribute a on a graph G of KB in which proj-vertex(a) can be projected, is the pair <a, v> built as follows. Let c be the concept vertex image of proj-vertex(a) through the projection:*

- *v is the referent of c if c is an individual concept vertex and type(c) = a;*
- *v is the type of c if c is a generic concept with type(c) < a;*
- *v is the pair <type(c), ref(c)> if c is an individual concept vertex and type(c) < a.*

For example, if the DB-projection attribute is pH and a CG G in the knowledge base is [pH : 7], we build the concept vertex of DB-projection [pH : *] which can be projected into G. The result of the DB-projection is <pH, 7>. If the DB-projection attribute is Bacteriocin and a CG G' in the knowledge base is [Nisin : *], we build the concept vertex of DB-projection [Bacteriocin : *] which can be projected into G'. The result of the DB-projection is <Bacteriocin, Nisin>. If the DB-projection at tribute is Bacteriocin and a CG G'' in the knowledge base is [Nisin : #212], we build the concept vertex of DB-projection [Bacteriocin : *] which can be projected into G''. The result of the DB-projection is <Bacteriocin, Nisin(#212)>.

Remark 3 *In the last example, a pair of values is returned for a single attribute. Even if such a result is not atomic, in contradiction with the usual DB-projection rule, we chose this solution because both values can bring meaningful information to the user. It is not a major drawback because we consider the DB-projection operation as a final one.*

5 An Analogy of the Notion of View in Terms of Conceptual Graphs

In section 2.1, we saw that the query in UQS is expressed in a view in terms of a set of projection attributes and a set of pairs of selection criteria <*attribute*, *value*>. We have seen that the notion of view is one of the central notions of our system. In terms of relational database, a view is a virtual table built by means of a query. This query contains attributes used in a "selection" purpose (the attributes appearing in the "where" clause) and attributes used in a "DB-p rojection" purpose (the attributes appearing in the "select" clause). All these attributes can be taken from different tables of the database. It is not easy to express this notion of view in terms of CGs. What we need is a process allowing us to build a "virtual table" by querying the graphs of the base.

Naïve approach. In order to extract information from the CGs, we need to find CGs which satisfy all the selection criteria, then get the values associated with all the DB-projection attributes.

Let $Q = \{V, a_1, \ldots, a_n, <a_{n+1}, v_{n+1}>, \ldots, <a_m, v_m>\}$ be the query we want to answer on KB.

A naïve method of finding such CGs would consist in filtering the knowledge base in three steps.

1. Building a disconnected CG query composed of:
   - m single concept vertices corresponding to $proj\text{-}vertex(a_i)$ for each projection attribute a_i of Q;
   - $m - n$ single concept vertices corresponding to $sel\text{-}vertex(<a_j, v_j>)$ for each selection criterium $<a_j, v_j>$ of Q.
2. Projecting the query into each CG of KB. We obtain a set of graphs which satisfy the selection criteria, and which contain each attribute of the DB-projection.
3. Building, for each of these graphs, a tuple $\{<a_1, v_1>, \ldots, <a_n, v_n>\}$ of the answer by extracting the result of the DB-projection of each projection attribute as presented in definition 7.

A drawback to this simple mechanism is its important lack of semantics. Whereas in the relational database view, the virtual table contains meaningful tuples (because the manager of the database implements pertinent queries), with the naïve method, we might get absurd results. For example, if we search for the temperature and the pH of an experience, and if a CG contains information about such an experience but also contains a reference to the external temperature, the projection of two single concept v ces will return the external temperature and the pH of the experience as a result, which is nonsense. In order to avoid such a "noise" in the answer, we propose to use "schemata" which are specific graphs containing all the concepts that belong to the considered view. These schemata link the attributes with semantics. For example, if we know that the temperature and the pH are relative to the condition of the same biological experience, we can have a schema which is a CG containing two generic concept v ertices [Temperature : *] and [pH : *], linked to another concept [Experience : *] with appropriate relation vertices. Then, if we find a projection of such a schema graph, we know that the values of the attributes *temperature* and *pH* belong to the same experience... Note that we assume that all the schema graphs for each view of the system are given by the manager of the knowledge base. This is a good way to ensure that all the schemata are meaningful and then that the results obtained by such a method are valid. So far, we have not worked on the automatic extraction of schemata, which could be done by searching for a common generalization of the graphs of the knowledge base for example. Moreover, we cannot use the schemata stored in the relational database as in [6] because (i) the information stored in both databases is different and (ii) we do not want to impose a unique schema preexistence for the semi-structured information.

Definition 8 *Let V be a view, let $a_1, \ldots, a_m$ be the m attributes belonging to the DB-view (in both the select and the where clauses). We will say that $S(V) = \{S_1, \ldots, S_r\}$ is the set of CGs schemata associated with the view V if each CG S_i of $S(V)$:*

- *is an acyclic and connected CG;*
- *contains exactly one generic concept vertex of the concept type (or a subtype) corresponding to the attribute a_i in the concept type set for each attribute a_i belonging to the view;*
- *links the attributes of the view with meaning in the considered view (this last point obviously depends on a human expert).*

Definition 9 *In a schema graph S_i, the unique concept vertex corresponding to the attribute a_i of the view is noted vertex(a_i). Its concept type is a_i or one of its subtypes (as we saw previously, each attribute belonging to a view has to be inserted in the concept type set).*

In this first version of our work, we voluntarily limit the schemata to connected and acyclic CGs - connected because it seems natural that all the attributes belonging to a same view should be linked, and acyclic for complexity reasons: as we will see in the next section, our searching algorithm is based on the projection operation, which is NP-complete, but polynomial in some particular cases such as that of a tree projected into a graph [12].

For example, we define the view Bacteriocin Interaction with the following attributes: Substrate, Bacteriocin, Duration, Temperature, Pathogen Germ, Expe. Result. The schema presented in fig. 4 is associated with this view.

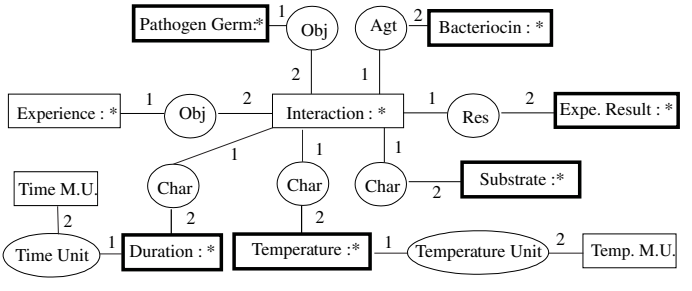


Fig. 4. A schema associated with the view Bacteriocin Interaction (concept vertices belonging to the view are framed in bold)

6 Querying the Knowledge Base

Let $Q = \{V, a_1, \ldots, a_n, <a_{n+1}, v_{n+1}>, \ldots, <a_m, v_m>\}$ be a query, and $S(V) = \{S_1, \ldots, S_r\}$ be the set of schema graphs associated with view V. The first version of the querying algorithm is the following:

$answer \leftarrow \emptyset$
for each $S \in S(V)$ do
  $query \leftarrow S$
  for each $vertex(a_i) \in$ set of vertices $vertex(a_{n+1}), \ldots, vertex(a_m)$ do
    restrict the label of the generic concept $[a_i$: *] of query to
      $sel\text{-}vertex(<a_i, v_i>)$
  done
  project $query$ graph into each graph of KB
  for each projection Π do
    build the tuple $\{<a_1, v_1>, \ldots, <a_n, v_n>\}$ from the result of the
      DB-projection of the vertices $vertex(a_1), \ldots, vertex(a_n)$
    add this tuple to $answer$
  done
done

Since the query graphs are acyclic, the projection algorithm we use is the polynomial algorithm for the projection of a tree into a graph proposed in [12]. For example, we consider the following query:

$Q = \{View = BacteriocinInteraction, Substrate, Bacteriocin, Duration, Temperature, PathogenGerm, ExperimentalResult, <Bacteriocin, Nisin>, <Duration, 2>, <Temperature, 37>, <PathogenGerm, Listeria>\}$

The query graph projected in the KB is given in fig. 5. The tuples returned by the algorithm (after projection on both examples shown in fig. 2 and 3) are the following:

Substrate	Bacteriocin	Duration	Temperature	Pathogen Germ	Experimental Result
Skimmed milk	Nisin	2	37	Listeria Scott A	Reduction
Half-skimmed milk	Nisin	2	37	Listeria Scott A	Stability

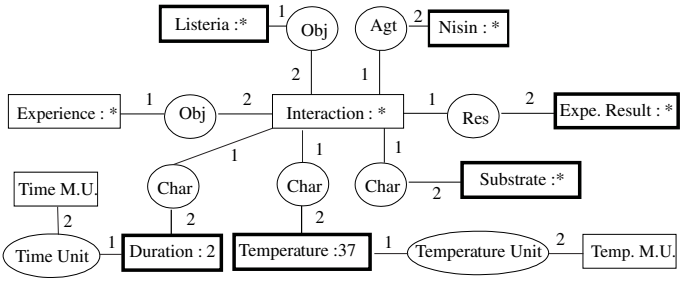


Fig. 5. An example of query graph projected in the knowledge base

This first version allows us to build an answer to the query as a virtual table based on the projection of partially instantiated schemata into the graphs of the knowledge base. But with this version, we obviously limit the result to an exact projection of the different schema graphs. Since the structure of the graphs of $S(V)$ and the structure of the graphs of KB are not necessarily the same, it is possible to get "silence" in the answer (the information allowing an answer is present in KB but the syste m cannot find it). For example, if the base contains the information stored in the CG shown in fig. 6 and if the query is that given in fig. 5, the projection is impossible.

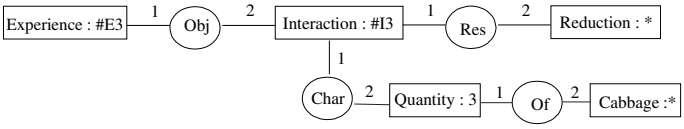


Fig. 6. A third example of semi-structured information (experience 3, only subpart different from experience 1 is shown)

This is a well-known problem with the search for projections of a query into a knowledge base, which is a Boolean operation (a CG can be projected into another one, or cannot). To solve this problem, it is possible to use a specific projection operation (such as α-β projection in [13] or "partial projection" in [14]). Another possibility is the generalization of the query graph as in [15].

In order to reduce the risk of "silence" in the answer, we propose to generalize the query graph by splitting it and by removing some parts of the graph, when no answer is found.

Generalization of a query graph In the following, we describe intuitively the main steps of the generalization process of a query graph *query*. During the first step, if the DB-view contains n attributes, we build at most n generalized query graphs for each schema graph. For example if the n attributes of the view are $a_1, a_2, \ldots, a_n$, we build n query graphs $query_1, query_2, \ldots, query_n$ from the query graph *query*. $query_i$ is a disconnected graph built by removing the vertices located on branches starting with $vertex(a_i)$ until the last relation vertex incident to *shortest-path*(a_i) or until the end of the branch if the branch is not incident to *shortest-path*(a_i). We define *shortest-path*(a_i) as the shortest path between all the attributes of *query* excluding a_i. $vertex(a_i)$ is not removed: the graph becomes non-connected. Note that if $vertex(a_i)$ is located on *shortest-path*(a_i), $query_i$ is not built (since it is isomorphic to *query*).

During the second step, all the generalized queries $query_1, query_2, \ldots, query_n$ are generalized again, following the same process, and so on, until the generalized query results in a disconnected graph composed of single generic concept vertices. This final generalization corresponds to the naïve approach presented in section 5. Of course, the higher the generalization of the query, the lower the level of confidence.

Note that 2^n is the upper bound of the number of generalized graphs if the view contains n distinct attributes. It is of course a major drawback of the mechanism used to generalize queries, but in actual practice, we will reduce the combinatorial explosion in limiting the generalization process to two disconnected vertices, so at most $n.(n-1)/2$ generalized queries are generated. Moreover, it is possible to store the generalizations of all the schema graphs, in order to avoid computing these generaliz ations every time.

In order to take into account the confidence level in the answers, we introduce an extension of the notion of answer.

Definition 10 *An answer A to a query Q is a set of tuples, each of them of the form* $\{<a_1, v_1, b_1>, \ldots, <a_n, v_n, b_n>\}$ *with* $a_1, \ldots, a_n$ *the attributes of the DB-projection,* $v_1, \ldots, v_n$ *the values resulting from the execution of the query associated with each attribute of the DB-projection and* $b_1, \ldots, b_n$ *Boolean values,* b_i *being set to* $false$ *if* $vertex(a_i)$ *is the image of an isolated vertex of one of the generalized query graphs.*

For example, if we consider the query given in fig. 5, the generalized query graph obtained when the vertex Substrate has been isolated is given in fig. 7. This generalized graph can be projected into the CG of fig. 6. The final result delivered by this method for the query of fig. 5 is given into the following table:

Substrate	Bacteriocin	Duration	Temp.	Pathogen Germ	Experimental Result
Skimmed milk	Nisin	2	37	Listeria Scott A	Reduction
Half-skimmed milk	Nisin	2	37	Listeria Scott A	Stability
Cabbage	Nisin	2	37	Listeria Scott A	Reduction

Remark 4 *The bold font used for the value* **cabbage** *means that this value has been obtained by the projection of an isolated vertex. The reliability of this value is not ensured.*

7 Conclusion and Perspectives

In this article, we presented in this paper the very first step of an important project concerning the storage and retrieval of semi-structured information. The underlying application concerns the domain of microbial risks in food products. In this first paper, we focused on the unified interrogation of a relational database and a knowledge base represented in terms of CGs. We implemented the part of this work concerning the CGs as a prototype built on the CoGITo platform [16].

Our very next work shall focus on three different points:

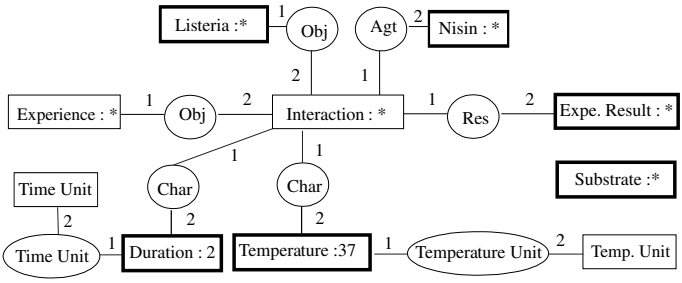


Fig. 7. The generalized graph from the query graph of fig. 6 obtained after isolation of the vertex Substrate.

- the extension of the CG model we use: due to the nature of the information we have to represent, it is important to extend the referent we use to numeric values, sets, fuzzy sets... Previous works have already studied the extension of the CG model to represent fuzzy information [17,18];
- the extension of the querying language of both the database and the knowledge base;
- the testing of our prototype on an entire knowledge base, which has to be created by the group of microbiologist experts working on our national project.

In a more distant future, we will have to think about our system being used by non-specialists of the CG model. An important work on the interfacing of our system has to be done. First, during the knowledge acquisition stage: for example, it is important to enable biologists to build the knowledge base with CGs which have to be completed. Second, during the querying stage, by displaying additional information extracted from CGs when the answer is built by using information found in the knowledge base.

References

1. J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom. Lore: A database management system for semistructured data. *SIGMOD Record*, 26(3):54–66, September 1997.
2. S. Coulondre and T. Libourel. Viewpoints handling in a object model with criterium-based classes. In *Proceedings DEXA'99 (Database and EXpert system Application), Lecture Notes in Computer Science #1677*, pages 573–583, Florence, Italy, August 1999. Springer-Verlag.
3. A. Michard. *XML, langage et applications*, pages 335–344. Eyrolles, Paris, 1999.
4. R. Goldman, J. McHugh, and J. Widom. From semistructured data to XML: Migrating the lore data model and query language. In *Proceedings of the 2nd International Workshop on the Web and Databases (WebDB '99)*, Philadelphia, USA, June 1999. Springer.
5. C. Boksenbaum, B. Carbonneill, O. Haemmerlé, and T. Libourel. Conceptual graphs for relational databases. In *Proceedings of the 1st International Conference*

on Conceptual Structures, Lecture Notes in Artificial Intelligence #699, pages 142–161, Quebec City, Canada, August 1993. Springer-Verlag.

6. B. Carbonneill and O. Haemmerlé. Standardizing and interfacing relational databases using conceptual graphs. In *Proceedings of the 2nd International Conference on Conceptual Structures, Lecture Notes in Artificial Intelligence #835*, College Park, MD, USA, August 1994. Springer-Verlag.
7. P. Buche and S. Loiseau. Using contextual fuzzy views to query imprecise data. In *Proceedings DEXA'99 (Database and EXpert system Application), Lecture Notes in Computer Science #1677*, pages 460–472, Florence, Italy, August 1999. Springer-Verlag.
8. J. Galindo, J.C. Cubero, O. Pons, and J.M. Medina. A server for fuzzy SQL queries. In *Proceedings of the workshop FQAS'98 (Flexible Query-Answering Systems)*, pages 161–171, Roskilde, Denmark, May 1998.
9. J.F. Sowa. *Conceptual structures - Information processing in Mind and Machine.* Addison-Welsey, 1984.
10. M.L. Mugnier and M. Chein. Représenter des connaissances et raisonner avec des graphes. *Revue d'Intelligence Artificielle*, 10(1):7–56, 1996.
11. D.S. Jung, F.W. Bodyfelt, and M.A. Daeschel. Influence of fat and emulsifiers on the efficacy of nisin in inhibiting listeria monocytogenes in fluid milk. *Journal of Dairy Science*, (75):387–393, 1992.
12. M.L. Mugnier and M. Chein. Polynomial algorithms for projection and matching. In *Proceedings of the 7th annual Workshop on Conceptual Graphs, Lecture Notes in Artificial Intelligence #754*, pages 239–251, Las Cruces, NM, USA, July 1992. Springer-Verlag.
13. D. Genest and M. Chein. An experiment in document retrieval using conceptual graphs. In *Proceedings of the 5th International Conference on Conceptual Structures, ICCS'97, Lecture Notes in Artificial Intelligence # 1257*, pages 489–504, Seattle, USA, August 1997. Springer.
14. B. Carbonneill and O. Haemmerlé. ROCK : un système de question/réponse fondé sur le formalisme des graphes conceptuels. In *Actes du 9ème Congrès Reconnaissances des Formes et Intelligence Artificielle*, pages 159–169, Paris, France, Janvier 1994.
15. J. Fargues. CG information retrieval using linear resolution, generalization and graph splitting. In *Proceedings of the Fourth annual workshop on conceptual graphs*, Detroit, USA, Janvier 1989.
16. O. Haemmerlé and O. Guinaldo. CoGITo v3.3 : plate-forme de d veloppement d'applications sur les graphes conceptuels. *Technique et Science Informatiques*, 18(9):933–965, Novembre 1999.
17. V. Wuwongse and T.H. Cao. Fuzzy conceptual graphs. In *Proceedings of the1st International Conference on Conceptual Structures, ICCS'93, Lecture Notes in Artificial Intelligence #699*, pages 430–449, Quebec City, Canada, August 1993. Springer-Verlag.
18. T. H. Cao and P. Creasy. Fuzzy order-sorted logic programming in conceptual graphs with a sound and complete proof procedure. In *Proceedings of the 6th International Conference on Conceptual Structures, ICCS'98, Lecture Notes in Artificial Intelligence # 1453*, pages 270–284, Montpellier, France, August 1998. Springer.

The Extensional Semantics of the Conceptual Graph Formalism

Guy W. Mineau

Dept. of Computer Science
Université Laval, Quebec City, Quebec
Canada, G1K 7P4
mineau@ift.ulaval.ca

Abstract. This paper gives the extensional semantics definitions associated with simple conceptual graphs[1] (CG) under a simple paradigm closely related to relational database (RDB) theory. We believe that by providing such definitions, the implementation of a CG system using this paradigm would be greatly facilitated. Considering the nature of CGs, existentially quantified assertions, the use of a RDB at the implementation level becomes attractive for applications with huge volume of data (like with deductive databases). Furthermore, RDB theory provides years of experience, stability and on-going support of the many tools that the industry offers today. These advantages become prerequisite when choosing a technological platform for a large scale industrial project. This paper provides a formal basis for understanding the deep semantics of the CG notation, including the compositional semantics of the canonical formation operators. It also provides a simple procedure that performs model checking of graphs in a CG system.

1 Introduction

A conceptual graph is mainly composed of existentially quantified concepts representing objects of some type. When translating a conceptual graph into a clause form as needed by a theorem prover, the skolemization of these variables introduces constants that represent these objects. This step, needed by a theorem prover whose unification procedure is based on universally quantified variables, prevents any reasoning on the possible cross-referencing of these variables. For applications where these cross-references must be established, for instance when information is exchanged between different knowledge bases or agent systems, the underlying semantics of these systems should not use skolemization to introduce constants. Rather, we advocate the use of *surrogates* (also called *witnesses* or *place-holders* in different theories). A surrogate is a variable used as a constant by the unification procedure, but which represents some individual that may not be known at the time the inference engine

[1] Simple conceptual graphs are those that do not bare nested graphs and do not use the negation operator.

B. Ganter and G.W. Mineau (Eds.): ICCS 2000, LNAI 1867, pp. 221-234, 2000.
© Springer-Verlag Berlin Heidelberg 2000

requires it. However, knowing that the individual object exists may be sufficient to provide interpretations over the formulae that use it, independently of its actual value (which can be determined at a later stage). Consequently, it provides for more flexibility when knowledge acquisition is done incrementally, as is the case in practical applications; the truth-value of a formula can be computed in spite of this lack of information.

In the CG literature, as surprising as it may seem, there are only a few attempts to define the extensional semantics of the CG formalism. Among them, let us cite the two most relevant work to this paper [1,2]. In [1], a brief correspondence between what is normally found in first order logic theory and the CG theory is sketched. Based on this correspondence, this paper provides the full definitions required to define the extensional semantics of the CG theory. In order to do so, necessary extensions proposed in this paper include: 1) a redefinition of the δ operator which now defines the extension set of a conceptual graph and is not limited to a type, 2) the introduction of variables into the definitions so that interpretations of formulae (conceptual graphs) could be defined over sets of existentially quantified concepts, and 3) the definition of the compositional semantics of the canonical formation operators.

In [2], the author presents the more thorough description of the CG extensional semantics of the existing literature. She proposes a general framework that links a grammar to its interpretation onto a set of individual objects. This is certainly a very general theory applicable in many cases, including the conceptual graph theory. What we propose in this paper could therefore be viewed as an instantiation of what is proposed in [2] for a special case where the grammar is known described in [3], and where the compositional semantics of the formation operators is given according to a RDB paradigm. Again, the choice of this paradigm as basis for providing these definitions aim at facilitating the mapping between theory and tools, which reduce the time to market constraint imposed on real-life industrial problems, and meets the usual requirements of software development in industrial settings: available expertise, stability of tool, support during development, installation and maintenance cycles, sustainability, efficiency, etc.

Section 2 provides the definitions of a canon of a CG system.[2] Section 3 describes the extensional semantics of the notation, including the compositional semantics of the basic canonical formation operators. Section 4 discusses how this framework can be extended to include semantic constraints. Section 5 concludes.

2 The Canon of a CG System

All graphs in a CG-based system are derived from the canon of the system. A canon contains the ontological elements that are used to describe knowledge in this domain. A canon is described as a tuple <T,I,::,B>. T is a set of concept and relation types. I is a set of referents representing individual objects, either directly identifiable (by

[2] We chose to make this paper self-contained, and therefore, to include definitions that are well-known to CGers [3].

constants) or not (then identifiable by variables). The conformity relation :: encodes whether a particular individual, represented by a referent $i \in I$, can be interpreted as conformant to a particular type $t \in T$, written t::i. B is called the canonical basis of the CG system; it encodes the signature of every relation type. The signature of a relation type t of arity j is a tuple of j concept types, each one indicating the concept type that the i^{th} component of a relation of type t must conform to (for $i \in [1,j]$).

This section presents the formal definitions associated with each of the four basic elements of the description of a canon <T,I,::,B> with respect to a fixed set of objects O called the domain of the system. Some definitions come from [3]; others were added or adapted in order to complete the definition of the extensional semantics of the CG theory.

2.1 The Set of Types T

Here we present a single set of types which encompasses concept types and relation types (of different arity). We make this choice in order to simplify information interchange between systems wishing to establish a common ontology in order to communicate. Together, the elements of T form a partial order $(T,\leq)$ with one single top element $\top$, the universal type (the most general type), and a single bottom element $\bot$, the absurd type (the most specific type). Concept types and relation types are pairwise incomparable; relation types of different arity are pairwise incomparable. Formally, we introduce the set of types as follows.

Definition 1. Let $(T_c,<_c)$ be a partially ordered set of concept types, and $(T_r,<_r)$ be a partially ordered set of relation types with arity function arity: $T_r\backslash\{\top,\bot\} \rightarrow N^*$, such that:

1.1) $T_c \cap T_r = \{\top,\bot\}$;
1.2) $<_c \top$ and $\bot <_r \top$;
1.3) for all $t \in T_c\backslash\{\top,\bot\}$: $\bot <_c t <_c \top$, and for all $t \in T_r\backslash\{\top,\bot\}$: $\bot <_r t <_r \top$;
1.4) for all $t_1,t_2 \in T_r\backslash\{\top,\bot\}$: $arity(t_1) \neq arity(t_2)$ implies t_1 and t_2 are incomparable, i.e., neither $t_1 <_r t_2$ nor $t_2 <_r t_1$ holds.

The set of concept types is defined by $T = T_c \cup T_r$, and the partial ordering $<$ on T is defined as the union of $<_c$ and $<_r$.

2.2 The Set of Referents I

Referents represent objects of the reality being modeled, or artifacts found in computer models needed to describe objects of the reality. A referent is an *individual marker* (a constant value) when the object that it represents is known (identifiable); or it is a *variable* when the object that it represents is known to exist but has not yet been identified. Let M be the set of individual markers; let X be the set of variables.

In the traditional CG literature, a star symbol is often used in the referent field to denote a concept that represents an existing but unknown (unidentified) object. Alternatively, in that case, the whole referent field may be omitted altogether. Such a

concept is called a *generic* concept and is existentially quantified using some variable representing the object. This mechanism should be seen as a shortcut notation provided to the knowledge modeler only for ease of expression. In fact, even when not directly inputted by the knowledge modeler, the existentially quantified variable does exist; it identifies the object that the generic concept represents.

Finally, in order to represent the whole set of individuals complying with the concept type of some concept, the universal quantifier $\forall$ may be used as a referent. The elements of $(M \cup X)$ are called *existential referents*; the $\forall$ symbol is called the *universal referent*.

Definition 2. Let M be a set of individual markers with $\forall \notin M$, and let X be a set of variables with $\forall \notin X$, such that $M \cap X = \emptyset$. Then the set of referents I is defined by $I = M \cup X \cup \{\forall\}$.

However, the universal referent is used as a short-cut notation to represent the *distributive set* of all individuals conformant to a certain type. For instance, we could use the concept: [EMPLOYEE: $\forall$] to represent all the employees that we know of, i.e., [EMPLOYEE: #John], [EMPLOYEE: #Joe] and [EMPLOYEE: #Peter], which could also be represented using a typed distributive set: [EMPLOYEE: Dist{#John, #Joe, #Peter}] as defined in [3]. Consequently, we do not need and will not include the universal referent in the definitions below.

The relationship between the referents in I and the domain O of the CG system is given by an allocation function α, i.e., a mapping $\alpha: I\backslash\{\forall\} \rightarrow O$ that maps all existential referents onto objects in O. In this paper, we assume α to be fixed.

Assumption 1. (*Unique Name Assumption*) For all individual markers $i_1, i_2 \in M$, $i_1 \neq i_2$ implies $\alpha(i_1) \neq \alpha(i_2)$, i.e., different individual markers always represent different objects.

Assumption 2. *(Global Coreference Assumption)* Since all variables appear in I and are hence accessible to any embedded context, and since α is defined as a function, each variable is seen as a global referent to the object that it represents (i.e., accessible in any context).

Due to the unique name assumption, the set of individual markers M can be seen as a subset of O in the following sense: the set $\alpha(M) = \{\alpha(m) \mid m \in M\}$ is a subset of O and there exists a bijection from M into $\alpha(M)$. In other words, we might use the object o with $\alpha(m)=o$ as referent instead of the individual marker $m \in M$. Note, however, that the unique name assumption does not apply to the set of variables X, i.e., α is not injective on X. Also, in environments where $\alpha(x)$ is known to exist but can not be resolved to a particular (identifiable) element of O (e.g., in cases of partial knowledge), X cannot be seen as a subset of O analogously to M, and a referent $x \in X$ may not be resolved by its corresponding object $\alpha(x) \in O$.[3]

[3] Referent resolution may not be needed in order for different agents to communicate if the information about the existence of some object is all that is required for the communication

2.3 The Canonical Basis B

The canonical basis of a CG system contains basic graphs, called *canonical graphs*, from which any other graph is derived through the application of the canonical formation operators (see [3] for an introduction on each of these operators). The canonical basis B contains one canonical graph for each relation type in T_r; this graph is a model of how any relation of that type must be used. In other words, it constrains (restricts) the concept types of the concepts that such a relation may link.

Canonical graphs are sometimes called *signature graphs*. Through the projection operator, the CG system makes sure that any relation complies with its signature graph. That is, there must always be a projection of the signature graph of a relation type onto a relation of that type in any conceptual graph that uses it. By the definition of the projection operation, each concept linked to the relation must be a specialization of the corresponding concept type appearing in its signature graph. Therefore, the signature graph of a relation type encodes a constraint on the use of relations of that type.

Definition 3. Let s denote the signature function that maps each relation type $t \in T_r\backslash\{\top,\bot\}$ onto its signature $s(t) \in (T_c\backslash\{\bot\})^{arity(t)}$. Let comp(i,t) denote the i^{th} component of the signature of the relation type t, i.e., $comp(i,t) = t_i$ if $i \leq$ arity(t) and $s(t)= (t_1, \ldots , t_i, \ldots , t_{arity(t)})$. For i > arity(t), we define $comp(i,t) = \bot$, meaning that comp(i,t) is not defined for any i greater than arity(t). The signature function s is conformed to the set of relational types T_r iff for all $t_1,t_2 \in T_r\backslash\{\top,\bot\}$, if $t_1 \leq t_2$, then for all $1 \leq i \leq arity(t_1)$, $comp(i,t_1) \leq comp(i,t_2)$.[4] For a signature function s that is conformed to the set of relational types T_r, the canonical basis B is defined by $B = \{<t,s(t)> \mid t \in T_r\backslash\{\top,\bot\}\}$.

In order to proceed with the formalization, we first recall the formal definition of a simple conceptual graph. A *simple conceptual graph* over a set of types T is a labeled bipartite graph of the form $u = \langle C, R, E, l\rangle$, where C is the set of concept nodes and R is the set of relation nodes, $E \subseteq C\times R$ is the edge relation, and l is the labeling of u: each concept node $c\in C$ is labeled with a tuple $l(c) = (type(c), ref(c)) \in T_c \times I$, called the type and the referent of c (where type and ref are two functions such that type: $C \rightarrow T_c$ and ref: $C \rightarrow I$); each relation node $r \in R$ is labeled with a relation type $type(r) \in T_r$; finally, all edges that are incident to the same relation node $r \in R$ are labeled with an integer number $\in [1,n]$, where n = the arity of r, in such a way that the set of all labels incident to the same relation $r = \{i, \forall i \in [1,n]\}$ By definition, all edges whose label is smaller than n are called in-coming edges (pointed toward the relation), the edge whose label is equal to n is called out-going edge (pointed away from the relation). A concept node c is called the j^{th} neighbor of a relation node r of type t iff c is linked to r by an edge such that j = l(c,r). The referent of r in u is denoted by $<i_1,i_2,\ldots i_{arity(t)}>$, where i_j is the referent of j^{th} neighbor of r in u. The following

to proceed. In that case, referent resolution can be postponed to a later stage. This improves interoperability between different knowledge-based systems.

[4] Note that $t_1 \leq t_2$ implies $arity(t_1) = arity(t_2)$ (see Definition 1).

assumption formalizes the idea that each simple CG that is said to be derivable from a canon <T, I, ::, B> should conform to the signature given by B.

Assumption 3. (*Signature Compliance Assumption*) Let u be a simple CG. For all r in u and for t = type(r), $1 \leq i \leq$ arity(t), type(c) $\leq$ comp(i,t), where c is the i^{th} neighbor of r in u.

2.4 The Conformity Relation ::

The conformity relation establishes the basis for doing model checking on simple CGs over a set of types T. Recall that we consider a fixed domain O and a fixed allocation function α. In what follows, we also assume that there exists a fixed interpretation β of the types in T with regard to O, i.e., each concept type $t \in T_c$ is interpreted by a subset $\beta(t)$ of O, and each relation type $t \in T_r$ with arity(t)=n is interpreted by an n-ary relation over O, $\beta(t) \subseteq O^n$. Based on this fixed interpretation β, we can now define the extensional semantics of CGs over T. In this section, we first introduce the extensional semantics of basic CGs; in Section 3, we use these definitions to present the semantics of canonically derived CGs obtained from basic CGs by the iterative application of the formation operators. The conformity relation :: of the canon corresponds to the characteristic function of the interpretation β, and we get Definition 4, and consequently, Assumption 4.

Definition 4. The conformity relation :: maps types and (tuples of) referents onto {True,False} according to the interpretation β. For all concept types $t \in T_c$ and all referents $i \in I\backslash\{\forall\}$, t::i = True iff $\alpha(i) \in \beta(t)$. For all relation types $t \in T_r$ with arity(t)=n and all tuples $(i_1, ..., i_n) \in (I\backslash\{\forall\})^n$, t::$(i_1, ..., i_n)$ = True iff $(\alpha(i_1), ..., \alpha(i_n)) \in \beta(t)$. There is no $i \in I$ (tuple $(i_1, ..., i_n) \in I^n$) such that $\perp$::i ($\perp$::$(i_1, ..., i_n)$) holds. For all concept types $t \in T_c$, t::$\forall$ = True iff $\beta(t) \neq \emptyset$.[5]

Assumption 4. *(Type Subsumption Assumption)* For all concept types $t_1 \leq t_2$, and for all referents $i \in I$, t_1::i implies t_2::i. For all relation types t_1 and t_2 with $t_1 \leq t_2$ and arity(t_1) = n, and for all tuples $(i_1, ..., i_n) \in I^n$, t_1::$(i_1, \ldots, i_n)$ implies t_2::$(i_1, \ldots, i_n)$.

3 The Extensional Semantics of Derived Conceptual Graphs

The purpose of this section is to introduce the compositional semantics of the canonical formation operators so that simple model checking procedures can be developed according to the specificity of the CG-based model. Of course we could choose to use standard methods on FOL formulae produced by the application of the ϕ

[5] Once again the $\forall$ symbol is used only as a shortcut notation to represent the denotation set of type t. Hence the interpretation of concept [t:$\forall$] may be false if this set is empty.

operator on the graph for which model checking is required. However, since conceptual graphs are mainly existentially quantified assertions, model checking can be achieved according to the semantics of the formation operators themselves, yielding a much more direct and simpler procedure totally adapted to the nature of CG systems. Close to relational database theory, a model checking procedure adapted to the nature of the CG operators would facilitate rapid prototyping and subsequently easy implementation of CG-based systems. Having formally defined the canon of a CG-system with regard to a given domain O, an allocation function α, and an interpretation function β, we can now define the extensional semantics of CGs derived from the canon. A derived CG is obtained by the iterative application of the canonical formation operators [3] on *basic* conceptual graphs (i.e., graphs having a single concept node or a single relation node together with its connected concept nodes).

3.1 The Interpretation of Basic CGs

We first recall the definition of the ϕ operator and introduce the notion of *support* of a conceptual graph. Intuitively, the support $\delta(u)$ of a CG u denotes the set of all tuples of referents allocated by α in such a way that u evaluates to True. We can use the support to characterize the CGs that are valid with respect to the given interpretation.

Definition 5. Formally, we define value(u) := ($\delta(u) \bullet \varnothing$), where the support set of graph u is defined below.

Definition 6. Let u be a *single concept graph*, i.e., a basic CG of the form [t:i] with $t \in T_c$ and $i \in I$. Then t is used as a unary predicate and we define:

6.1) if $i \in M$, then $\phi(u) \mapsto t(i)$, where i is used as a constant; and $\delta(u) := \{<i>\}$ if $\alpha(i) \in \beta(t)$; otherwise, $\delta(u) := \varnothing$;

6.2) if $i \in X$, then $\phi(u) \mapsto \exists i\ t(i)$, where i is used as an existentially quantified variable; and $\delta(u) := \{<j> \mid j \in I \setminus \{\forall\}$ and $\alpha(j) \in \beta(t)\}$;

6.3) if $i = \forall$, $\phi(u) \mapsto (\wedge_{j \in \beta(t)}\ \phi([t:j]))$.[6]

And in each case, value(u) can be computed based on Definition 5.

Note that for a single concept graph u=[t:i] the support $\delta(u)$ coincides with the denotation set of t as introduced in [3].

Definition 7. Let u be a basic CG composed of a single conceptual relation r of type $t \in T_r \setminus \{\top,\bot\}$, linked to arity(t) concepts according to s(t), its signature graph. For $j \in [1,\text{arity}(t)]$, let concept(j,r,u) denote the j^{th} concept linked to r in u. Then t is used as an arity(t)-ary predicate and we define: $\phi(u) \mapsto \wedge_{j \in [1,\text{arity}(t)]}\ \phi(\text{concept}(j,r,u)) \wedge t(i_1,i_2,\ldots i_{\text{arity}(t)})$, where i_j denotes the referent of concept(j,r,u); and $\delta(u) := \{<i_1,i_2,\ldots i_{\text{arity}(t)}> \mid (\alpha(i_1), \alpha(i_2), \ldots, \alpha(i_{\text{arity}(t)})) \in \beta(t)\}$. And value(u) can be computed according to Definition 5.

[6] The universal referent being only a short-cut notation, 6.3 expands a universally quantified concept to its complete representation, so that 6.1 and 6.2 can be applied.

3.2 The Compositional Semantics of the CG Formation Operators

Canonically derived conceptual graphs are those obtained from the canon by the iterative application of the canonical formation operators onto basic conceptual graphs. This section shows how to compute the set of valid interpretations, the support, of such graphs.[7] Therefore, we present the compositional semantics of each operator. But before, we need to introduce useful definitions and assumptions which will help us define a precise mapping between the concepts of a graph u and the corresponding referents in the tuples of $\delta(u)$, its support.

Assumption 5. *(Normal Form Assumption)* A graph $u = \langle C,R,E,l \rangle$ is said to be in normal form iff for any pair of distinct concepts $c_1,c_2 \in C$, $ref(c_1) \bullet ref(c_2)$. Then we say that NF(u) holds.[8]

Definition 8. Based on some total ordering function, like a lexicographic comparison function between labels $<_l$, for any pair of distinct concepts $c_1,c_2 \in C$ from a graph u in normal form, either $l(c_1) <_l l(c_2)$ or $l(c_2) <_l l(c_1)$ holds. We can rank all concepts in a graph u according to this function. For any concept c_1 in u, let us define $pred(c_1,u) = \{c_2 \text{ in } C \mid c_1 \bullet c_2 \text{ and } c_2 <_l c_1\}$. Then it is easy to define an index function as $index(c_1,u) = |pred(c_1,u)| + 1$. And similarly, we can define $index^{-1}(i,u) = \{c \in C \text{ such that } index(c_1,u) = i\}$. If NF(u) holds, $|index^{-1}(i,u)| = 1$.

Now we can establish a mapping between concepts in a normal form graph u and the elements (referents) of each tuple in $\delta(u)$, its support. This mapping can be described as follows: for any i_j, the j^{th} element of any tuple $\langle i_1,i_2,\ldots i_{|C|} \rangle$ in $\delta(u)$ (with $j \in [1, |C|]$), we have: $j = index(c_1,u)$, that is, according to the valid interpretation of u represented by $\langle i_1,i_2,\ldots i_{|C|} \rangle$, i_j is the referent of concept c_1 in u for that interpretation. We also define $\delta(w)$ the support of a graph w, a subgraph of u, in terms of $\delta(u)$.

Definition 9. If w is a subgraph of $u = \langle C,R,E,l \rangle$ and $w = \langle C',R',E',l' \rangle$ with $C' \subseteq C$, $R' \subseteq R$, $E' \subseteq E$, $l' \subseteq l$, $m = |C'|$ and $n = |C|$, its support with regard to the support of u, written $\delta_s(w,u)$, is defined by construction. We have $\delta_s(w,u) := \{\langle i_{k_1}, i_{k_2}, \ldots, i_{k_m} \rangle$ where $k_p \in [1,n]$ for all $p \in [1,m]$, and where for each i_{k_p} there is a j such that $i_{k_p} = i_j$ in $\langle i_1, i_2, \ldots, i_n \rangle \in \delta(u)$, with $k_p \bullet k_q$ if $p \bullet q$ for all $p,q \in [1,m]\}$.

Having established mapping functions between concepts of a graph u and elements of its support $\delta(u)$, and having defined the support of a subgraph w in terms of the

---

[7] The canonical formation operators were described in [1], together with a proof of completeness and soundness. Therefore, only their compositional semantics will be described in this paper.

[8] The normal form of a conceptual graph was originally defined in [7]. Here we assume that all acquired graphs will be expressed under that form, thus we formulate an assumption to that effect. Of course, in practice, there could be actions to be taken to ensure that it is so.

support of the graph u from which w is a subgraph, we can now proceed with the definition of canonical formation operators.

The Copy Operator. Let $u = <C_1,R_1,E_1,l_1>$ and $v = <C_2,R_2,E_2,l_2>$. If v is a copy of u, written v := copy(u), then:
- $|C_2| = |C_1|$ and $C_2 \cap C_1 = \varnothing$
- $|R_2| = |R_1|$ and $R_2 \cap R_1 = \varnothing$
- $\exists m_1$: $C_1 \rightarrow C_2$ a bijective function mapping the concepts of C_1 onto those of C_2
- $\exists m_2$: $R_1 \rightarrow R_2$ a bijective function mapping the concepts of R_1 onto those of R_2
- $E_2 := \{(m_1(c),m_2(r)) \mid (c,r) \in E_1\}$
- $l_2(m_1(c)) := l_1(c)\ \forall c \in C_1$
- $l_2(m_2(r)) := l_1(r)\ \forall r \in R_1$
- $l_2(m_1(c),m_2(r)) := l_1(c,r)\ \forall (c,r) \in E_1$
- $NF(u) \Rightarrow NF(v)$ because $|C_2| = |C_1|$, and $l_2 = l_1$ on mapped concepts of u and v
- $\phi(v) := \phi(u)$ since v is isomorphic to u and $l_2 = l_1$ on mapped concepts and relations of u and v, and consequently, $\delta(v) := \delta(u)$ and value(v) := value(u).

The Simplify Operator. Let $u = <C_1,R_1,E_1,l_1>$ and $v = <C_2,R_2,E_2,l_2>$. If v is obtained from u by deleting a redundant[9] relation r of type t from u ($r \in R_1$), written v := simplify(r,u), then:
- $C_2 := C_1$
- $R_2 := R_1 \backslash \{r\}$
- $E_2 := E_1 \backslash \{(c,r) \mid c \in C_1\}$
- $l_2(c) := l_1(c)\ \forall c \in C_1$
- $l_2(r) := l_1(r)\ \forall r \in R_1$ (the set of labels is unchanged over the elements of R)
- $l_2(c,r) := l_1(c,r)\ \forall (c,r) \in E_2$
- $NF(u) \Rightarrow NF(v)$ because $C_2 = C_1$ and $l_2 = l_1$ on the concepts of u and v
- $\phi(v) := \phi(u)$ with a redundant predicate t(referent(r,u)) in $\phi(u)$ was deleted
- since r is redundant, $C_2 = C_1$, and $l_2 = l_1$, then: $\delta(v) := \delta(u)$ and value(v) := value(u).

The Restrict Operator. Let $u = <C_1,R_1,E_1,l_1>$, $v = <C_2,R_2,E_2,l_2>$, and let NF(u) hold. If v is obtained from u by restricting concept $c_1 = [t_1:i_1]$ in u to concept $c_2 = [t_2:i_2]$[10] (with $[t_2:i_2]$ being a specialization of $[t_1:i_1]$, and $l(c_2)$ being the new label of concept c_1), written v := restrict(c_1,c_2,u), then two cases arise: either 1) there is no c in C_1 such that $l_1(c) = l(c_2)$ or 2) $\exists c$ in C_1 such that $l_1(c) = l(c_2)$. In Case 1, the normal form assumption may not hold after the operation since two distinct labels may have the same referent (as a result of a restrict operation). In this case, additional steps (additional restrictions) may be needed to maintain the normal form assumption (see Case 2 below). In Case 2, if NF(u) holds, then NF(v) will automatically hold since no new

9 A relation r in u is said to be redundant if $\exists r'$ in u such that $l(r) = l(r')$ and $\forall j \in$ [1,arity(type(r))], concept(j,r,u) = concept(j,r',u).

10 Concepts c_1 and c_2 can be seen as single concept graphs: $<\{c_1\},\varnothing,\varnothing,l'>$ and $<\{c_2\},\varnothing,\varnothing,l>$ respectively.

label is introduced by the restrict operation. So Case 2 is defined to automatically join concepts having the same label. This is often called the *internal join operation* in CG literature [4].

Case 1: (there is no concept in u with the new label)
- $C_2 := C_1$
- $R_2 := R_1$
- $E_2 := E_1$
- $l_2(c) := l_1(c)\ \forall c \in C_1$ such that $c \bullet c_1$
- $l_2(c_1) := l(c_2)$
- $l_2(c,r) := l_1(c,r)\ \forall (c,r) \in E_1$
- $\phi(v) := \phi(u)$ where a (sub)formula $\phi(c_1)$ of $\phi(u)$ was substituted by formula $\phi(c_2)$
- since $l_2(c_1)$ is a specialization of $l_1(c_1)$, $C_2 = C_1$, and $l_2 = l_1$ for all concepts except c_1, then $\delta(v) := \{<i_{k_1},i_{k_2},\ldots i_{k|_{C_1}|}> \mid <i_1,i_2,\ldots i_{|_{C_1}|}> \in \delta(u),\ j = index(c,v),\ k_j = index(c,u),\ c = index^{-1}(j,v) = index^{-1}(k_j,u)$, and where $<i_p> \in (\delta_s(c_1,u) \cap \delta(c_2))$ for $p = index(c_1,u)\}$. Then Definition 5 can be applied to compute value(v).

Case 2: (there is already a concept in u with the new label)
- let $c', c \in C_1$ and $l_1(c') = l(c_2)$, then we define $m_3(c_1) := c'$ and $m_3(c) := c\ \forall c \neq c_1$
- $C_2 := C_1 \backslash \{c_1\}$
- $R_2 := R_1$
- $E_2 := \{(m_3(c),r) \mid (c,r) \in E_1\}$
- $l_2(c) := l_1(c)\ \forall c \in C_1 \backslash \{c_1\}$
- $l_2(r) := l_1(r)\ \forall r \in R_1$
- $l_2(m_3(c),r) := l_1(c,r)\ \forall (c,r) \in E_1$
- NF(v) holds because NF(u) holds and l_2 contains all the labels of l_1 but one.
- $\phi(v) := \phi(u)$ where subformula $\phi(c_1)$ was deleted.

In order to define $\delta(v)$ we now introduce a *tuple reduction* operator needed to eliminate a designated element in each tuple of its operand; this is useful for deleting a duplicate element.

> **Definition 10.** Let us define the tuple reduction operator $-_j$ that makes a copy of a tuple without the identified element j. That is, if $e = <i_1, i_2, \ldots, i_j, \ldots, i_n>$, then $-_j e = <i_1, i_2, \ldots, i_{j-1}, i_{j+1}, \ldots, i_n>$. So we can define: $\delta(v) := \{-_k<i_1,i_2,\ldots i_{|_{C_1}|}> \mid <i_1,i_2,\ldots i_{|_{C_1}|}> \in \delta(u),\ k = index(c_1,u),\ j = index(c',u)$, and $i_j = i_k\}$, and value(v) can be computed according to Definition 5.

In terms of relational database operators, the restrict operator described here performs a *select* according to a restricting condition on some element of the tuples on which it is applied.

The Join Operator. In this section we define the *maximal join* operator since it favors the enforcement of the normal form assumption over the resulting graph. Of course, after a join operation, two concepts could have different labels but the same

referent. As in Section 3.2.3 above, additional restriction operations may be needed to ensure that the newly joined graph is in normal form. At least, the maximal join operator joins all concepts which share the same label. Thus, we describe this operator below.

If $u = <C_1,R_1,E_1,l_1>$, $v = <C_2,R_2,E_2,l_2>$, $w = <C_3,R_3,E_3,l_3>$, and NF(u) and NF(v) holds, and if $\mathbf{x}$ is the maximal join operator applied on u and v, two distinct graphs, then we can write: $w = u \mathbf{x} v$. Let $C' = \{c \in C_2 \mid$ there is no $c' \in C_1$ with $l_1(c') = l_2(c)\}$. Then we have:

- $C_3 := C_1 \cup C'$
- $R_3 := R_1 \cup R_2$
- $\forall c \in C_2 \backslash C'$, we define $m_3(c) = c' \in C_1$ where $l_1(c') = l_2(c)$
- $\forall c \in C_3$, we define $m_3(c) = c$
- $E_3 := \{(m_3(c),r) \mid (c,r) \in E_1 \cup E_2\}$
- $l_3(c) := l_1(c) \; \forall c \in C_1$
- $l_3(c) := l_2(c) \; \forall c \in C'$
- $l_3(r) := l_2(r) \; \forall r \in R_1 \cup R_2$
- $l_3(m_3(c),r) := l_2(c,r)$ if $(c,r) \in E_2$
- $l_3(c,r) := l_1(c,r)$ if $(c,r) \in E_1$
- $\phi(w) :=$ the conjunction of $\phi(u)$ and $\phi(v)$ where subformulae $\phi(c)$ for all concepts c having duplicate labels were deleted

Let $J = \{(j_p,k_p)\}$, a set of associated indexes. Let $\otimes_J$ be the relational multiple join operator whose operands are tuples of arbitrary length. Tuples from each operand are joined (concatenated) whenever the j_p^{th} element of the first tuple is equal to the k_p^{th} element of the second tuple, *for all* $p \in [1, |J|]$. The concatenation is done by adding to the end of the tuple of the first operand, the non duplicate elements of the second operand, i..e, it is done in such a way to avoid duplicating the join attributes. That is the k_p^{th} element of the second tuple, for all $p \in [1, |J|]$ will not be part of the concatenated tuple. With our example, we have $J = \{(j_p,k_p)$ where $j_p = index(m_3(c_p),u)$ and $k_p = index(c_p,v)$, $\forall c_p \in C_2\backslash C'$ and for $p \in [1, |C_2\backslash C'|]\}$. If $m = |C_1|$, we have: $\delta(w) := \{<i_{k_1},i_{k_2},\ldots i_{k_n}> \mid <i_1,i_2,\ldots i_n> \in \delta(u) \otimes_J \delta(v)$, $j = index(c,w)$, $k_j = index(c,u)$ if $c \in C_1$ or $k_j = m+index(c,v)$ if $c \in C'$, $c = index^{-1}(j,w)\}$, and value(w) can be computed according to Definition 5.

3.3 The Interpretation of Universally Quantified Concepts

Universally quantified concepts represent *all* objects conforming to a certain type; they are used as a short-cut notation. Therefore, the extensional semantics of the CG notation does not depend on that notation. However, its use in a CG expresses a constraint that we describe here. If a universally quantified concept $c = [t:\forall]$ appears as the j^{th} concept in a normal form graph u, i.e., $index(c,u) = j \in [1,n]$ where $n = |C|$, that means that there is no restriction when it comes to the instantiation of that concept with one of the objects in $\beta(t)$, with regard to the truth value of u, as long as $\beta(t)$ is not empty. Consequently, if $e = <i_1, i_2, \ldots, i_j, \ldots, i_n> \in \delta(u)$, there should be

$|\delta(c)|$-1 other tuples with the exact same elements except for the j^{th} element which should be a different element of $\delta(c)$ each time. So we have: satisfiability

Assumption 6. *(Scope Satisfiability Assumption)* If the j^{th} concept c in a graph u is universally quantified, then for any $e = <i_1,i_2, \ldots, i_n>$ in $\delta(u)$ let us define $comp(e) = \{e' = <k_1,k_2, \ldots, k_n> \in \delta(u)$ where $i_p = k_p$ for all $p \bullet j\}$. If the union of the j^{th} element of each e' in comp(e) = $\delta(c)$, then we say that the predicate scope_satisfiability(u) holds (by default, scope_satisfiability(u) always hold for a graph u that does not contain any universally quantified concept).[11]

Definition 11. Let us now define the truth value of a normal form graph u as: truth_value(u) := value(u) $\wedge$ scope_satisfiability(u).

With the definitions given in this paper, we showed how the truth value of a normal form graph can be computed (as the outcome of Definition 11) for any graph derived from the canon, based on the interpretation given by β. Section 4 below discusses how the set of definitions given above can be extended to include the enforcement of semantic constraints.

4 Extended Canons

4.1 Semantic Constraints

Our previous work on constraints [5,6] showed how to formalize all semantic constraints found in database literature today in terms of a simple yet powerful mechanism based on conceptual graphs and on the projection operator.[12] For instance, if u_1 and v_1 are the graphs of Figures 1 and 2 below, then *false u_1 unless v_1* is constraint C_1 stating that: *no employee can work on a project unless s/he is assigned to that project*.

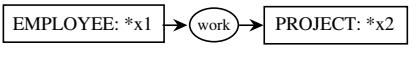


Fig. 1. A graph u_1 which should always be false according to constraint C_1.

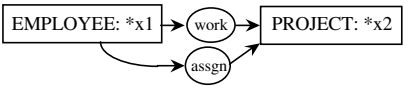


Fig. 2. The exception graph (v_1) to the constraint of Figure 1.

11 The advantage of having a scope_satisfiability predicate is that its definition may be extended to include other quantifiers without having to change the definitions given in this paper.

12 Since space is scarce, the reader is invited to read [3] as a more thorough introduction on the representation of semantic constraints under the CG formalism.

Among other things, we showed in [5,6] how to represent a relation type signature constraint using *false-unless* constructs. For example, the graphs of Figures 3 and 4 represent constraint C_2 = *false* u_2 *unless* v_2: *no relation of type agnt can be used unless its first parameter is of type ACTION_VERB and its second parameter is of type PHYSICAL_ENTITY.*

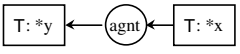


Fig. 3. Graph u_2, needed to represent constraint C_2 over the relation type agnt.

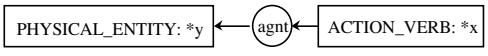


Fig. 4. Graph v_2, which is the signature graph of the relation type agnt.

By including constraints other than those encoded in B, this constraint representation mechanism allows the representation of knowledge according to a more precise model of the reality. Let H be the set of all semantic constraints related to the domain to be modeled (with $B \subseteq H$). We can now extend the definition of a canon by using <T,I,::,H> instead of <T,I,::,B>, a more general definition of what a canon is. Of course, we now need to define how the truth value of canonically derived graphs.

4.2 Validating Derived Graphs under an Extended Canon

The only difference with what was defined so far, is that we now have in H additional conditions under which the interpretation of a derived graph must be evaluated. For each constraint C_i of the form: *false* u_i *unless* v_i, we know that the support set $\delta(u_i)$ should be empty, otherwise the constraint would be violated, with the exception of elements in $\delta(u_i)$ that are also in the support set of the corresponding projection of u_i onto v_i, i.e., also in $\delta_s(u_i,v_i)$, since these elements satisfy the *unless* clause of the constraint. Let us define the support set of a constraint C_i = *false* u_i *unless* v_i as: $\delta_c(C_i) = \delta(u_i) \setminus \delta_s(u_i,v_i)$. We state below the constraint satisfiability assumption.

Definition 12. *(Constraint Satisfiability)* If C_i is a constraint in H and $\delta_c(C_i) = \varnothing$, then C_i is said to be satisfied. Otherwise C_i is said to be violated.

Definition 13. If u is a graph that violates some constraint, then truth-value(u) := False.

Definition 14. If truth_value(u) = True according to Definition 11 and truth_value(u) = False according to Definition 13, then the CG system is said to be inconsistent.

5 Conclusion and Future Developments

This paper presented the extensional semantics of the conceptual graph notation in a framework closely related to relational database (RDB) theory. The main motivation behind this choice is its subsequent facility of transfer from theory to practice. Furthermore, rapid prototyping, development and maintenance of CG based systems would highly benefit from well established theory and well tested tools such as those found in RDB literature. The database of a CG system, a set of existentially quantified assertions (graphs), is easily mapped onto the RDB model. Finally, years of practice under this model makes it a favorite with investors when a technological platform must be chosen for a large industrial project. Consequently, this paper favored this paradigm to provide an extensional semantics for the CG theory. It also described the compositional semantics of the canonical formation operators, along with a simple procedure to implement model checking. Right away one could think of applying RDB search heuristics to speed up model checking. We will leave that exploration for future work.

Acknowledgements

I wish to thank Mr. Ralf Molitor, Ph.D. student working under the supervision of Prof. Franz Baader, for his very useful comments over the different versions of this paper. By his comments, he convinced me that intuition falls short of being a good means of communication. Therefore, there is a world of difference between the first version of this paper that proposed to lay out the basic intuitive notions behind the extensional semantics of CGs, and this one, which provides formal definitions that forms the extensional semantics of the notation. For that, I thank him.

References

1. Mugnier, M.-L. & Chein, M.: Représenter des connaissances et raisonner avec des graphes. Revue d'Intelligence Artificielle, volume 10-1, (1996) 7-56
2. Prediger, S.: Simple Concept Graphs: A Logic Approach. Lecture Notes in AI, vol. 1453. Springer-Verlag, (1998) 225-239.
3. Sowa, J. F.: Conceptual Structures: Information Processing in Mind and Machine. Addison-Wesley, (1984).
4. Chein, M. & Mugnier, M.-L.: Conceptual Graphs: Fundamental Notions. Revue d'Intelligence Artificielle, volume 6-4, (1992) 365-406
5. Mineau, G.W. & Missaoui, R.: The Representation of Semantic Constraints in CG Systems. Lecture Notes in AI, vol. 1257. Springer-Verlag, (1997) 138-152.
6. Mineau, G.W. & Missaoui, R.: Semantic Constraints in Conceptual Graph Systems. DMR Consulting Group, R&D Division. Report #960611A. (1996)

Semantics of Attribute Relations in Conceptual Graphs

Pavel Kocura

Department of Computer Science
Loughborough University
Loughborough
Leicestershire LE11 3TU, England
P.Kocura@lboro.ac.uk

Abstract: The work analyses typical uses of attribute-like relations in conceptual graphs and identifies general sources of failure to convey intended meaning. These include relations that do not contain a semantic bond between their arguments, semantically superfluous concept types, and confusion between first-order and second-order levels. The analysis focuses on the semantics of physical quantities and results in their formal definition in terms of conceptual graphs.

1 Introduction

The inherent power and flexibility of the CGs formalism makes it potentially possible to formulate models of any semantic complexity. This power makes it particularly important that we fully understand the semantics and logic of the basic CGs building blocks. Understanding the semantics of CGs models becomes particularly critical should we embark on their full implementation. This requirement for semantic rigor has forced an extensive theoretical analysis and re-evaluation of the semantics of a number of typical CGs examples. Attention was quickly drawn to the traditional treatment of attribute-like relations, which frequently failed to convey the intended meaning of the graph. The main causes for this appeared to be:

Confusion about which parts of the domain knowledge should be represented by concepts, and which should map onto relations;

Overuse or misuse of the attribute relation linked to sink concepts that do not represent a valid space of attribute values;

Mixing object-level and meta-level statements in the same model without clearly defining, or even understanding, which is which;

Confusing a general property of an object of a given category with the space of the values for this property, and with the actual value of that property for a particular object.

Such problems lead to semantic ambiguity or inconsistency and result in a loss of knowledge. This paper will examine, in some depth, typical examples of incorrect use of attribute-like relations and suggest solutions that will avoid such problems.

B. Ganter and G.W. Mineau (Eds.): ICCS 2000, LNAI 1867, pp. 235-248, 2000.
© Springer-Verlag Berlin Heidelberg 2000

A few comments on our typography : Quoted graphs or formula examples are shown in Courier New. Any fragment of a graph or formula shown in this font retains its original meaning. Our own illustrative examples use Lucida Console.

2 Knowledge Content of Types and Relations

Let us illustrate issue a) above, starting with an example of the representational primitive of *Prehension*, [1] p. 85, Fig. 2.9, which represents a concrete fact of relatedness . It is symbolized by the relation (Has), as in:

```
[Mother]. (Has). [Child].
```

Graph 2.1

Sowa s interpretation of this graph, *Some mother has a child*, is not justified if it means that the woman is the child s biological mother. The type label CHILD could have two different meanings. The one apparently implicit in the graph, assuming the relation (CHILD), is given by:

```
type Child(x) is [PERSON]. (CHILD). [PERSON:*x].
```

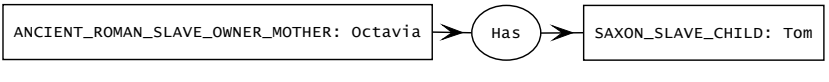
Definition 2.1

In this case, the interpretation of [Child] is *a person who is somebody's child*. The type Child thus defined has practically no knowledge content. Every person is somebody s child, i.e. (. x)Child(x) = PERSON(x), and there is no semantic link to the defining parent . Such a link could be only established by a definition constructed along the following lines:

```
type PAVEL'S_CHILD(x) is [MAN:Pavel]. (CHILD). [PERSON:*x].
```

Definition 2.2

The second interpretation of [CHILD] could be possibly given as *a person who is less than 13 years old*. Strictly speaking, CHILD is a role type, not a natural type, whereas PERSON is a natural type. As with Child, there is nothing in Graph 2.1 to show that the child has been born to the given woman, even if it has. If a woman s child is now a grown-up man, the type label CHILD would not apply to him, and we would have to change Graph 2.1 to [MOTHER]. (Has). [MAN], which suggests an interpretation radically different from *Some mother has a child*. Obviously, one could drastically dilute the original interpretation to *There exists an individual of type Mother and an individual of type Child that are related by relation Has*. Because it is so heavily overloaded, Has contains very little meaning on its own. It could stand-in for many valid relations between two concepts, but we would not know what the resulting graph would mean. In the graph



Graph 2.2

we can deduce from additional domain knowledge that Tom is not Octavia s child. We still do not know, though, if Tom is Octavia s slave, or whether she had him for dinner as a guest or as the main course. There is another aspect of (Has) that further complicates its semantics, namely, its strong temporal or ACT-like content. This means that Has is not a relation, but a state or a mapping from an act into a state, i.e. Has<STATE or Has<ACT, depending on the actual situation. We suggest either defining HAS rigorously for dedicated, specific use, or avoiding it altogether.

2.1 The Spurious Sink Concept

Relation (ATTR) may or may not have a temporal component, but it is even more overloaded than (Has) and suffers from the same semantic confusion, demonstrated by the next graph:

[PERSON]. (ATTR). [BIRTHDATE].

Graph 2.3

Let us define BIRTHDATE as the date of a day when somebody (anybody) was born. Similar to Graph 2.1, a [BIRTHDATE] may or may not be associated with a given person. There is more. Obviously, BIRTHDATE < DATE. Whereas DATE is a natural type, BIRTHDATE is a role type. Within the temporal limits of human existence, each day has been the birth date for some person. Thus, for the past x million years and, hopefully, a long way into the future, every date is a birth date:

$$(\forall\ x)\ \mathrm{DATE}(x) = \mathrm{BIRTHDATE}(x)$$

You may notice that BIRTHDATE and Child (as defined in Def. 2.1) fall into the same category, namely, that of role types that do not add any knowledge to the model. If we remove BIRTHDATE from our ontology, Graph 2.3 will become [PERSON]. (ATTR). [DATE]. Because ATTR is so weak in this interpretation, the graph does not say what the significance of the date to the person is: it could be the date of their dentist appointment, graduation, wedding, death, etc. We can solve the problem simply by using the relation (DATE_OF_BIRTH). This relation, either specified as primitive or explicitly defined, contains enough knowledge to represent the required link between the natural concepts [PERSON] and [DATE]:

[PERSON]. (DATE_OF_BIRTH). [DATE].

Graph 2.4

We suspect that some of the semantic confusion analyzed above stems for the conventional interpretation of database tables such as the one below, where the attribute

PERSON	BIRTHDATE
Tom	10/03/1960
Dick	20/04/1970
Harry	20/04/1970

name BIRTHDATE is effectively re-defined for each row, i.e. TOM'S_BIRTHDATE, DICK'S_BIRTHDATE, etc. This is possible because of the ordering of the table, which actually implies relation (DATE_OF_BIRTH) of graph 2.4. Because a CGs knowledge base, being a conjunction of monadic and generally n-adic predicates, is not structured in this way, such relations must be made fully explicit.

2.2 'Conceptualness' and 'Transitiveness'

This problem results from using the (Attr) relation in situations where it cannot have a meaningful sink concept, as shown by examples from [1], p. 434 and [2]:

[Relation]. (Attr). [Transitive]
[Graph:{*}]. (Attr). [Conceptual].

Type labels normally correspond to nouns or verbs of natural language. However, the type labels of the sink concepts above correspond to adjectives. Adjectives behave either attributively (e.g. rich man) o r predicatively (th e man is rich). They normally cannot stand on their own, nor can the concepts [Transitive] and [Conceptual]. This becomes more apparent if we try to instantiate them. The reading of [Transitive:#], [Conceptual:#] would be **the transitive* and **the conceptual*. We could use types *TRANSITIVENESS* or *CONCEPTUALNESS*, but then we would have to ask how many different values of transitiveness or conceptualness there are. Because concepts with such types cannot be instantiated, we conclude that these types are not semantically valid components of our ontology.

2.3 First-Order v. Second-Order Ambiguity

[Ski:#]. (Chrc). [Length]. (Amt). [Measure:<167,cm>].

Graph 2.5

In [1], p. 32, Sowa describes (Chrc) as a second-order relation that links an object to a characteristic, as in [Ball]. (Chrc). [Color: Red]. In comparison with [Color: Red], [Length] does not look like a second-order concept. If it were, it would have to have a first-order predicate as its referent, and the type label would have to be a second-order predicate, such as LENGTH_TYPE. We could suggest [LENGTH_TYPE: great_length] as a possible solution, but then the type label GREAT_LENGTH cannot be linked to the first-order relation (Amt) that links it to the first-order concept [Measure] - a type cannot be measured!

Let us accept that [Length] is a first-order concept, even though it means that [1] offers two contradicting interpretations for (Chrc). Now we may ask what its referents would be (as we would ask in the case of [GREAT_LENGTH]). Because [Length] is assumed to be independent of any actual measurement, its referents would have to be standard individual identifiers, each standing for a single length that can be associated with any number of physical objects and their dimensions, distances,

etc. Then an individual [Length:#] could be mapped into a number of equivalent values, each belonging to a different measurement system (e.g. imperial, metric, etc.). There are several considerations suggesting that [Length] and similar concepts are superfluous.

As there are infinitely many different lengths, there would have to be infinitely many individual identifiers that would conform to the type Length. This set of identifiers would have a one-to-one mapping, for a single measurement system, into the set of real numbers. Thus, these identifiers would be equivalent to non-negative real numbers, effectively duplicating them. This is neither necessary nor desirable. We can use real number concepts directly as the values of the 'length attribute' and connect them, through a suitable relation, to the concept of the object whose length we are modeling.

Another reason why we question the representational value of [Length] is that, arguably, no representation of a physical quantity can be fully independent of an actual measurement, whatever form it might take. Even comparing which of two sticks is longer, say, involves mapping them to viewing angles, which are 'measured' by the number of relevant photosensitive cells on the retina, and by using distance clues.

We conclude that [Length], as used in Graph 2.5, is superfluous. We reach the same conclusion with concepts such as [Speed], [Temperature], etc.

3 Semantics of Quantities

Another source of potential semantic ambiguity or inconsistency in a CGs-based model, with respect to modeling properties, is the use of 'complex' concept referents, especially those associated with quantities. To illustrate this problem, let us analyze in some detail the concept [Salary], taken from an example in [1], p. 34:

[Person:Tom]←(Agnt)←[Earn]→(Thme)→[Salary:@ $30,000].

Graph 3.1

Because the unique naming convention demands that the concepts representing actual salaries of different people be associated with different individual identifiers, Sowa adds a unique identifier into the referent field of [Salary], producing [Salary: #78902 @ $30,000]. This is different from the concept [Salary: #41337 @ $30,000] of a different person, Sue, who earns as little as Tom.

The Φ operator maps concept types to monadic predicates of FOL, and concept referents to corresponding predicate arguments. Before applying Φ to [Salary:…], we will slightly alter one of its 'referents' by separating the semantic components of the value $30,000, getting [Salary: #78902 @ $ 30,000]. This translates into the predicate Salary(#78902 @ $ 30,000). We can see that the type Salary corresponds to a tetradic predicate with arguments #78902, @, $, and 30,000. Actually, Salary is effectively a second-order predicate, as @ and $ subsume or imply first-order binary relations. To get an individual concept for each of these referents, we will decompose the complex structure of [Salary] into a graph whose concepts map into strictly first-order monadic predicates, and which has only first-order relations.

Individual identifier: Let us assume that Tom is paid in cash. Then we can think of the individual identifier #78902 as the surrogate for the actual bundle of all the bank notes and coins that Tom has received in his pay packets during the year: [MONEY_BUNDLE:#78902][1]. Sue s pay, [MONEY_BUNDLE:#41337], contained different notes and coins, but their total value was the same.

The @ symbol: Semantically @ represents a binary relation that links a bundle of money to its financial value expressed in terms of a unit of currency. We will replace @ with the relation (VALUE), leaving the sink concept still underspecified:

[MONEY_BUNDLE: #78902]. (VALUE). [*monetary quantity*].

Graph 3.2

Numerical value: Clearly, 30,000 is a number: [NUMBER: 30,000].

Currency unit: After detaching the $ symbol from the 30,000 and replacing it with the more specific US$, we can represent the latter as an individual referent that conforms to the type label CURRENCY_UNIT. To define a type label representing a monetary quantity in terms of a currency unit, we need a relation that would link the concept [CURRENCY_UNIT] to [NUMBER]. To get it, we will look for inspiration to the domain of physical quantities.

A physical quantity consists of a scalar that has a *dimension*, (DIM), given by a unit of measurement, where QUANT<NUMBER. A monetary quantity has the same structure:

```
type PHYS_QUANT(x) is[NUMBER:*x]. (DIM). [PHYSICAL_UNIT].
type MONEY_QUANT(x) is[NUMBER:*x]. (DIM). [CURRENCY_UNIT].
```

To produce a direct equivalent to the conventional notation, as in $30,000, we reconnect the numerical and dimensional (currency) information, using a type definition:

```
type US$_QUANT(x) is
                    [NUMBER:*x]. (DIM). [CURRENCY_UNIT:US$].
```

Definition 3.1

Obviously, US$_QUANT < NUMBER. However, the purpose of this type is not to show whether it applies to any particular number al l (non-negative) numbers with a relevant number of decimal places will conform to US$_QUANT. The purpose of this type label is to carry the measurement dimension information for use in numerical comparisons and arithmetic operations. Now we can formulate the following graph:

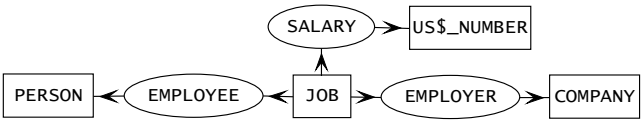
[MONEY_BUNDLE:#78902]. (VALUE). [US$_QUANT:30,000].

Graph 3.3

Why did we use the type label MONEY_BUNDLE instead of Salary, as in [Salary:#78902]? The fact that a sum of money was paid as a salary is coincidental a

[1] Obviously, this would not work for EFT (Electronic Funds Transfer). We could differentiate between a physical bundle of money and electronic bundle of money .

bundle of money could be used for a loan, bribe, ransom, it can be stolen or won in a poker game. Used as a type label, Salary represents a role for a bundle of money, as opposed to the natural type MONEY_BUNDLE. However, defined as an attribute relation, SALARY can be used as follows:



Graph 3.4

Apart from the role type vs. natural type dichotomy, the value of a salary is dissociated from any actual bundle of money because the payments are spread throughout the year. A new employee may have not received any money yet, but the monetary value of his/her salary is clearly specified in their contract. The relation (SALARY), as used in Graph 3.4, is not associated with such problems.

4 Semantics of Attributes and Values

In [1] are presented three examples of color representation (p.32, Fig. 1.9):

(Red). [Ball].

Graph 4.1

The relational label Red means here *red object*, not *red color*. Graph 4.1 translates into (. x:Ball)Red(x). Using RED_OBJ instead of Red, we can translate this typed predicate calculus formula into an equivalent FOL statement (. x){BALL(x)∧RED_OBJ(x)}. Graph 4.1 could be also re-expressed in typed calculus as (. x:BALL)(. y:RED_OBJ)x=y. Using Φ, both these formulas map back into CGs as [BALL]--[RED_OBJ]. However, Graph 4.1 can be also reformulated in typed calculus as (. x:RED_OBJ)BALL(x), which maps back into CGs as (BALL). [RED_OBJ]. We may notice here that Sowa s Red or our equivalent RED_OBJ do not explicitly represent a red color. The semantic correctness of these types is implied by the semantics of the color concepts used in their definitions.

[Ball]. (Attr). [Red].

Graph 4.2

In this graph, Red obviously means *red color*, not *red object* of Graph 4.1. The concept [RED_COLOR] can represent any physically possible shade of red, as defined by its hue, saturation and brightness (or using any other relevant color space system). We can think of a referent of the concept [COLOR] as of the paint code embossed on a car identification plate. Like (Has), (Attr) is a very general relation. This generality negatively affects the power and computational efficiency of our queries. For example, if we wanted to find in the knowledge base the ball s color, we would use as a query the graph [BALL]. (Attr). [COLOR]. The system would then have to project [COLOR] on every concept attached to [BALL] by (Attr). This projection would

involve any other types of attribute value spaces (for size, weight, price, etc.). Such matching would include costly accesses to the type lattice, or less costly type code comparisons. As we shall see later, using (Attr) in this way would also make it difficult to formulate meta-level definitions of, or queries about, the types of attributes, as opposed to their values. The solution to these problems is, as before, to put maximum information on the relation between the two relevant concepts:

[BALL]. (COLOR). [RED_COLOR].

Graph 4.3

For any non-colorless physical object, a more general graph would apply: [PHYS_OBJ]. (COLOR). [COLOR]. Here the label COLOR appears in two completely different roles, as a relational label and as a type label - we could name the relation (COLOR_OF). With a less human-oriented view of color identification, we could replace [COLOR] with a concept representing a measurement of light wavelength, brightness, etc., which would remove this apparent dual role.

Using COLOR as a relation, as in Graph 4.3, if we wanted to know the color of our ball, we would use the query [BALL]. (COLOR). [T]. The system would directly retrieve the sink concept of (COLOR), without having to access the type lattice or make type code comparisons. Let us consider the following graph:

[Ball]. (Chrc). [Color:Red].

Graph 4.4

Hypothetically, there could be two possible interpretations, depending on whether we define (Chrc) as (i) taking exclusively first-order concepts for arguments, or (ii) taking a second-order concept as its second argument.

First-order Color: We can interpret [Color:Red]as a first-order concept. Then (Chrc) would have the same meaning as (Attr), and there could be only one shade of red color in the whole domain, e.g. as if modeling traffic lights. The referent Red is the individual identifier of this color.

Second-order Color: [1] interprets Graph 4.4 as *there exists a ball, Red is a color, and the ball has Red as its characteristic*. Thus, Color is a second-order type and, presumably, the characteristic Red has some actual value., e.g. [Red:#4385], which is not shown. Then, to quote, (Chrc) relates an entity to a second-order characteristic like color, size or weight . To avoi d confusing this second-order Color with the first-order type label COLOR, as in COLOR > RED_COLOR > CRIMSON, and [CRIMSON:#3458], we should use the label COLOR_TYPE. Then Graph 4.4 becomes:

[BALL]. (Chrc). [COLOR_TYPE: red_color].

Graph 4.5

We might ask: *What was the question answered by Graph 4.5?* There are three possible questions that we could ask in the context of object properties:

a) What are the general properties that an object (or a category of objects) has?

b) What are the concrete properties (attribute values) of a given object?

c) What types of values are associated with a given property (attribute)?

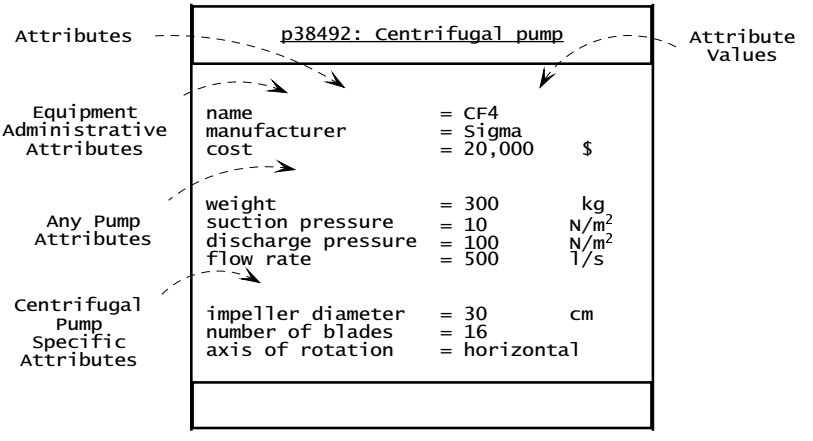


Fig. 4.1. Illustrating aspects a) c) above on an example represented in UML

The answer to question a) about pump p38492 would consist of the first column in the attribute compartment of the object box in Figure 4.1. Using another domain: a geometric point has a position in a co-ordinate system, but no length, width, thickness, color, mass, specific heat, or conductivity. A solid sphere will have diameter, mass, color (unless made from a completely transparent material), but no length, width or height (unless we define these from diameter), etc. By asking question b), we want to ascertain where, given a space of values, an object stands with respect to a property identified by a). For our pump, the answers are in the second column. Question c) determines what the space of the values of a specific property identified by a) is, e.g. for manufacturer the space is the set of company names, for weight it is the set of non-negative numbers, each multiplied by a unit of mass, for axis of rotation it is the set {horizontal, vertical}.

We can see that Graph 4.5 does not answer question b), which asks about attribute values, because Graph 4.5 does not contain a first-order concept [RED_COLOR] that would hold such a value in its referent. We suggest that Graph 4.5 attempts to answer question a). It does not quite succeed, because the sink argument of (Chrc), [COLOR_TYPE:red_color], clearly cannot represent the relation type COLOR, which we used in Graph 4.3. Instead, it represents a restricted space of values (i.e. value type) for the attribute COLOR. The individual attribute relations, the different attribute value spaces and nested subspaces serve as cognitive navigation points on a path (represented by the arrow lines) leading from an object to the values of its properties. We do not have to reach the ultimate value constant, but every step on this path will give us more specific information about the properties of the object. We may illustrate these differences by the following diagram:

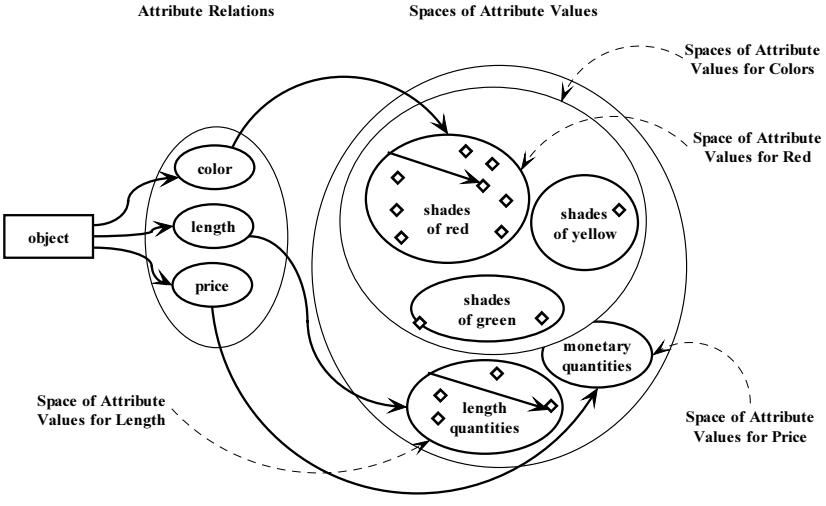


Fig.4.2 Attributes are relations, which point to progressively more focused spaces of attribute values

Inspired by this metaphor, we will interpret the relation label ATTR as representing the first step and define it formally. First, however, let us outline our development of a formal representation of physical attributes and quantities in CGs, based on the ideas discussed in previous sections.

5 Representing Physical Quantities

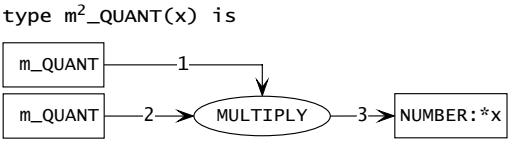
First we need a type for non-negative numbers:

```
type 0≤NUMBER(x) is [NUMBER:*x]. (=). [NUMBER:0].

type m_QUANT(x) is
        [0≤NUMBER:*x]. (DIM). [LENGTH_UNIT:m].
```

Definition 5.1 An m_QUANT is any non-negative quantity whose dimension is the meter.

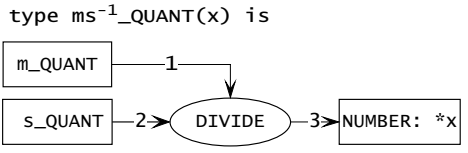
Obviously, LENGTH_UNIT::cm, km, mm, mile, etc. We need to use physical quantity units explicitly only when defining primary quantities: length, time, etc. To define derived quantities, we use u_QUANT types, where u is a physical unit.



Definition 5.2 A quantity in m^2 is the product of multiplying two quantities in m.

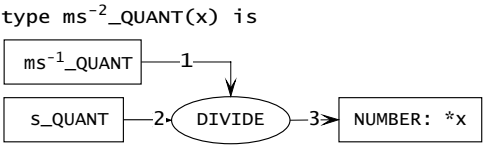
We could have defined m²_QUANT using the derived unit [AREA_UNIT: m²], which itself could be defined as an individual. The following graph defines a time quantity:

```
type s_QUANT(x) is [0<NUMBER:*x]. (DIM). [TIME_UNIT: s].
```



Definition 5.3 A length quantity (m) divided by a time quantity (s) produces a velocity quantity (m/s^2).

For the arcs of the relation (DIVIDE), 1= Dividend, 2= Divisor, 3= Quotient.

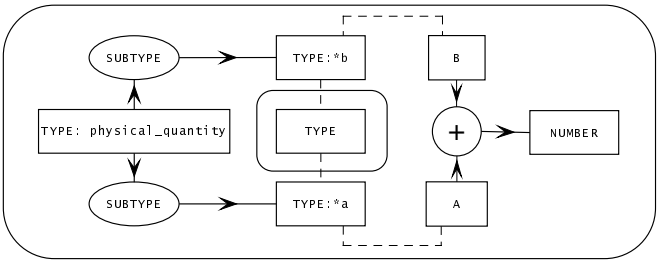


Definition 5.4 Acceleration is a quantity produced by dividing velocity (m/s) by time (s)

All the SI (or any other) units and corresponding physical quantities can be represented in CGs this way. This representation enables us to define and implement constraints on mathematical operations on physical quantities.

5.1 Arithmetic Operations with Physical Quantities

All the definitions of physical quantities, as well as the knowledge representation of attributes and attribute-like relations, can be fully deployed using the OWLS CGs theorem prover. The following OWLS script example formulates the rule that different physical quantities cannot be added together.



Graph 5.1 If there are two different subtypes of the type PHYS_QUANT then there does not exist a number that is the result of the sum of these quantities.[2]

[2] This does not include physical quantities that use only different units of the same type, e.g. meters, centimeters, miles, and thus can be numerically converted into each other.

6 ATTR as a Second-Order Relation

In the previous sections, we analyzed graphs, some containing the relational label Attr, that failed to carry their intended message. During the transformation of these graphs into more meaningful ones, the relation (Attr) invariably disappeared from the result. However, we might feel that there is a role for such a relation, especially in the context of the question *What are the permanent, 'inalienable' properties (attribute relations) associated with an object?* We will use ATTR to play a role similar to that intended for Sowa s second-order Chrc. Using our example of Graph 4.3, we suggest that (ATTR) is a second-order relation, with COLOR as the referent of its sink argument, [RELATION_TYPE: color]. But what is the source argument? If we used a first-order concept, e.g. [BALL], the following graph would assert the existence of a ball that is somehow associated with the relational label COLOR:

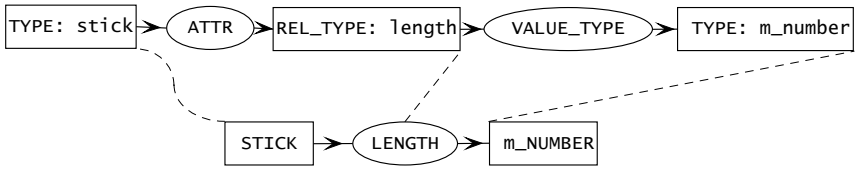
? [BALL]. (ATTR). [RELATION_TYPE: color].

Graph 6.1

That may not seem satisfactory for a several reasons:

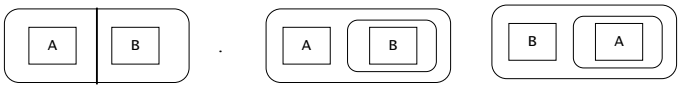
1) We may want to say that all balls have some color, not that there only exist some colorful balls. (Obviously, we could universally quantify over [BALL]);
2) We may want to describe the inheritance of properties from supertypes to subtypes;
3) We may want to keep connected graphs within FOL. (In Graph 6.1, the relation (ATTR) connects the object model with its meta-level).

We suggest that a possible solution is for (ATTR) to use second-order concepts for both its arguments, as in:

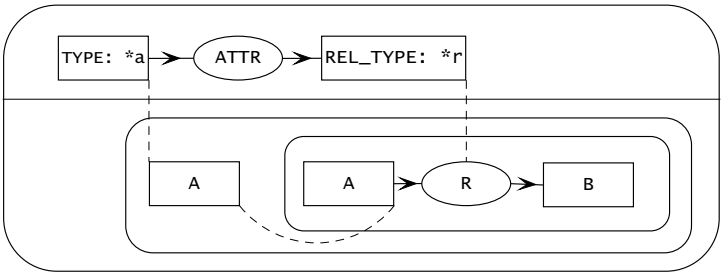


Graph 6.2

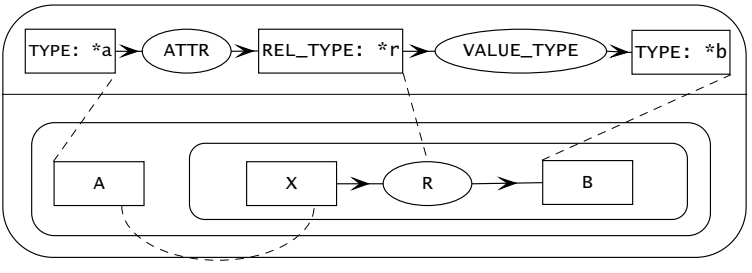
You may notice that the lines of identity connect between the referents of second-order concepts to the type labels of first-order concepts. The following graph shows the definition, by logical equivalence in terms of first-order concepts and relations, of the relation (ATTR). (We could equally well use a . -abstraction).



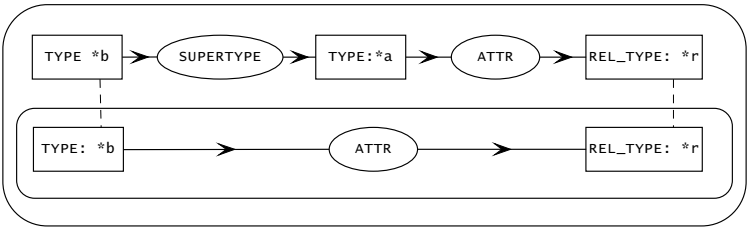
Definition 6.1 Notational shorthand for Peirce logic equivalence



Definition 6.2 Type A has attribute R iff for every instance of A there exists an instance of B that is connected to the instance of A by R.



Graph 6.3 This shows the relations between attributes and attribute value spaces



Graph 6.4 If type A is a supertype of type B and A has attribute R then B has attribute R

The definitions of physical quantities and of the (ATTR) relation enables us to reformulate Graph 2.5, [Ski: #]. (Chrc). [Length]. (Amt). [Measure: <167,cm>]. It breaks down into:

(i) A first-order statement that answers the query *How long is the ski?*

[SKI:#]. (LENGTH). [cm_NUMBER: 167].

(ii) A second-order statement, which is part of the answer to the request *Give me one property that all skis have*.

[TYPE: ski]. (ATTR). [REL_TYPE: length].

According to Def. 6.1, second-order relation (ATTR) points to the first-order attribute relations shared by a whole class of objects. However, we may want to show the unique attribute relations of a unique individual object. For that we would need to use a relation similar to Sowa s second-order (Chrc), or the version of (ATTR) shown in Graph 6.1. In that context, we might ask whether a completely ad-hoc relation that is not part of the semantic space of a fully defined type, i.e. one that is unique to an individual concept, can be classified as an attribute. This and other questions concerning meta-level descriptions of conceptual graphs will be addressed in another paper.

7 Conclusion

Our study of the semantics of attribute-like conceptual relations was necessitated by the need to define them rigorously and reason with them formally. We have attempted to identify, categorize and suggest solutions to some of the typical causes of semantic deficiencies, repeatedly found in publicly accessible examples of conceptual graphs. These causes included the use of weak relations that do not provide a sufficient semantic bond between their arguments, e.g. Has and, to a large extent Attr. Another source of problems was the use of types for the sink concepts of such relations that did not contain much or any relevant knowledge, e.g. [BIRTHDATE], [Child], [Length], [Transitive], etc. Based on our analysis, we have defined a formal representation of physical quantities, and shown a general approach to defining second-order relations from universally quantified first-order statements, as used in the definition of the (ATTR) relation. This work seems to have uncovered a number of interesting problems, which need to be explored further, both theoretically and empirically.

References

1. Sowa, John F.: Knowledge Representation: Logical, Philosophical and Computational Foundations, Brooks/Cole, New York, London (2000).
2. http://www.bestweb.net/~sowa/
3. Kocura, P.: Conceptual Graphs Canonicity and Semantic Constraints, In Proc. 4th Int. Conference on Conceptual Structures (b), ICCS 96, Sydney, Australia, ISBN 0 7334 1387 0, pp 133-145.

Nested Concept Graphs and Triadic Power Context Families

A Situation-Based Contextual Approach

Susanne Prediger

Technische Universität Darmstadt, Fachbereich Mathematik
Schloßgartenstr. 7, D–64289 Darmstadt, prediger@mathematik.tu-darmstadt.de

Abstract. In this paper, a contextual semantics for nested concept graphs shall be presented, that has especially two aims: We will propose a situation-based semantics that fits with Sowa's understanding of contexts and draw the connection to formal concept analysis by founding on formal contexts, more precisely triadic power context families. Its basic idea is to understand interpreted nested concept graphs as situation-based judgements in triadic power context families. With it, the logical theory can be well developed and inferences can be characterized in three ways: By semantical entailment, by means of a standard model and by a sound and complete calculus.

1 Introduction

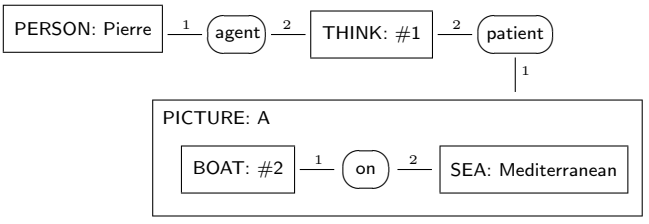
Several papers have been presented that develop the language of concept graphs over power context families in order to combine the approaches of formal concept analysis with the theory of conceptual graphs ([Wi97], [Wi98], [Pr98a], [Pr98b], [PW99]). The main idea of the logic approach presented in [Pr98b] was to define simple concept graphs as syntactical constructs with a semantics in power context families.

In this paper, an important extension for the language of conceptual graphs shall be included: the nestings in concept graphs. With nested conceptual graphs, knowledge can be structured and the coded information can be referred to different situations. A typical example is the concept graph shown below. It is taken (in a slightly changed form) from [MC96].

Its intuitive semantics is explained in the following way: The right answer to the question "Is there a boat on the Mediterranean Sea?" would have to be "Yes, in the context of picture A.", because the information given in the nesting of picture A only has a meaning in the context that is determined by A (cf. [MC96, S. 46]). Thus, this intuitive semantics is based on the notion of context as it is formalized in McCarthy's theory of contexts [McC93]. It is related to the notion of situations introduced by Jon Barwise and Jon Perry [BP83].

The discussion about a *formal* interpretation for nested conceptual graphs is not at all finished. Important contributions have been made by John Sowa [So84] (who developed a logical interpretation that exceeded the first-order predicate

B. Ganter and G.W. Mineau (Eds.): ICCS 2000, LNAI 1867, pp. 249–262, 2000.
© Springer-Verlag Berlin Heidelberg 2000



logic), Anne Preller [PMC95] (who developed a special logical language with a system of Gentzen sequents) and Geneviève Simonet ([Si96], [CMS98]). She proposed a logical interpretation in first-order logic where the predicates are extended by another argument describing the context. All these approaches only provide a logical syntax, but no explicit extensional semantics.

In this paper, a contextual semantics for nested concept graphs shall be presented. It has especially two aims: We will propose a situation-based semantics that fits with Sowa's understanding of contexts (cf. [So97]) and draw the connection to formal concept analysis by founding on formal contexts. Thus, the semantics is *contextual* in two senses: It is based on Sowas's understanding of contexts and uses the basic ideas of situation semantics, and it provides a semantics in formal contexts in the sense of formal concept analysis. That is why we call it a *situation-based, contextual semantics*.

As a foundation for the situation-based approach, we will introduce situations into the language of formal contexts in Section 2. Generalizing the logic approach for simple concept graphs (cf. [Pr98b]), we will define nested concept graph as syntactical constructs with a semantics in triadic power context families with situations in Section 3. In Section 4, we show that the logical theory of nested concept graphs can be developed as in the case of simple concept graphs. In particular, we can give three characterizations of inferences, a model-theoretic notion of entailment, the validity in the standard model and a sound and complete calculus for a syntactical characterization.

2 Situations in Triadic Power Context Families

A convincing formalization for situations was first given in Jon Barwise's situation theory [BP83] and has developed to the growing field of situation theory and situation semantics. Here, we follow Keith Devlin who starts his situation-based theory of information in [De91] with the formalization of the smallest units of information, the so-called *infons*. They look for example like this:

$$\ll \mathsf{marries}, \mathsf{Bob}, \mathsf{Carol}, \mathsf{New\ York}, 11/12/58, 1 \gg$$

It usually consists of a k-ary relation (here marries), k objects (here Bob, Carol), some locations (here New York, 11/12/58) and the polarity 1 (if the objects are in the relation at the given location) or 0 (if not) (cf. [De91, S. 35]).

Infons can refer to situations, then it is said "an infon is made true by a situation". As it is difficult to formalize *real situations*, Devlin proposes the definition of *abstract situations* as sets of infons.

These notions *infons* and *abstract situations* can easily be translated into the language of formal contexts: Using triadic power context families, an infon is an instance of the incidence relation of a context and a situation is a sub family of a power context family. For explaining this idea, let us first recall the formal definition of triadic contexts and triadic power context families as they are given in [LW95], [Wi95] and [Wi97].

Definition 1. *A* triadic (formal) context *is a quadruple* (G, M, B, Y)*, where* G*,* M *and* B *are sets whose elements are called (formal)* objects, attributes *and* conditions*, resp., and* Y *is a ternary relation between* G*,* M *and* B *where* $(g, m, b) \in Y$ *can be read as "object* g *has attribute* m *under condition* b*".*

In these triadic contexts, an infon $\ll R, g, b, 1 \gg$ (where R is a unary relation, g an object and b a location) can be understood as an instance of the ternary incidence relation Y of a triadic context. Thus, we consider the locations to be conditions of a triadic context. If relations of higher arity shall be expressed, we must consider a family of triadic contexts with identical sets of conditions where the sets of objects consist of tuples of objects. For it, we need triadic power context families.

Definition 2. *A* TRIADIC POWER CONTEXT FAMILY $\mathbb{K} := (\mathbb{K}_0, \ldots, \mathbb{K}_n)$ *of type* n *(for an* $n \in \mathbb{N}$*) is a family of triadic contexts* $\mathbb{K}_k := (G_k, M_k, B, Y_k)$ *that satisfy* $G_k \subseteq (G_0)^k$ *for each* $k = 1, \ldots, n$*. Then, we write* $\mathbb{K} = (G_k, M_k, B, Y_k)_{k=0,\ldots,n}$*.*

In the language of triadic power context families, the infon $\ll R, g_1, \ldots, g_k, b, 1 \gg$ can be expressed by "$((g_1, \ldots, g_k), R, b) \in Y_k$", and $\ll R, g_1, \ldots, g_k, b, 0 \gg$ by "$((g_1, \ldots, g_k), R, b) \notin Y_k$". (If the context is not considered to be dichotomic, we do not allow infons with polarity 0).

The notion of an abstract situation (being a set of infons) can be translated into the language of contexts as a sub family of a power context family. In the most general approach, arbitrary sub families were considered (cf. [Pr98a], but in this paper, we will restrict to so-called simple situations, that are sub families being one "layer" of the triadic power context family.

Definition 3. *A* (SIMPLE) SITUATION *in the triadic power context family* $\mathbb{K} := (G_k, M_k, B, Y_k)_{k=0,\ldots,n}$ *is a power context family* $\mathbb{S}_b := (G_k, M_k, \{b\}, I_k)_{k=0,\ldots,n}$ *satisfying* $b \in B$ *and* $I_k = Y_k \cap (G_k \times M_k \times \{b\})$ *for all* $k = 0, \ldots, n$*.*

In this formalization, there is a natural translation for infons and situations into the langauge of formal contexts: infons are the elementary judgements by which the formal contexts of a power context family and the situations are constituted. For being able to talk about the richer *conceptual structure* of a power context family, we must provide *concepts of situations* and judgements of the form "the object g falls under the concept $\mathfrak{c}$ in the situation s". Therefore, situative concepts are defined in [Pr98a] as triadic concepts of the sub families, i. e. of the situations. For this paper, we can simplify in the following way: As each simple situation only has one condition, we can consider situations like

dyadic formal contexts having "dyadic" formal concepts that are called situative concepts.

The units of thinking we work with are the so-called distributed concepts consisting of one situated concept in each situation. Thus, a distributed concept is a tuple of situative concepts that formalizes the continuity of concepts in different situations:

Definition 4. *A* situative concept of order k *(for a* $k \in \{0, \ldots, n\}$*) of a situation* $\mathbb{S}_b := (G_k, M_k, \{b\}, I_k)_{k=0,\ldots,n}$ *is a tuple* (A_1, A_2) *with* $A_1 \subseteq G_k$, $A_2 \subseteq M_k$ *where* (A_1, A_2) *is maximal with respect to the set inclusion under all triples with* $A_1 \times A_2 \times \{b\} \subseteq I_k$. *For* $\mathfrak{t} := (A_1, A_2)$, *we call Ext* $\mathfrak{t} := A_1$ *the* extension *of* $\mathfrak{t}$ *and Int* $\mathfrak{t} := A_2$ *the* intension *of* $\mathfrak{t}$. *The set of all situative concepts of* $\mathbb{S}_b$ *is denoted* $\underline{\mathfrak{B}}(\mathbb{S}_b)$.

A distributed concept of order k *of the triadic power context family* $\mathbb{K}$ *is a tuple* $(\mathfrak{t}_b)_{b\in B}$ *with* $\mathfrak{t}_b \in \underline{\mathfrak{B}}(\mathbb{S}_b)$ *for all* $b \in B$ *(and for a* $k \in \{0, \ldots, n\}$*). For two distributed concepts* $(\mathfrak{t}_b)_{b\in B}$ *and* $(\mathfrak{u}_b)_{b\in B}$, *we define the order* $\lesssim$ *by*

$$(\mathfrak{t}_b)_{b\in B} \lesssim (\mathfrak{u}_b)_{b\in B} \quad :\Longleftrightarrow \quad \forall\, b \in B:\ \mathit{Ext}\,(\mathfrak{t}_b) \subseteq \mathit{Ext}\,(\mathfrak{u}_b)$$
$$(\Longleftrightarrow \quad \forall\, b \in B:\ \mathit{Int}\,(\mathfrak{t}_b) \supseteq \mathit{Int}\,(\mathfrak{u}_b)).$$

As usual, we define the projection of the tuple onto its components by $\pi_b((\mathfrak{t}_b)_{b\in B}) := \mathfrak{t}_b =: \pi_{\mathbb{S}_b}((\mathfrak{t}_b)_{b\in B})$. It is an important operation, because we do usually not talk about the whole tuple of a distributed concept, but only about some situative concepts of some specific situations.

One important special case for distributed concepts are those that are intensionally defined (as Wille proposed implicitly in [Wi98]). Formally, we define:

Definition 5. *Let* $\mathbb{K} := (G_k, M_k, B, Y_k)_{k=0,\ldots,n}$ *be a triadic power context family. For a* $k = 0, \ldots, n$ *and* $N \subseteq M_k$, *the* intensionally defined concept $\langle N\rangle$ (of order k) *is defined by the largest distributed concept* $(t_b)_{b\in B}$ *satisfying* $N \subseteq \mathrm{Int}\,(t_b)$ *for all* $b \in B$.

For this definition, we use implicitely that for each situation $\mathbb{S}_b$, the extension of the intensionally defined concept $\langle N\rangle$ satisfies the following condition:

$$\mathrm{Ext}\ \pi_b(\langle N\rangle) = N^{Y_k \cap (G_k \times M_k \times \{b\})} := \{g \in G_k \mid (g, m, b) \in Y_k \text{ for all } m \in N\}$$

Those introduced notions will be needed in the next section to define the semantics for the language of nested concept graphs.

3 Syntax and Semantics for Nested Concept Graphs

According to the classification proposed by Chein and Mugnier in [CM97], we will consider *untyped, positive, nested concept graphs without generic markers.* That means, we do not allow negations, and the nestings are not specified by types.

The syntax of the language for nested concept graphs will be defined as it is done in [Pr98b]. We start with an alphabet of object names, concept names and

relation names and define the concept graphs as the well-formed formulas over this alphabet.

Definition 6. *The* alphabet *for nested concept graphs is a tuple* $\mathcal{A} := (\mathcal{G}, \mathcal{C}, \mathcal{R})$ *where* $\mathcal{G}$ *is a finite set of* object names, $(\mathcal{C}, \leq_{\mathcal{C}})$ *is a finite ordered set of* concept names, *and* $(\mathcal{R}, \leq_{\mathcal{R}})$ *is a finite set that is partitioned into finite ordered sets* $(\mathcal{R}_k, \leq_{\mathcal{R}_k})_{k=1,\ldots,n}$ *of* relation names.

In accordance with [Pr98b], the structure of concept graphs is described by means of nested directed multi-hypergraphs and labeling functions. We present the formal definitions and give an explanation and an example below.

Definition 7. *A* nested, directed multi-hypergraph *(of type n) is a quadruple* (V, E, ν, σ), *where*

- V *and* E *are finite sets of* vertices *and* edges, *resp.,*
- $\nu : E \to \bigcup_{k=1}^{n} V^k$ *is a mapping (with a* $n \geq 2$*),*
  (For $e \in E$ *with* $\nu(e) = (v_1, \ldots, v_k)$*, we denote* $\nu(e)\big|_i := v_i$ *and*
  $e^\nu := \{v \in V \mid \exists\, i = 1, \ldots, n : \nu(e)\big|_i = v\}$. *)*
- $\sigma: V \to \mathfrak{P}(V)$ *is a mapping such that, for all* $e \in E$ *and* $v, w \in V$ *with* $v \neq w$*, the conditions* $e^\nu \subseteq V \backslash \sigma(w)$ *or* $e^\nu \subseteq \sigma(w)$*,* $\sigma(v) \cap \sigma(w) = \emptyset$ *and* $v \notin \sigma^m(v)$ *are satisfied for all* $m \in \mathbb{N}^+$.

By the triple (V, E, ν), a directed multi-hypergraph is described (the mapping ν assigns the incident vertices to each directed edge). A vertex that is not incident to any edge is called an *isolated vertex*. The mapping σ describes the nestings: If $v \in \sigma(w)$, we say, v *belongs to the nesting of* w. The vertices with nestings, i. e. the vertices with $\sigma(w) \neq \emptyset$ are called *complex vertices*. All vertices which belong to any nesting are called *inner vertices*, the vertices with $v \notin \sigma(w)$ for all $w \in V$ are called *outer vertices*. The conditions for σ make sure that the nestings are closed with respect to the edges and that the nestings do not intersect and do not have cycles. The following denotations will be used: dom $\sigma := \{v \in V \mid \sigma(v) \neq \emptyset\}$, $\sigma^{-1}(v) := w$ for $w \in V$ with $v \in \sigma(w)$, and $\sigma^{-1}(v) = \emptyset$ for $v \in V \backslash \bigcup_{w \in V} \sigma(w)$. Analogically $\sigma^{-1}(e^\nu) := w$ for $w \in V$ with $e^\nu \subseteq \sigma(w)$, and $\sigma^{-1}(e^\nu) := \emptyset$ for $e^\nu \subseteq V \backslash \bigcup_{w \in V} \sigma(w)$.

With these notions and notations, a syntax our language can be specified by defining the structure of nested concept graphs.

Definition 8. *A* nested concept graph *(of type n) over the alphabet* $\mathcal{A} := (\mathcal{G}, \mathcal{C}, \mathcal{R})$ *is a structure* $\mathfrak{G} := (V, E, \nu, \sigma, \kappa, \rho)$ *where*

- (V, E, ν, σ) *is a nested, directed multi-hypergraph (of type n),*
- $\kappa : V \cup E \to \mathcal{C} \cup \mathcal{R}$ *is a mapping with* $\kappa(V) \subseteq \mathcal{C}$ *and* $\kappa(E) \subseteq \mathcal{R}$ *and for all* $e \in E$ *with* $\nu(e) = (v_1, \ldots, v_k)$ *is* $\kappa(e) \in \mathcal{R}_k$*, and*
- $\rho: V \to \mathcal{G}$ *is a mapping.*

As in the case of simple concept graphs, the mapping κ labels the vertices and edges with concepts and relation names. The mapping ρ assigns, to each vertex v, a *reference* $\rho(v)$. If $v \in \sigma(w)$, we say, v belongs to the nesting referenced by

$\rho(w)$. In order to abbreviate, we denote $\rho(e) := (\rho(v_1), \ldots, \rho(v_k))$ for the edge $e \in E$ with $\nu(e) = (v_1, \ldots, v_k)$ and call $\rho(e)$ the reference of e.

Let us explain the formal definitions by the example given on the first page: The nested concept graph has five vertices that are labeled (by means of κ) by the concept names PERSON, THINK, PICTURE, BOAT and SEA. The mapping ρ assigns the object names Pierre, #1, A, #2 and Mediterranean. The three edges are labeled (by means of κ) by the relation names agent, patient and on. The nesting is described by σ which assigns the vertices labeled with BOAT and SEA to the vertex labeled with PICTURE. To each other vertex, $\emptyset$ is assigned.

As we can see, the syntax of nested concept graphs is only a slight generalization of the syntax for simple concept graphs. We just have to add the mapping σ and some conditions. The crucial point is the *interpretation* of these syntactical constructs. Therefore, we will present a situation-based approach.

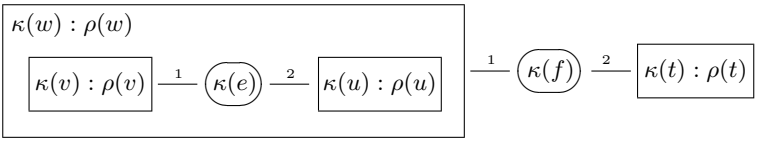
The basic idea for a contextual situation-based semantics for nested concept graphs is that we want to understand nested concept graphs as situation-based judgements in a triadic power context family. That means, whenever objects are put in relation by a syntactical construct, this is considered to be a situation-specific judgement that is interpreted in the restricted world of a formal context. Therefore, objects names are interpreted twice as objects and as situations of a power context family, because the situations themselves are references of concepts. By this double-interpretation, we respect the double character of the object names in nested concept graphs. The concept names are interpreted by distributed concepts of order 0 and the relation names of $\mathcal{R}_k$ (for $k = 1, \ldots, n$) by distributed concepts of order k. By this, concept names can be interpreted situation-based when we consider, in each situation, the corresponding component. More formally:

Definition 9. *Let $\mathcal{A} := (\mathcal{G}, \mathcal{C}, \mathcal{R})$ be an alphabet, $\mathbb{K} := (G_k, M_k, B, Y_k)_{k=0,\ldots,n}$ be a triadic power context family and $\mathfrak{S} := \{\mathbb{S}_b \mid b \in B\}$. We denote $\mathfrak{C}(\mathbb{K})$ for the set of distributed concepts of order 0, and (for each $k = 1, \ldots, n$) we denote $\mathfrak{R}_k(\mathbb{K})$ for the set of distributed concepts of $\mathbb{K}$ of order k and $\mathfrak{R}(\mathbb{K}) := \bigcup_{k=1,\ldots,n} \mathfrak{R}_k(\mathbb{K})$. The union $\iota := \iota_\mathcal{G} \dot\cup \iota_\mathcal{S} \dot\cup \iota_\mathcal{C} \dot\cup \iota_\mathcal{R}$ is called a* $\mathbb{K}$-interpretation *of $\mathcal{A}$, if the mappings satisfy the following conditions:*

- $\iota_\mathcal{G}: \mathcal{G} \to G_0$ *and* $\iota_\mathcal{S}: \mathcal{G} \to \mathfrak{S}$,
- $\iota_\mathcal{C}: \mathcal{C} \to \mathfrak{C}(\mathbb{K})$ *is order-preserving and*
- $\iota_\mathcal{R}: \mathcal{R} \to \mathfrak{R}(\mathbb{K})$ *is order-preserving and satisfy $\iota_\mathcal{R}(\mathcal{R}_k) \subseteq \mathfrak{R}_k(\mathbb{K})$ for all $k = 1, \ldots, n$.*

Now, it can be specified, under what conditions a concept graph is *valid* under a $\mathbb{K}$-interpretation. For the concept graph in our first example, we will postulate for example, that the object by which the object name #2 is interpreted, really falls under the situative concept interpreting BOAT in the situation interpreting A. Thus, we have to assign BOAT to a distributed concept whose component for the situation interpreting A satisfies this condition. With it, we have formalized the sub judgment "$\iota_\mathcal{G}$(#2) falls under the concept $\iota_\mathcal{C}$(BOOT) in the situation $\iota_\mathcal{S}$(A)." (see the formal definition). For outer vertices, we must specify a universal

situation $\mathbb{S}_u$ and claim the analogical conditions for the references. The edge conditions can be formulated in a similar way. For the formal definition, it might help to look at the following schematized nested concept graph.



Definition 10. *Let $\mathcal{A}$ be an alphabet and $\mathbb{K}$ a triadic power context family. If ι is a $\mathbb{K}$-interpretation of $\mathcal{A}$ and $u \in B$, the triple $(\mathbb{K}, u, \iota)$ is called a* context-interpretation *of $\mathcal{A}$.*

The nested concept graph $\mathfrak{G} := (V, E, \nu, \sigma, \kappa, \rho)$ is called valid *in $(\mathbb{K}, u, \iota)$, and $(\mathbb{K}, u, \iota)$ is called* model for $\mathfrak{G}$*, if*

i.) *all $v, w \in V$ with $v \in \sigma(w)$ satisfy* $\iota_{\mathcal{G}}\rho(v) \in \text{Ext}\ (\pi_{\iota_{\mathcal{S}}(\rho(w))}\, \iota_{\mathcal{C}}\kappa(v))$,

ii.) *all $v \in V \setminus \bigcup_{w \in V} \sigma(w)$ satisfy* $\iota_{\mathcal{G}}\rho(v) \in \text{Ext}\ (\pi_{\mathbb{S}_u}\, \iota_{\mathcal{C}}\kappa(v))$,

iii.) *all $e \in E$ with $e^\nu \subseteq \sigma(w)$ for a $w \in V$ satisfy* $\iota_{\mathcal{G}}\rho(e) \in \text{Ext}\ (\pi_{\iota_{\mathcal{S}}(\rho(w))}\, \iota_{\mathcal{R}}\kappa(e))$,

iv.) *all $e \in E$ with $e^\nu \subseteq V \setminus \bigcup_{w \in V} \sigma(w)$ satisfy* $\iota_{\mathcal{G}}\rho(e) \in \text{Ext}\ (\pi_{\mathbb{S}_u}\, \iota_{\mathcal{R}}\kappa(e))$.

The conditions i.) and ii.) generalize the *vertex conditions* for inner and outer vertices, the conditions iii.) and iv.) are *edge conditions* for inner and outer edges.

To sum up, we have introduced a semantics that allows an interpretation of nested concept graphs in triadic power context families with situations. By this, the judgements are interpreted situation-specifically, and in this way, we obtain a contextual, situation-based semantics.

4 Reasoning with Nested Concept Graphs

The Standard Model and Semantical Entailment

For reasoning with nested concept graphs, we can specify a standard model in which all the information that is coded in the concept graph is given on the context level. Before giving a precise formal definition for the standard model, we give an idea how the power context family for the standard model is constructed. Starting with a nested concept graph $\mathfrak{G} := (V, E, \nu, \sigma, \kappa, \rho)$ and an alphabet $\mathcal{A} := (\mathcal{G}, \mathcal{C}, \mathcal{R})$, we take the object names of $\mathcal{G}$ as objects of the context $\mathbb{K}_0$. The sets of objects for the higher contexts $\mathbb{K}_k$ are constructed by $\mathcal{G}^k$. As sets of attributes for each context $\mathbb{K}_k$, we take the concept names or relation names with corresponding arity, respectively.

For being able to construct all necessary situations, the conditions of the contexts are defined by the set of all object names of $\rho(\text{dom}\ \sigma)$, that means by all object names that appear as a reference of a complex vertex. Then, we add an universal condition u for outer vertices and an "empty condition" ε for the references of non-complex vertices.

If we chose the right incidence relations Y_k, we can interpret the object names by themselves, i.e. $\iota_{\mathcal{G}}^{\mathfrak{G}} = id_{\mathcal{G}}$, and by $\iota_{\mathcal{S}}^{\mathfrak{G}}$ with $\iota_{\mathcal{S}}^{\mathfrak{G}}(g) = \mathbb{S}_g$ or $\mathbb{S}_\varepsilon$. The concept and relation names are interpreted by $\iota_{\mathcal{C}}^{\mathfrak{G}}$ and $\iota_{\mathcal{R}}^{\mathfrak{G}}$ the corresponding intensionally defined concept.

Definition 11. *For the nested concept graph* $\mathfrak{G} := (V, E, \nu, \sigma, \kappa, \rho)$ *over the* $\mathcal{A} := (\mathcal{G}, \mathcal{C}, \mathcal{R})$*, the triadic power context family* $\mathbb{K}^{\mathfrak{G}} := (G_k^{\mathfrak{G}}, M_k^{\mathfrak{G}}, B^{\mathfrak{G}}, Y_k^{\mathfrak{G}})_{k=0,\ldots,n}$ *is defined by* $G_0^{\mathfrak{G}} := \mathcal{G}$*,* $M_0^{\mathfrak{G}} := \mathcal{C}$*,* $G_k^{\mathfrak{G}} := \mathcal{G}^k$ *and* $M_k^{\mathfrak{G}} := \mathcal{R}_k$ *for all* $k = 1, \ldots, n$ *and* $B^{\mathfrak{G}} := \rho(\mathrm{dom}\ \sigma) \dot\cup \{u, \varepsilon\}$.

For all $k = 0, \ldots, n$*, the incidence relations* $Y_k^{\mathfrak{G}}$ *are defined in such a way that all* $g \in \mathcal{G}$*,* $(g_1, \ldots, g_k) \in \mathcal{G}^k$*,* $c \in \mathcal{C}$*,* $R \in \mathcal{R}_k$ *with* $k = 1, \ldots, n$ *and* $b \in B^{\mathfrak{G}} \backslash \{u, \varepsilon\}$ *satisfy*

- $(g, c, b) \in Y_0^{\mathfrak{G}} \quad :\Longleftrightarrow \quad \exists\, v, w \in V :$
  $v \in \sigma(w)$, $\rho(v) = g$, $\kappa(v) \leq_{\mathcal{C}} c$ *and* $\rho(w) = b$,
- $(g, c, u) \in Y_0^{\mathfrak{G}} \quad :\Longleftrightarrow \quad \exists\, v \in V \backslash \bigcup_{w \in V} \sigma(w) :$
  $\rho(v) = g$ *and* $\kappa(v) \leq_{\mathcal{C}} c$,
- $((g_1, \ldots, g_k), R, b) \in Y_k^{\mathfrak{G}} \quad :\Longleftrightarrow \quad \exists\, e \in E :$
  $\rho(e) = (g_1, \ldots, g_k)$, $\kappa(e) \leq_{\mathcal{R}} R$ *and* $\rho(\sigma^{-1}(e^\nu)) = b$,
- $((g_1, \ldots, g_k), R, u) \in Y_k^{\mathfrak{G}} \quad :\Longleftrightarrow \quad \exists\, e \in E :$
  $\rho(e) = (g_1, \ldots, g_k)$, $\kappa(e) \leq_{\mathcal{R}} R$ *and* $\sigma^{-1}(e^\nu) = \emptyset$.
- $(g, c, \varepsilon) \notin Y_0^{\mathfrak{G}}$ *and* $((g_1, \ldots, g_k), R, \varepsilon) \notin Y_k^{\mathfrak{G}}$

The $\mathbb{K}$*-interpretation* $\iota^{\mathfrak{G}} := \iota_{\mathcal{G}}^{\mathfrak{G}} \dot\cup \iota_{\mathcal{S}}^{\mathfrak{G}} \dot\cup \iota_{\mathcal{C}}^{\mathfrak{G}} \dot\cup \iota_{\mathcal{R}}^{\mathfrak{G}}$ *is defined by* $\iota_{\mathcal{G}}^{\mathfrak{G}} := id_{\mathcal{G}}$*;* $\iota_{\mathcal{S}}^{\mathfrak{G}}(g) := \mathbb{S}_g$ *for all* $g \in \rho(\mathrm{dom}\ \sigma)$ *and* $\iota_{\mathcal{S}}^{\mathfrak{G}}(g) := \mathbb{S}_\varepsilon$ *for all* $g \in \mathcal{G} \backslash \rho(\mathrm{dom}\ \sigma)$*;* $\iota_{\mathcal{C}}^{\mathfrak{G}}(c) := \langle c \rangle$ *for all* $c \in \mathcal{C}$*; and* $\iota_{\mathcal{R}}^{\mathfrak{G}}(R) := \langle R \rangle$ *for all* $R \in \mathcal{R}$*. The context-interpretation* $(\mathbb{K}^{\mathfrak{G}}, u, \iota^{\mathfrak{G}})$ *is called the* standard model *of* $\mathfrak{G}$*.*

It is easy to prove that the standard model is really a model for $\mathfrak{G}$ (by checking vertex and edge conditions).

The standard model is a useful tool to do reasoning with nested concept graphs. As usual in logic, inferences of nested concept graphs over the same alphabet can be mathematized by semantical entailment. Let me shortly recall the definition: We say that the nested concept graph $\mathfrak{G}_1$ *entails (semantically)* $\mathfrak{G}_2$, if $\mathfrak{G}_2$ is valid in all models for $\mathfrak{G}_1$. Then, we denote $\mathfrak{G}_1 \models \mathfrak{G}_2$. This relation can be characterized by means of the standard model as it is done in the following proposition (which is proved in the appendix).

Proposition 1. *The nested concept graph* $\mathfrak{G}_1$ *entails* $\mathfrak{G}_2$*, if and only if* $\mathfrak{G}_2$ *is valid in the standard model of* $\mathfrak{G}_1$*.*

That means, by translating the information given in a nested concept graph onto the context level, i.e. by constructing the standard model, we can easily treat questions of inference. For every concept graph, we decide if it is entailed by checking the validity in the standard model.

Inference Rules

In addition to these two ways to treat inferences, we can give a characterization on the syntactical level, that means we can specify a sound and complete set of inference rules with respect to the semantical entailment.

Definition 12. *We call the nested concept graph* $\mathfrak{G}_2$ derivable *from* $\mathfrak{G}_1$ *(and denote* $\mathfrak{G}_1 \vdash \mathfrak{G}_2$*), if* $\mathfrak{G}_2$ *can be derived by the following inference rules (which are elaborated in the appendix):*

1. Double a vertex.
   Double a vertex v *and its incident edges (several times if* v *occurs more than once in* $\nu(e)$*). Double also all vertices and edges, that are nested in the doubled vertex. Extend the mappings* κ *and* ρ *to the doubles.*
2. Delete an isolated, non-complex vertex.
   Delete a vertex v *and restrict* κ *and* ρ *accordingly.*
3. Double an edge.
   Double an edge e *and extend the mappings* ν *and* κ *to the double.*
4. Delete an edge.
   Delete an edge e *and restrict the mappings* ν *and* κ *accordingly.*
5. Exchange a concept name.
   Substitute the assignment $v \mapsto c$ *for* $v \mapsto \kappa(v)$ *for such a concept name* $c \in \mathcal{C}$ *for which there is a vertex* $w \in V$ *with* $\kappa(w) \leq_{\mathcal{C}} c$*,* $\rho(v) = \rho(w)$*, and* $\rho(\sigma^{-1}(v)) = \rho(\sigma^{-1}(w))$ *(i.e. a vertex that has the same reference and is in a nesting with equal reference).*
6. Exchange a relation name.
   Substitute the assignment $e \mapsto R$ *for* $e \mapsto \kappa(e)$ *for such a relation name* $R \in \mathcal{R}$ *for which there is an edge* $f \in E$ *with* $\kappa(f) \leq_{\mathcal{C}} R$*,* $\rho(e) \subseteq \rho(f)$*, and* $\rho(\sigma^{-1}(e^{\nu})) = \rho(\sigma^{-1}(f^{\nu}))$.
7. Join vertices with equal references in the same nesting.
   Join two vertices $v, w \in V$ *satisfying* $\rho(v) = \rho(w)$ *and* $\sigma^{-1}(v) = \sigma^{-1}(w)$ *into a vertex* $v \vee w$ *with the same incident edges and references, and set* $\kappa(v \vee w) = c$ *for a* $c \in \mathcal{C}$ *with* $\kappa(v) \leq_{\mathcal{C}} c$ *and* $\kappa(w) \leq_{\mathcal{C}} c$*. The sub graphs that are nested in* v *and* w *are combined by juxtaposition and nested in* $v \vee w$.
8. Copy a sub concept graph into an equally referenced nesting.
   Construct a sub graph (with all complete edges and nestings) that is identical to the first sub graph (up to the names of vertices and edges) into a nesting with equal references.

Proposition 2 (Soundness and Completeness).
Two nested concept graph $\mathfrak{G}_1$ *and* $\mathfrak{G}_2$ *over the same alphabet satisfy*

$$\mathfrak{G}_1 \models \mathfrak{G}_2 \quad \Longleftrightarrow \quad \mathfrak{G}_1 \vdash \mathfrak{G}_2.$$

The soundness of these rules can be shown easily when the rules are specified more formally (see appendix). Then, the proof is analogical to the proof of soundness for simple concept graphs in [Pr98b]. An idea for the proof of completeness is given in the appendix.

On the whole, we have found a sound and complete system of inference rules for nested concept graphs. Thus, we have three possibilities to characterize inferences: semantical entailment, syntactical derivability and validity in the standard model. This shows that the semantics we proposed in Section 3 allows to develop a rich logical theory for nested concept graphs.

5 Coding Nested Concept Graphs in Graphs and in Contexts

By constructing the standard model of a given nested concept graph, we can translate the information coded in the concept graph into the context level. Proposition 1 makes sure that the informational content is really identical.

There is also a way back from the context level to the graph level: For a given triadic power context family, we can construct a nested concept graph that encodes the same information. Logically speaking, this is the nested concept graph that entails all concept graphs being valid in the power context family. We will only give an idea of the construction of this so-called standard graph. For details, see [Pr98a].

We start with a triadic power context family $\mathbb{K}$, a $u \in B$ and an (injective) mapping $o : B\backslash\{u\} \to G_0$ that assigns an object to each condition of $\mathbb{K}$. In order to determine the standard graph, we must specify an alphabet whose elements are interpreted in the power context family. The set of object names will be G_0, the sets of concept and relation names will be the power set of attributes of the corresponding contexts, i.e. we define $\mathcal{A}^{\mathbb{K}} := (G_0, \mathfrak{P}(M_0), \bigcup_{k=1,\ldots,n} \mathfrak{P}(M_k))$ and interprete each concept and relation name by the generated intensionally defined distributed concept.

For the construction of the standard graph, we proceed as follows: For each object $g \in G_0$ and each condition $b \in B$, we construct the vertex (g, b) that is referenced with g and labeled with the concept intent of the smallest situative concept of $\mathbb{S}_b$ that has g in its extension. We can easily see that this is exactly the set $g^{Y_0^b} := \{m \in M_0 \mid (g, m, b) \in Y_0\}$. Analogically, we construct the edges. For each k-ary tuple $(g_1, \ldots, g_k) \in G_k$ and each $b \in B$, we construct an edge $((g_1, \ldots, g_k), b)$. It is labeled by $(g_1, \ldots, g_k)^{Y_0^b} := \{m \in M_k \mid ((g_1, \ldots, g_k), m, b) \in Y_k\}$, and its incident vertices are $(g_1, b), \ldots, (g_k, b)$. Edges and vertices that belong to a condition $b \in B\backslash\{u\}$ are nested in the vertex $(o(b), u)$. In this way, we obtain a nested concept graph with nestings of maximal depth 1. Its non-complex vertices are of the form (g, u) where $g \notin o(B\backslash\{u\})$.

It can be shown that this standard graph is valid in the model we started from and that it has the wanted property to entail all other nested concept graphs being valid in $(\mathbb{K}, u, \iota^{\mathbb{K}})$, i.e. that it encodes, with respect to the alphabet $\mathcal{A}^{\mathbb{K}}$, all information of the power context family $\mathbb{K}$.

With it, we have the possibility to translate information which is given on the context level onto the graph level and vice versa. This allows an integration of nested concept graphs into the framework of formal concept analysis. Thus, nested concept graphs can be activated for conceptual knowledge processing

with power context families. Knowledge systems can be constructed in which knowledge is coded in power context families and in which nested concept graphs serve for the communication about the knowledge.

6 Appendix: Formal Definitions and Proofs

Proof for Proposition 1. We only show the non-trivial implication: If the nested concept graph $\mathfrak{G}_2 := (V_2, E_2, \nu_2, \sigma_2, \kappa_2, \rho_2)$ is valid in the standard model $\mathbb{M}^{\mathfrak{G}_1} := (\mathbb{K}^{\mathfrak{G}_1}, u, \iota^{\mathfrak{G}_1})$ of $\mathfrak{G}_1 := (V_1, E_1, \nu_1, \sigma_1, \kappa_1, \rho_1)$, then it is valid in every other model $\mathbb{M} := (\mathbb{K}, t, \lambda)$ for $\mathfrak{G}_1$ with $\mathbb{K} := (G_k, M_k, B, Y_k)_{k=0,\ldots,n}$ and $\lambda := \lambda_{\mathcal{G}} \dot\cup \lambda_{\mathcal{S}} \dot\cup \lambda_{\mathcal{C}} \dot\cup \lambda_{\mathcal{R}}$.

i.) Let be $v, w \in V_2$ with $v \in \sigma_2(w)$. We show the vertex condition for $\mathfrak{G}_2$ in $\mathbb{M}$, i.e. $\lambda_{\mathcal{G}}\rho_2(v) \in \text{Ext}\,(\pi_{\lambda_{\mathcal{S}}\rho_2(w)}\,\lambda_{\mathcal{C}}\kappa_2(v))$. As $\mathfrak{G}_2$ is valid in $\mathbb{M}^{\mathfrak{G}_1}$, the vertex condition in $\mathbb{M}^{\mathfrak{G}_1}$ and the definition of the incidence relation $Y_0^{\mathfrak{G}_1}$ implies $\iota_{\mathcal{G}}^{\mathfrak{G}_1}\rho_2(v) = \rho_2(v) \in \text{Ext}\,\pi_{\iota_{\mathcal{S}}^{\mathfrak{G}_1}\rho_2(w)}\,\iota_{\mathcal{C}}^{\mathfrak{G}_1}\kappa_2(v) = \text{Ext}\,\langle\kappa_2(v)\rangle_{\iota_{\mathcal{S}}^{\mathfrak{G}_1}\rho_2(w)} = \{g \in \mathcal{G} \mid (g, \kappa_2(v), \rho_2(w)) \in Y_0^{\mathfrak{G}_1}\} = \{\rho_1(x) \mid \exists\, y \in V_1 :\ x \in \sigma_1(y),\, \kappa_1(x) \leq_{\mathcal{C}} \kappa_2(v),\, \rho_1(y) = \rho_2(w)\}$.

Thus, for all vertices $v, w \in V_2$ with $v \in \sigma_2(w)$ there is a pair $x, y \in V_1$ with $x \in \sigma_1(y)$, $\rho_1(x) = \rho_2(v)$, $\kappa_1(x) \leq_{\mathcal{C}} \kappa_2(v)$ and $\rho_1(y) = \rho_2(w)$. From this, we can conclude (with the vertex condition for $\mathfrak{G}_1$ in $\mathbb{M}$ and the order preserving property of $\lambda_{\mathcal{C}}$) that $\lambda_{\mathcal{G}}\rho_2(v) = \lambda_{\mathcal{G}}\rho_1(x) \in \text{Ext}\,(\pi_{\lambda_{\mathcal{S}}\rho_1(y)}\,(\lambda_{\mathcal{C}}\kappa_1(x))) \subseteq \text{Ext}\,(\pi_{\lambda_{\mathcal{S}}\rho_2(w)}\,(\lambda_{\mathcal{C}}\kappa_2(v)))$.

ii.) Analogically, we show the vertex condition $\lambda_{\mathcal{G}}\rho_2(v) \in \text{Ext}\,(\pi_{\mathbb{S}_t}\,\lambda_{\mathcal{C}}\kappa_2(v))$ for $v \in V_2 \backslash \bigcup_{w \in V_2} \sigma_2(w)$. As $\mathfrak{G}_2$ is valid in $\mathbb{M}^{\mathfrak{G}_1}$, we have $\iota_{\mathcal{G}}^{\mathfrak{G}_1}\rho_2(v) = \rho_2(v) \in \text{Ext}\,\pi_{\mathbb{S}_u}\,\iota_{\mathcal{C}}^{\mathfrak{G}_1}\kappa_2(v) = \{g \in \mathcal{G} \mid \exists\, x \in V_1 \backslash \bigcup_{w \in V_1} \sigma_1(w) :\ \kappa_1(x) \leq_{\mathcal{C}} \kappa_2(v),\, \rho_1(x) = g\}$.

Thus, for every outer vertex $v \in V_2 \backslash \bigcup_{w \in V_2} \sigma_2(w)$ there is an outer vertex $x \in V_1 \backslash \bigcup_{w \in V_1} \sigma_1(w)$, with $\kappa_1(x) \leq_{\mathcal{C}} \kappa_2(v)$ and $\rho_1(x) = g$. From it, we can conclude (with the vertex condition for $\mathfrak{G}_1$ in $\mathbb{M}$ and the order preserving property of $\lambda_{\mathcal{C}}$) the assertion by $\lambda_{\mathcal{G}}\rho_2(v) = \lambda_{\mathcal{G}}\rho_1(x) \in \text{Ext}\,(\pi_{\mathbb{S}_t}\,\lambda_{\mathcal{C}}\kappa_1(x)) \subseteq \text{Ext}\,(\pi_{\mathbb{S}_t}\,\lambda_{\mathcal{C}}\kappa_2(v))$.

iii.) Let be $w \in V_2$ and $e \in E_2$ with $e^{\nu_2} \subseteq \sigma_2(w)$ and $\nu(e) \in V_2^k$. We show the edge edge condition $\lambda_{\mathcal{G}}\rho_2(e) \in \text{Ext}\,(\pi_{\lambda_{\mathcal{S}}\rho_2(w)}\,\lambda_{\mathcal{R}}\kappa_2(e))$ for $\mathfrak{G}_2$ in $\mathbb{M}$. For it, we conclude from the edge condition for $\mathfrak{G}_2$ in $\mathbb{M}^{\mathfrak{G}_1}$ and the definition of $\iota_{\mathcal{R}}^{\mathfrak{G}_1}$ and $Y_k^{\mathfrak{G}_1}$ that $\iota_{\mathcal{G}}^{\mathfrak{G}_1}\rho_2(e) = \rho_2(e) \in \text{Ext}\,\pi_{\iota_{\mathcal{S}}^{\mathfrak{G}_1}\rho_2(w)}\,\iota_{\mathcal{R}}^{\mathfrak{G}_1}\kappa_2(e) = \{(g_1, \ldots, g_k) \in \mathcal{G}^k \mid ((g_1, \ldots, g_k), \kappa_2(e), \rho_2(w)) \in Y_k^{\mathfrak{G}_1}\} = \{(g_1, \ldots, g_k) \in \mathcal{G}^k \mid \exists f \in E_1\colon \rho_1(f) = (g_1, \ldots, g_k),\, \kappa_1(f) \leq_{\mathcal{R}} \kappa_2(e),\, \rho_1(\sigma_1^{-1}(f^{\nu_1})) = \rho_2(w)\} = \{\rho_1(f) \mid f \in E_1\colon \kappa_1(f) \leq_{\mathcal{R}} \kappa_2(e),\, \rho_1(\sigma_1^{-1}(f^{\nu_1})) = \rho_2(w)\}$. Then, we proceed as for the vertex condition, and finally, analogically for condition iv.). □

Precision for Inference Rules in Definition 12. Let $\mathfrak{G} := (V, E, \nu, \sigma, \kappa, \rho)$ and $\mathfrak{G}'$ be two nested concept graphs over the alphabet $\mathcal{A} := (\mathcal{G}, \mathcal{C}, \mathcal{R})$. We define the following rules:

1. If $\mathfrak{G}' := (V', E', \nu', \sigma', \kappa', \rho')$ satisfies $V' = (V \backslash W_v) \dot\cup (W_v \times \{1,2\})$ for a $v \in V$ and $W_v := \bigcup_{m\in\mathbb{N}_0} \sigma^m(v)$; $E' = (E \backslash \bigcup_{m\in\mathbb{N}_0} E_{\sigma^m(v)}) \dot\cup E^v \dot\cup (\bigcup_{m\in\mathbb{N}} E_{\sigma^m(v)} \times \{1,2\})$ with $E_M := \{e \in E \mid \exists\, i = 1,\ldots,n : \nu(e)|_i \in M\}$ and $E^v := \{(e,\delta) \mid e \in \{e \in E \mid \exists\, i = 1,\ldots,k : \nu(e)\big|_i = v\}, \delta \in \{1,2\}^{[e,v]}\}$ (where $[e,v] := \{i \in \{1,\ldots,k\} \mid \nu(e)|_i = v\}$); $\nu'|_i : E' \to V'$ with $\nu'|_i(e) := \nu(e)|_i$ for all $e \in E \backslash \bigcup_{m\in\mathbb{N}_0} E_{\sigma^m(v)}$, $\nu'|_i(e,j) := (\nu(e)|_i, j)$ for all $(e,j) \in \bigcup_{m\in\mathbb{N}} E_{\sigma^m(v)} \times \{1,2\}$ and $\nu'|_i(e,\delta) := \nu(e)|_i$ if $i \notin [e,v]$ and $\nu'|_i(e,\delta) := (v, \delta(i))$ if $i \in [e,v]$ for all $(e,\delta) \in E^v$; $\sigma' : V' \to \mathfrak{P}(V')$ with $\sigma'(w) := \sigma(w)$ for all $w \in \mathrm{dom}\ \sigma \backslash (W_v \dot\cup \sigma^{-1}(v))$, $\sigma'(w,j) := \sigma(w) \times \{j\}$ for all $(w,j) \in W_v \times \{1,2\}$ and $\sigma'(\sigma^{-1}(v)) := (\sigma(\sigma^{-1}(v)) \backslash \{v\}) \dot\cup \{(v,1), (v,2)\}$; $\kappa' : V' \cup E' \to \mathcal{C} \cup \mathcal{R}$ with $\kappa'(x) := \kappa(x)$ for all $x \in V \backslash W_v \cup E \backslash \bigcup_{m\in\mathbb{N}_0} E_{\sigma^m(v)}$ and $\kappa'(x,l) := \kappa(x)$ otherwise; $\rho'|_{V\backslash W_v} = \rho|_{V\backslash W_v}$ and $\rho'(w,j) = \rho(w)$ for all $(w,j) \in W_v \times \{1,2\}$ then we say that $\mathfrak{G}'$ is derived from $\mathfrak{G}$ by *doubling the vertex* $v \in V$.
2. If $\mathfrak{G}' = (V', E, \sigma|_{V'}, \nu, \kappa|_{V' \cup E}, \rho|_{V'})$ with $V' = V \backslash \{v\}$ for a $v \in V \backslash \mathrm{dom}\ \sigma$ with $\{e \in E \mid \exists\, i = 1,\ldots,k : \nu(e)\big|_i = v\} = \emptyset$, then $\mathfrak{G}'$ is derived from $\mathfrak{G}$ by *deleting the isolated, non-complex vertex* $v \in V$
3. If $\mathfrak{G}' = (V, E', \nu', \sigma, \kappa', \rho)$ satisfies $E' = E \backslash \{e\} \dot\cup \{(e,1),(e,2)\}$ for an $e \in E$, $\nu'|_{E\backslash\{e\}} = \nu|_{E\backslash\{e\}}$, $\nu'(e,j) = \nu(e)$ for $j = 1,2$, $\kappa'|_{V\cup(E\backslash\{e\})} = \kappa|_{V\cup(E\backslash\{e\})}$ and $\kappa'(e,j) = \kappa(e)$ for $j = 1,2$, then, we say that $\mathfrak{G}'$ is derived from $\mathfrak{G}$ by *doubling the edge* $e \in E$.
4. If we have $\mathfrak{G}' = (V, E\backslash\{e\}, \nu|_{E\backslash\{e\}}, \sigma, \kappa|_{V\cup E\backslash\{e\}}, \rho)$ for a $e \in E$, then we say that $\mathfrak{G}'$ is derived from $\mathfrak{G}$ by *deleting the edge* $e \in E$.
5. If we have $\mathfrak{G}' = (V, E, \nu, \kappa', \sigma, \rho)$ and a $v \in V$ with $\kappa'|_{V\backslash\{v\}\cup E} = \kappa|_{V\backslash\{v\}\cup E}$, and if there is a $w \in V$ such that $\rho(w) = \rho(v)$, $\rho(\sigma^{-1}(w)) = \rho(\sigma^{-1}(v))$ and $\kappa(w) \leq_{\mathcal{C}} \kappa'(v)$, then we say that $\mathfrak{G}'$ is derived from $\mathfrak{G}$ by *exchanging the concept name* $\kappa(v)$.
6. If $\mathfrak{G}' = (V, E, \nu, \sigma, \kappa', \rho)$ with $\kappa'|_{V\cup E\backslash\{e\}} = \kappa|_{V\cup E\backslash\{e\}}$, and if there is an $f \in E$ such that $\rho(f) = \rho(e)$, $\rho(\sigma^{-1}(f^\nu)) = \rho(\sigma^{-1}(e^\nu))$ and $\kappa(f) \leq_{\mathcal{R}} \kappa'(e)$, then we say that $\mathfrak{G}'$ is derived from $\mathfrak{G}$ by *exchanging the relation names* $\kappa(e)$.
7. If two vertices $v, w \in V$ satisfy $\rho(v) = \rho(w)$ and $\sigma^{-1}(v) = \sigma^{-1}(w)$, and if $\mathfrak{G}' = (V', E', \nu', \sigma', \kappa', \rho')$ satisfies $V' = V \backslash \{v,w\} \dot\cup \{v \vee w\}$, $E' = E$; all $e \in E$ and all $i \in \{1,\ldots,k\}$ satisfy $\nu'(e)|_i = v \vee w$ if $\nu(e)|_i \in \{v,w\}$ and otherwise $\nu'(e)|_i = \nu(e)|_i$; $\sigma'|_{\mathrm{dom}\ \sigma\backslash\{\sigma^{-1}(v),v,w\}} = \sigma|_{\mathrm{dom}\ \sigma\backslash\{\sigma^{-1}(v),v,w\}}$, $\sigma'(\sigma^{-1}(v)) = (\sigma(\sigma^{-1}(v)) \backslash \{v,w\}) \dot\cup \{v \vee w\}$ and $\sigma'(v \vee w) = \sigma(v) \cup \sigma(w)$; $\kappa'|_{V\backslash\{v,w\}\cup E} = \kappa|_{V\backslash\{v,w\}\cup E}$ and $\kappa'(v \vee w) = c$ for a $c \in \mathcal{C}$ with $\kappa(v) \leq_{\mathcal{C}} c$ and $\kappa(w) \leq_{\mathcal{C}} c$; $\rho'|_{V\backslash\{v,w\}} = \rho|_{V\backslash\{v,w\}}$ and $\rho'(v \vee w) = \rho(v)$. then we say that $\mathfrak{G}'$ is derived from $\mathfrak{G}$ by *joining the vertices* v *and* w *with equal references in the same nesting.*
8. Let $\mathfrak{H} := (W, F, \nu_\mathfrak{H}, \sigma_\mathfrak{H}, \kappa_\mathfrak{H}, \rho_\mathfrak{H})$ be a nested sub graph of $\mathfrak{G}$. That means, it satisfies $W \subseteq V$ and $F \subseteq E$; the mappings ν, κ and ρ are continuations of $\nu_\mathfrak{H}$, $\kappa_\mathfrak{H}$ and $\rho_\mathfrak{H}$; all $w \in dom\, \sigma_\mathfrak{H}$ satisfy $\sigma_\mathfrak{H}(w) = \sigma(w) \cap W$ and $\sigma^{-1}(W \backslash \bigcup_{v\in W} \sigma_\mathfrak{H}(v)) = \emptyset$ or $\sigma^{-1}(W \backslash \bigcup_{v\in W} \sigma_\mathfrak{H}(v)) = \{v\}$ for a $v \in V \backslash W$.
If we have $\mathfrak{G}' := (V \dot\cup W, E \dot\cup F, \nu \dot\cup \nu_\mathfrak{H}, \sigma \dot\cup \sigma_\mathfrak{H}, \kappa \dot\cup \kappa_\mathfrak{H}, \rho \dot\cup \rho_\mathfrak{H})$, and $\emptyset = \sigma^{-1}(W \backslash \bigcup_{v\in W} \sigma_\mathfrak{H}(v))$, then we say that $\mathfrak{G}'$ is derived from $\mathfrak{G}$ by *copying the sub graph* $\mathfrak{H}$ *into the outer nesting of* $\mathfrak{G}$.

If there is a $v \in V \backslash W$ with $\sigma^{-1}(W \backslash \bigcup_{v \in W} \sigma_{\mathfrak{H}}(v)) = \{v\}$ and if we have $\mathfrak{G}' = (V \dot\cup W,\ E \dot\cup F, \nu \dot\cup \nu_{\mathfrak{H}}, \sigma', \kappa \dot\cup \kappa_{\mathfrak{H}}, \rho \dot\cup \rho_{\mathfrak{H}})$ with $\sigma'|_{V \backslash \{v\}} = \sigma|_{V \backslash \{v\}}$, $\sigma'(v) = \sigma(v) \dot\cup W \backslash \bigcup_{v \in W} \sigma_{\mathfrak{H}}(v)$, and $\sigma'|_{\bigcup_{v \in W} \sigma_{\mathfrak{H}}(v)} = \sigma_{\mathfrak{H}}$, then we say that $\mathfrak{G}'$ is derived from $\mathfrak{G}$ by *copying the sub graph $\mathfrak{H}$ into the nesting of v*.

Idea for the Proof of Completeness (Proposition 2). For the proof we consider the elementary modules of nested concept graphs: The sub graphs that are generated by one edge or one vertex, that means the sub graph consisting of all vertices in which the generating edge or vertex is successively nested. Those elementary modules are important for the inferences on concept graphs: A nested concept graph does not entail all its sub graphs because if there is an outer nesting missing, the sub graph is not "in the right situation" anymore. Only the sub graphs being generated by one vertex or edge are entailed by $\mathfrak{G}$ (because they can be derived by deleting all other edges and vertices).

In order to prove completeness, we derive, for each edge and each vertex of $\mathfrak{G}_2$ the corresponding generated sub graph of $\mathfrak{G}_1$. That means, e.g. for each edge $f \in E_2$ there is an edge $e_f \in E_1$ with $\rho_1(e_f) = \rho_2(f), \kappa_1(e_f) \leq_{\mathcal{R}} \kappa_2(f)$ and $\rho_1(\sigma_1^{-1}(e_f^{\nu_1})) = \rho_2(\sigma_2^{-1}(f^{\nu_2}))$ (it exists because of the edge condition for $\mathfrak{G}_2$ in the standard model of $\mathfrak{G}_1$). This edge e_f generates a sub graph with a relation name $\kappa_1(e_f)$ that can be generalized into $\kappa_2(f)$. Then, the sub graph's inner edge and its reference of the inner nesting equals those of f, but not the other nestings and labels of vertices).

Having derived these modules for all edges and vertices, they can be joined successively into a concept graph which is isomorphic to $\mathfrak{G}_2$. For it, the modules are copied into the right nesting and then they are joined. Finally, all superfluous modules are eliminated. (For a detailed formal proof see [Pr98a]). □

References

BP83. J. Barwise, J. Perry: Situations and Attitudes, MIT Press, Cambridge 1983.

CM97. M. Chein, M.-L. Mugnier: Positive Nested Conceptual Graphs, in: D. Lukose et al. (eds.): Conceptual Structures: Fulfilling Peirce's Dream, LNAI 1257, Springer Verlag, Berlin–New York 1997, 95–109.

CMS98. M. Chein, M.-L. Mugnier, G. Simonet: Nested Graphs: A Graph-based Knowledge Representation Model with FOL Semantics, Rapport de Recherche, LIRMM, Université Montpellier II, 1998.

De91. K. Devlin: Logic and Information, Cambridge University Press, 1991.

LW95. F. Lehmann, R. Wille: A triadic approach to formal concept analysis, in: G. Ellis et al. (Hrsg.): Conceptual Structures: Applications, Implementations and Theory. Springer Verlag, Berlin–New York 1995, 32–43.

McC93. J. McCarthy: Notes on Formalizing Context, in: Proceedings of the IJCAI 1993, 555–560.

MC96. M.-L. Mugnier, M. Chein: Représenter des Connaissances et Raisonner avec des Graphes, Revue d'Intelligence Artificielle 10, 1996, 7–56.

PMC95. A. Preller, M.-L. Mugnier, M. Chein: A Logic for Nested Graphs, Research Report 95038, LIRMM, Université Montpellier II, 1995.

Pr98a. S. Prediger: Kontextuelle Urteilslogik mit Begriffsgraphen. Ein Beitrag zur Restrukturierung der mathematischen Logik, Dissertation, FB Mathematik, TU Darmstadt, Shaker Verlag, Aachen 1998.

Pr98b. S. Prediger: Simple Concept Graphs: A Logic Approach, in: M.-L. Mugnier / M. Chein (eds.): Conceptual Structures: Theory, Tools and Application, LNAI 1453, Springer Verlag, Berlin–Heidelberg 1998.

PW99. S. Prediger / R. Wille: The Lattice of Concept Graphs of a Relationally Scaled Context, in: W. Tepfenhart / W. Cyre (eds.): Conceptual Structures: Standards and Practices, LNAI 1640, Springer Verlag, Berlin – Heidelberg – New York 1999, 401-414.

Si96. G. Simonet: Une sémantique logique pour les graphes emboités, Research Report 96047, LIRMM, Université Montpellier II, 1996.

So84. J. F. Sowa: Conceptual Structures: Information Processing in Mind and Machine, Adison-Wesley, Reading 1984.

So97. J. F. Sowa: Peircean Foundation for a Theory of Context, in: D. Lukose et al. (eds.): Conceptual Structures: Fulfilling Peirce's Dream, LNAI 1257, Springer Verlag, Berlin–New York 1997, 41–64.

Wi95. R. Wille: The Basic Theorem of Triadic Concept Analysis, in: Order 12 (1995), 149–158.

Wi97. R. Wille: Conceptual Graphs and Formal Concept Analysis, in: D. Lukose et al. (eds.): Conceptual Structures: Fulfilling Peirce's Dream, LNAI 1257, Springer Verlag, Berlin–New York 1997, 290–303.

Wi98. R. Wille: Triadic Concept Graphs, in: M.-L. Mugnier / M. Chein (eds.): Conceptual Structures: Theory, Tools and Application, LNAI 1453, Springer Verlag, Berlin–Heidelberg 1998, 198–208.

Negations in Simple Concept Graphs

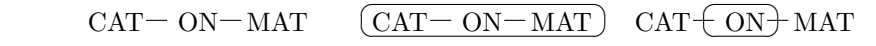
Frithjof Dau

Technische Universität Darmstadt, Fachbereich Mathematik
Schloßgartenstr. 7, D-64289 Darmstadt, dau@mathematik.tu-darmstadt.de

Abstract. The aim of this paper is to mathematically introduce negation to concept graphs (which are a mathematical modification of conceptual graphs) as a well-defined syntactical construct. First off, we discuss some questions which arise when negations for conceptual graphs are defined. In our view, a solution for these questions is to express negations by cuts in the sense of Peirce's theory of existential graphs. A set-theoretical semantics for (nonexistential) concept graphs with cuts is developed in the framework of contextual logic. A modification of Peirce's alpha-calculus, which is sound and complete, is presented.

1 Motivation

Conceptual graphs are based on the *existential graphs* of Charles Sanders Peirce. These graphs consist of lines called *lines of identities*, predicate names of arbitrary arity and ovals around subgraphs which are used to negate the enclosed subgraph. The following three examples are well known:



The meanings of these graphs are: 'a cat is on a mat', 'no cat is on any mat' and 'there is a cat and there is a a mat such that the cat is not on the mat'.

As Peirce says, "That a proposition is false is a *logical* statement about it, and therefore in a logical system deserves special treatment." ([Pe98]). The graphical element *oval* which Peirce used to negate its enclosure has been transferred to context boxes in conceptual graphs. These boxes are used to express that some information is valid in specific contexts or situations. Hence, the character of negation as logical operator in existental graphs changed to a metalevel character in conceptual graphs. Of course, in knowledge representation and natural language, negations are unavoidable. So the feature to express negations is desirable in concept graphs.

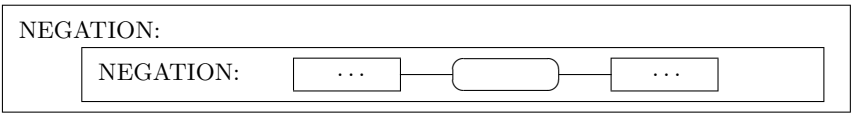
To handle negation in concept graphs, we need to achieve the following aims: For the mathematical treatment, formation rules for the well-formed formulas must exist that can express negations, and negation has to be covered by rules of inference. To do this in the spirit of Peirce, the semantics of negation has to be intelligible, and the graphical representation of negations must be easily readable and intuitively understandable (which has been an important goal in the theory of conceptual graphs from the very beginning, too).

B. Ganter and G.W. Mineau (Eds.): ICCS 2000, LNAI 1867, pp. 263–276, 2000.
© Springer-Verlag Berlin Heidelberg 2000

Negations occur in several approaches for conceptual graphs. Why we do not adopt and mathematizise one of these approaches shall be explained in the rest of this section.

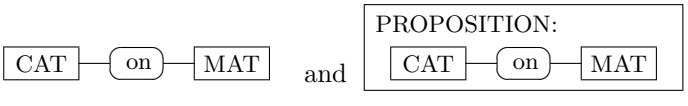
In order to handle negations, a specific syntactical element of the well-formed formulas has to be declared to express them. For this purpose, the standard approach use a context box of type `Proposition` which is linked to a unary relation of type `neg` ([So99]). Sometimes, these special context boxes are abbreviated by context boxes of type `Negation` (e.g. [So00]) or by drawing a simple rectangle with the mathematical negation symbol ¬ ([So99]). Some approaches use these rectangles without declaring whether the box is a specific syntactical element or just an abbreviation for context boxes of a specific type (e.g. [We95]). But both a calculus and a translation of conceptual graphs into other formal languages (like the translation to first order logic with the Φ-operator), have to respect the logical role of negation.

So, if negation is expressed just by special context boxes, any calculus and any tranlation has to treat these special context boxes differently from all other context boxes. For example, if negation is expressed by context boxes of type `Negation`, a calculus should allow the nested boxes in the following graph to be erased (and vice versa, to be introduced again):



This seems to be not possible in any calculus which does not treat the negation boxes separately (like the calculus of Prediger ([Pr98b]) or any calculus which is based on projections).

If negation is expressed with contexts of type `Proposition`, linked to a unary relation `neg`, another difficulty appears. This shall be shown by the following two conceptual graphs:

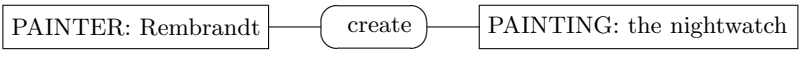


The first graph is well known: Its meaning is 'a cat is on a mat'. In particular, the graph claims to be true. The meaning of the second graph is, strictly speaking, 'there exists a proposition, which states that a cat is on a mat' ([So99]), and therefore different to the meaning of the first graph. Indeed; in none of the common calculuses, one graph can be derived from the other one. Hence it is problematic to express the negation of the first graph by the second graph with a relation `neg`.

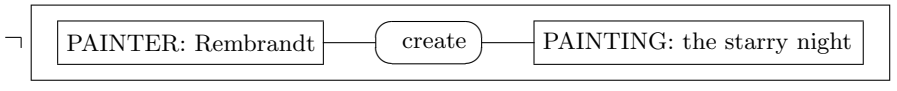
To summarize: It seems to be difficult to introduce negation as a special context box. These context boxes have to be treated differently to other boxes in the calculus and in any translation from conceptual graphs to other formal languages (like the operator Φ). This yields the following conclusion: For the mathematical treatment of negation in concept graphs, in the definition of their

well-formed formulas there should be a specific syntactical element which is used to express negation.

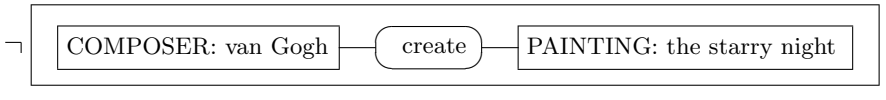
The next step has to clarify the semantics, i.e. the meaning, of negation. To prepare this, we will discuss a small example. Consider the true proposition 'the painter Rembrandt created the painting 'the nightwatch", which can be translated to the following conceptual graph:



This graph represents not only the information that Rembrandt created 'the nightwatch', but also that Rembrandt is a painter and 'the nightwatch' is a painting. Now, consider the painting 'a starry night' instead of 'the nightwatch'. This painting was created by van Gogh, so the proposition 'the painter Rembrandt did not create the painting 'the starry night" is true. How can this proposition be transformed to a conceptual graph? The following graph is a first attempt:



This graph is *not* the translation of the former proposition: In the proposition, only the verb 'to create' is negated, but in the graph, the negation box also encloses the information that Rembrandt is a painter and 'the starry night' is a painting. The information in the concept boxes can fail, too, as can be seen in the following graph:



This graph is true although van Gogh did create 'the starry night'. In particular, this graph should not be read as

The composer Van Gogh did not *create the painting 'a starry night'.*

But this understanding is suggested when Sowa in [So00] says, that the meaning of the graph `[Negation: [Cat: Yoyo]→(On)→[Mat]]` is 'the graph denies that the cat Yoyo is on a mat'. Now, the goal is to negate only the verb 'to create' in the false proposition 'the painter Rembrandt created the painting 'the starry night". This problem has already been addressed, one approach for its solution is the following graph:

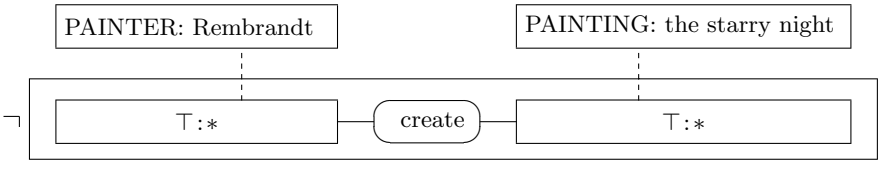


Fig. 1. CG for 'the painter Rembrandt did not create the painting 'a starry night"

Indeed, this graph expresses the proposition 'the painter Rembrandt did not create the painting 'the starry night'. But obviously, the aim of making conceptual graphs easily readable and intuitively understandable is not fulfilled.

Expressing identity in conceptual graphs with coreference-links or coreference-sets leads to another class of difficulties. In particular, the meaning of coreference-links connected to a concept box within a negation is not straightforward. This shall be explained next:

In conceptual graphs, coreference-links (which are used to express coreference-sets ([So99])) are used to express the identity of two entities: "Two concepts that refer to the same individual are coreferent. [...] To show that they are coreferent, they are connected with a dotted line, called a *coreference link*." ([So92]). Consider the following graph:

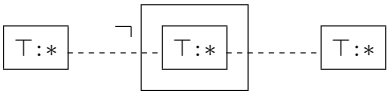
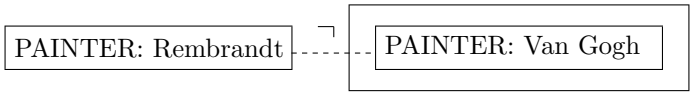


Fig. 2. conceptual graph for $\exists x \exists y . x \neq y$

According to Sowa ([So00]), the operator Φ translates this graph into the first order logic formula $\exists x \exists y \neg \exists z (z = x \wedge z = y)$, which is equivalent to $\exists x \exists y . x \neq y$. In particular, the three concept boxes cannot refer to the same individual. Note that Φ assigns different variables to the generic markers of different concept boxes, even if they are connected with a coreference-link. These variables are explicitly set to be equal in the formula, and they are equated *inside the negated part* of the formula. But since the links in the graph looks symmetric, it is not clear to a reader why the equating in the formula is placed *inside* and not *outside* of the negated subformula. This ambiguity can be seen even better in the following example:

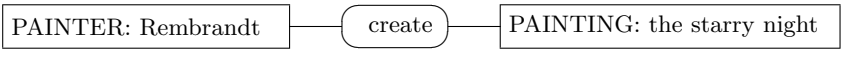


If Φ translates this graph to $PAINTER(R) \wedge \neg(R = VG \wedge PAINTER(VG))$, the resulting formula is true, but if Φ translates this graph to $PAINTER(R) \wedge Rembrandt = VG \wedge \neg(PAINTER(VG))$ the resulting formula is not true (the names in the formulas are abbreviated by R and VG). Hence, in order to understand the right meaning of this graph, the reader must have in mind the implicit agreement that equality is always placed in the inner context.

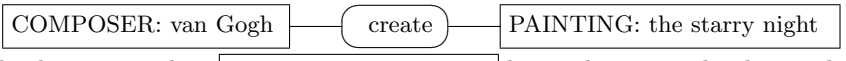
If we accept this meaning of corerence links, the next step is is to make clear which syntactical element in the well-formed formulas is used for them, and how they are handled by a calculus. In the abstract syntax of conceptual graphs ([So99]), coreference-links are generalized to coreference-sets. For example, Figure 2 has two coreference sets which are represented by coreference-links. Coreference-sets are sufficient to handle coreference in conceptual graphs with negations. But still rules are needed that treat these sets in a sound and complete way (for example, there have to be rules which allow a link to be drawn or erased from a concept box to itself). Some calculuses lack rules like this.

To summarize again: Introducing coreference sets to express identity may lead to misunderstandings of their meaning and to gaps in their syntactical implementation.

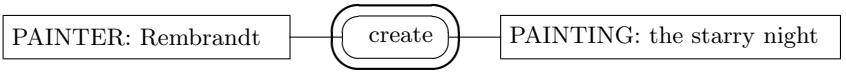
To cope with all the mentioned problems, we suggest the following: First, negations should be introduced as a new syntactical element, namely the ovals of Peirce, which can be drawn around arbitrary parts of a conceptual graph. To distinguish these ovals from the ovals which are drawn around relation names, we propose drawing them in bold. For example, in

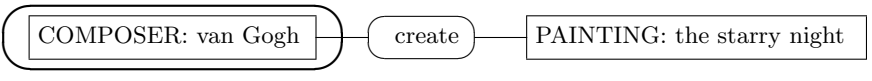


only the relation `create` has to be negated, and in



only the concept box COMPOSER: van Gogh has to be negated. The resulting conceptual graphs are:





Because the present interpretation of coreference-links is not intuitive in some sense, we suggest to introduce a special binary relation *id* (as in first order logic). The advantages of this approach are

1. *id* can be trated like other relations and
2. the identity can be negated without loss of readability.

In our view this yields a more understandable notion of identity (as understood in mathematics). For example, the meaning of the graph

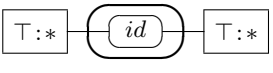
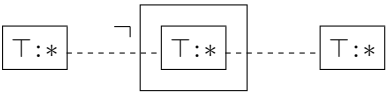


Fig. 3. CG with negation ovals for $\exists x \exists y . x \neq y$

is 'there exist at least two things'. Thus, it has the same meaning as the graph in Figure 2, but is much simpler. Furthermore, it shows that *id* is a proper syntactical extension and not a direct mathematization of coreference links.

Since the syntactical elements which allow negations and identity are extended and in this approach, every conceptual graph with negation boxes and coreference-links can be translated into a concept graph with negation ovals and the relation *id*. Of course, negation boxes (for example, context boxes of type `Negation`) are translated to negation ovals. A coreference-link between two context boxes is translated into a relation *id* between the boxes such that the relation node *id* is placed in the negation oval of the dominated context box. We will exemplify this with the following: The conceptual graph



is translated to the following conceptual graph with negation ovals and the relation *id*:

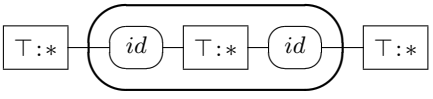


Fig. 4. another CG with negation ovals for $\exists x \exists y . x \neq y$

This graph can be transformed to the graph in Figure 3, which has the same meaning, but looks much simpler. On the other hand, concept graphs with negation ovals and the relation *id* can be translated to graphs with negation boxes and coreference-links. But because negation ovals need not include subgraphs, but arbitrary subsets of concept nodes and relations nodes, this translation is more complicated than the translation in the other direction. For example: Before the graph in Figure 3 can be translated, it has to be transformed into the graph in Figure 4.

The approach we present here is closely related to the original ideas of Peirce. It is easier to mathematize than approaches based on concept boxes of a specific type. In this paper, our approach shall be elaborated for simple concept graphs without generic markers, but with negation ovals and the relation *id*. In particular, the syntax for these graphs is defined, an extensional semantics for these graphs is introduced (which is based on power context families), and a sound and complete calculus is presented. Furthermore, this approach allows to define mathematically the operator Φ on simple concept graphs (which maps graphs to first order logic formulas) and its inverse operator Ψ (which maps first order logic formulas to graphs) such that both respect the (syntactical or semantic) entailment relation on graphs and formulas, respectively. In particular, the expressiveness of simple graphs and first order logic formulas is the same. This will be elaborated in a work which is in progress now.

2 Basic Definitions

Simple concept graphs are introduced by Prediger in [Pr98b] as mathematically defined syntactical constructs. We take into account her approach and extended it to include the possibily to express negations by using cuts and the possibility to express identity by using a special binary relation *id*.

First we have to start with ordered sets of names for objects, names and relations. These orders represent the conceptual ontology of the domain we consider.

Definition 1. *An alphabet of conceptual graphs is a tripel* $\mathcal{A} := (\mathcal{G}, \mathcal{C}, \mathcal{R})$ *such that*

- $\mathcal{G}$ *is a finite set whose elements are called object names*
- $(\mathcal{C}, \leq_{\mathcal{C}})$ *is a finite ordered set with a greatest element* $\top$ *whose elements are called concept names*
- $(\mathcal{R}, \leq_{\mathcal{R}})$ *is a union of finite ordered sets* $(\mathcal{R}_k, \leq_{\mathcal{R}_k})$, $k = 1, \ldots, n$ *(for an* $n \in \mathbb{N}$ *with* $n \geq 1$*) whose elements are called relation names. Let* $id \in \mathcal{R}_2$.

Now we can define the underlying structures of concept graphs with cuts. This definition extends the definition of directed multi-hypergraphs given in [Pr98b] by cuts, so that negations can be expressed.

Definition 2. *A directed multi-hypergraph with cuts (of type* n*) is a structure* $(V, E, \nu, Cut, area)$ *such that*

- V *and* E *are finite sets whose elements are called vertices and edges, respectively,*
- $\nu : E \to \bigcup_{k=1}^{n} V^k$ *(for a* $n \in \mathbb{N}, n \geq 1$*) is a mapping,*
- Cut *is a finite set whose elements are called cuts and*
- $area : Cut \to \mathfrak{P}(V \cup E \cup Cut)$ *is a mapping such that* $c \notin area(c)$ *for each* $c \in Cut$ *and, for two cuts* c_1, c_2 *with* $c_1 \neq c_2$*, exactly one of the following conditions holds:*
  - *i)* $\{c_1\} \cup area(c_1) \subseteq area(c_2)$,
  - *ii)* $\{c_2\} \cup area(c_2) \subseteq area(c_1)$,
  - *iii)* $(\{c_1\} \cup area(c_1)) \cap (\{c_2\} \cup area(c_2)) = \emptyset$.

For an edge $e \in E$ *with* $\nu(e) = (v_1, \ldots, v_k)$ *we define* $|e| := k$ *and* $\nu(e)\big|_i := v_i$. *For each* $v \in V$*, let* $E_v := \{e \in E \mid \exists\, i\colon \nu(e)\big|_i = v\}$*, and analogously for each* $e \in E$*, let* $V_e := \{v \in V \mid \exists\, i\colon \nu(e)\big|_i = v\}$. *If it cannot be misunderstood, we write* $e\big|_i$ *instead of* $\nu(e)\big|_i$.

The notion of *cuts* and *areas* is closely related to the ideas of Peirce, as they are described in the work of Roberts (see [Ro73]). Peirce negated parts of an existential graph just by drawing an oval around it. This oval (more exactly just the line which is drawn on the sheet of assertion) is called a *cut*. In particular, a cut is not a graph. The space within a cut is called its *close* or *area*. So the area of a cut c contains vertices, edges and other cuts, even if they are deeper nested inside other cuts, but not the cut c itself. All the edges, vertices and cuts in the area of c are said to be *enclosed by* c.

Cuts do not intersect each other by the definition of Peirce. So for two different cuts c_1, c_2, exactly one of the following cases occurs:

- c_1 and its area is entirely enclosed by c_2,
- c_2 and its area is entirely enclosed by c_1,
- c_1 and its area and c_2 and its area have nothing in common.

Obvioulsy, these three cases coincide with the three conditions for the mapping *area* in Definition 2. Now, let us first mention some simple properties for the mapping *area* which can be shown easily:

- $c_1 \neq c_2 \wedge area(c_1) = area(c_2) \Longrightarrow area(c_1) = area(c_2) = \emptyset$
- $\emptyset \subsetneq area(c_1) \subsetneq area(c_2) \Longrightarrow c_1 \in area(c_2)$
- $c_1 \in area(c_2) \Longrightarrow area(c_1) \subseteq area(c_2)$

In many cases it makes sense to treat the outermost context, the sheet of assertion, as an (additional) cut. If we abbreviate the sheet of assertion by $\top$, we immediately come to the following definition:

Definition 3. *If Cut is a set of cuts of a directed multi-hypergraph with cuts and if area is the appropriate mapping, then let $Cut^\top := Cut \dot\cup \{\top\}$ and $area(\top) := V \cup E \cup Cut$.*

It is easy to see that this extension still satisfies the conditions for the mapping *area* which are given in Definition 2. This means that the properties we have just shown for Cut hold for $Cut^\top$, too.

By $c_1 \leq c_2 :\Longleftrightarrow c_1 \in area(c_2)$ a canonical ordering on $Cut^\top$, which is a tree with $\top$ as greatest element, is defined. This can be verified with the properties for the mapping *area*.

Obviously, each edge and vertex is enclosed *directly* (and not deeper nested) in a uniquely given cut c. For the further work, the notion of a *subgraph* is needed. It seems to be evident that a subgraph is enclosed directly in a uniquely given cut c, too. The notions of being *directly enclosed* and *subgraph* shall become precise through the following definition:

Definition 4. *Let $\mathfrak{G} = (V, E, \nu, Cut, area)$ be a directed multi-hypergraph with cuts.*

- *For each $k \in V \cup E \cup Cut$ we define*

$$cut(k) := min\{c \in Cut^\top \mid k \in area(c)\}$$

  $cut(k)$ is called the cut of k and $cut(k)$ is said to enclose directly the vertex (the edge, the cut) k.
- *The graph $\mathfrak{G}' = (V', E', \nu', Cut', area')$ is called a subgraph of $\mathfrak{G}$ in the cut c if $c \in Cut^\top$ is the smallest cut such that the following conditions hold:*
  - *$V' \subseteq V, E' \subseteq E, Cut' \subseteq Cut$ and the mappings ν' and $area'$ are just the restrictions of ν and $area$ to E' resp. Cut' (and are therefore well defined),*
  - *$area(c') \subseteq V' \cup E' \cup Cut'$ for each $c' \in Cut'$,*
  - *$cut(k') \in Cut' \cup \{c\}$ for each $k' \in V' \cup E' \cup Cut'$,*
  - *$v \in V'$ for each edge $e' \in E'$ and every vertex $v \in V_e$.*

  We write: $\mathfrak{G}' \subseteq \mathfrak{G}$ and $cut(\mathfrak{G}') = c$.

Note, that for each vertex (or edge, cut or subgraph), the set of all cuts containing the vertex forms a chain. If the number of cuts enclosing the vertex is even, the edge is said to be *evenly enclosed*, and analogously, if the number is odd, the vertex is said to be *oddly enclosed*. More formally:

Definition 5. *Let $\mathfrak{G} = (V, E, \nu, Cut, area)$ be a directed multi-hypergraph with cuts, let k be a subgraph or an element of $V \cup E \cup Cut^\top$. Let n be the number of cuts which enclose k ($n := |\{c \in Cut \mid c \in area(c)\}|$). If n is even, k is said to be evenly enclosed, otherwise k is said to be oddly enclosed. An evenly enclosed cut is called positive, an oddly enclosed cut is called negative.*

Now, the structure of simple concept graphs with cuts is derived from the structure of directed multi-hypergraphs with cuts by additionally labeling the vertices and edges with concept names and relation names, respectively, and by assigning a reference to each vertex. In particular all definitions concerning directed multi-hypergraphs with cuts can be transferred to concept graphs. So in the following we will deal with subgraphs of concept graphs etc.

Definition 6. *A (nonexistential) simple concept graph with cuts over the alphabet $\mathcal{A}$ is a structure $\mathfrak{G} := (V, E, \nu, Cut, area, \kappa, \rho)$, where*

- *$(V, E, \nu, Cut, area)$ is a directed multi-hypergraph with cuts*
- *$\kappa : V \cup E \to \mathcal{C} \cup \mathcal{R}$ is a mapping such that $\kappa(V) \subseteq \mathcal{C}$ and $\kappa(E) \subseteq \mathcal{R}$, and all $e \in E$ with $v(e) = (v_1, \ldots, v_k)$ satisfy $\kappa(e) \in \mathcal{R}_k$*
- *$\rho : V \to \mathcal{G}$ is a mapping*

It is not clear what a graph containing vertices with more than one object, enclosed by a cut, means, and this might lead to misunderstandings. For this reason, the mapping ρ maps vertices only to elements of $\mathcal{G}$, not to subsets of $\mathcal{G}$ (in contrast to the definition of Prediger in [Pr98b]). Furthermore, ρ can be naturally extended to the edges: If e is an edge with $v(e) = (v_1, \ldots, v_k)$, let $\rho(e) := (\rho(v_1), \ldots, \rho(v_n))$.

3 Semantics

Usually, a semantics for conceptual graphs is given by a translation of graphs into formulas of first order logic, hence into formulas of another syntactically given structure. In Prediger (cf. [Pr98a], [Pr98b]), a different approach is presented. There, an extensional semantics which is based on power context families as model structures is introduced. The motivation for this *contextual semantics* can be read in [Pr98a]. With this semantics, Prediger develops a semantical entailment relation between concept graphs, and a sound and complete calculus for this entailment relation is presented. Now this approach shall be extended to concept graphs with cuts.

In concept graphs without cuts, only the conjunction of positive information can be expressed. For this reason it was possible for Prediger to construct for each concept graph a standard model in which all the information of the concept graph is encoded. Standard models have been an additional possibility (besides the entailment relation and the calculus) for doing reasoning with concept graph. If negations are used, one can express with concept graphs the disjunction of pieces of information. But disjunction of information can not be canonically encoded in standard models. Thus if we introduce negations to concept graphs, unfortunately the construction of standard models has to be dropped.

Now, let us recall the basic definitions of Prediger.

Definition 7. *A power context family $\mathbb{K} := (\mathbb{K}_0, \ldots, \mathbb{K}_n)$ of type n (for an $n \in \mathbb{N}$) is a family of contexts $\mathbb{K}_k := (G_k, M_k, I_k)$ that satisfies $G_k \subseteq (G_0)^k$ for each $k = 1, \ldots, n$. Then we write $\mathbb{K} := (G_k, M_k, I_k)_{k=0,\ldots,n}$. The elements of the set $\mathfrak{R}_{\mathbb{K}} := \bigcup_{k=1}^{n} \mathfrak{B}(\mathbb{K}_k)$ are called relation-concepts.*

Interpreting a concept graph in a power context family, the object names will be interpreted by objects, e.g. by elements of the set G_0. The concept names of our alphabet will be interpreted by concepts in the context $\mathbb{K}_0$, and relation names of arity k will be interpreted by relation-concepts in the context $\mathbb{K}_k$. Of course, every reasonable interpretation has to respect the orders on the names. This leads to the following definition:

Definition 8. *For an alphabet $\mathcal{A} := (\mathcal{G}, \mathcal{C}, \mathcal{R})$ and a power context family $\mathbb{K}$, we call the union $\lambda := \lambda_\mathcal{G} \dot\cup \lambda_\mathcal{C} \dot\cup \lambda_\mathcal{R}$ of the mappings $\lambda_\mathcal{G}: \mathcal{G} \to G_0$, $\lambda_\mathcal{C}: \mathcal{C} \to \underline{\mathfrak{B}}(\mathbb{K}_0)$ and $\lambda_\mathcal{R}: \mathcal{R} \to \mathfrak{R}_\mathbb{K}$ a $\mathbb{K}$-interpretation of $\mathcal{A}$, if $\lambda_\mathcal{C}$ and $\lambda_\mathcal{R}$ are order-preserving, $\lambda_\mathcal{C}(\top) = \top$, $\lambda_\mathcal{R}(\mathcal{R}_k) \subseteq \underline{\mathfrak{B}}(\mathbb{K}_k)$ for all $k = 1, \ldots, n$, and $(g_1, g_2) \in \lambda_\mathcal{R}(id) \Leftrightarrow g_1 = g_2$ for all $g_1, g_2 \in \mathcal{G}$ hold. The tupel $(\mathbb{K}, \lambda)$ is called context-interpretation of $\mathcal{A}$ or, according to classical logic, $\mathcal{A}$-structure.*

Recall that we defined $\rho(e) := (\rho(v_1), \ldots, \rho(v_n))$ for edges e with $v(e) = (v_1, \ldots, v_k)$. Because $\lambda_\mathcal{G}$ is a mapping on the set $\mathcal{G}$ of object names, it can be naturally extended to tupels of object names. In particular we get $\lambda_\mathcal{G}(\rho(e)) := (\lambda_\mathcal{G}(\rho(v_1)), \ldots, \lambda_\mathcal{G}(\rho(v_n)))$.

Now we can define whether a concept graph is valid in an $\mathcal{A}$-structure. This is done in a canonical way:

Definition 9. *Let $\mathbb{K}$ be a power context family and let $\mathfrak{G}$ be a concept graph. Inductively over $c \in Cut^\top$, we define $\mathbb{K} \models \mathfrak{G}[c]$ in a canonical way:*

$\mathbb{K} \models \mathfrak{G}[c] \iff$

- *$\lambda_\mathcal{G}(\rho(v)) \in Ext(\lambda_\mathcal{C}(\kappa(v)))$ for each $v \in V$ with $cut(v) = c$ (vertex condition)*
- *$\lambda_\mathcal{G}(\rho(e)) \subseteq Ext(\lambda_\mathcal{R}(\kappa(e)))$ for each $e \in E$ with $cut(e) = c$ (edge condition)*
- *$\mathbb{K} \not\models \mathfrak{G}[c']$ for each $c' \in Cut$ with $cut(c') = c$ (iteration over $Cut^\top$)*

For $\mathbb{K} \models \mathfrak{G}[\top]$ we write $\mathbb{K} \models \mathfrak{G}$.

If we have two concept graphs $\mathfrak{G}_1$, $\mathfrak{G}_2$ such that $\mathbb{K} \models \mathfrak{G}_2$ for each $\mathcal{A}$-structure with $\mathbb{K} \models \mathfrak{G}_1$, we write $\mathfrak{G}_1 \models \mathfrak{G}_2$.

Intuitively, $\mathbb{K} \models \mathfrak{G}[c]$ can be read as $\mathbb{K} \models \mathfrak{G}\big|_{area(c)}$. But note that generally $area(c)$ is not a subgraph of $\mathfrak{G}$. Therefore this should not be understood as a precise definition.

4 Calculus

The following calculus is based on the α-calculus of Peirce for existential graphs without lines of identity. These existential graphs consist only of propositional variables and ovals and are equivalent to propositional calculus.

For the sake of intelligibility, the whole calculus is described using common spoken language. Only the rules 'erasure', 'iteration', and 'merging two vertices' will be described in a mathematically precise manner to show that using full sentences does not imply the loss of precision. This precision is definitely necessary because there must not be any possibility for misunderstandings of the rules. The rule 'iteration' for example, says that a subgraph of a graph can be

copied into the same or a nested context. If this is to have a unique meaning, one requires a precise definition of 'subgraph' and 'same or nested context'.

First, we present the whole calculus. The first five rules of the calculus are the original rules of Peirce's α-calculus. The further rules are needed to encompass the orders on the concept- and relation names, to encompass the special properties of the concept name $\top$ and the relation name *id* and to deal with the possibility that different vertices can have the same reference.

Definition 10. *The calculus for (nonexistential) simple concept graph with cuts over the alphabet* $\mathcal{A}$.

- **erasure**
  In positive cuts, any directly enclosed edge, isolated vertex and closed subgraph may be erased.
- **insertion**
  In negative cuts, any directly enclosed edge, isolated vertex and closed subgraph may be inserted.
- **iteration**
  Let $\mathfrak{G}_0 := (V_0, E_0, \nu_0, \kappa_0, \rho_0, Cut_0)$ *be a subgraph of* $\mathfrak{G}$ *and let* $\bar{c} \leq cut(\mathfrak{G}_0)$ *be a cut such that* $\bar{c} \notin Cut_0$. *Then a copy of* $\mathfrak{G}_0$ *may be inserted into* $\bar{c}$.
- **deiteration**
  If $\mathfrak{G}_0$ *is a subgraph of* $\mathfrak{G}$ *which could have been inserted by rule of iteration, then it may be erased.*
- **double negation**
  Double cuts (two cuts c_1, c_2 *with* $cut^{-1}(c_2) = \{c_1\}$*) may be inserted or erased.*
- **isomorphism**
  A graph may be substituted by an isomorphic copy of itself.
- **generalization**
  For evenly enclosed vertices and edges the concept names respectively relation names may be generalized.
- **specialization**
  For oddly enclosed vertices and edges the concept names respectively relation names may be specialized.
- **$\top$-rule**
  For each object name g, *an isolated vertex* $\boxed{\top : g}$ *may be inserted or erased in arbitrary cuts.*
- **merging two vertices**
  For each object name g, *a vertex* $\boxed{\top : g}$ *may be merged into a vertex* $\boxed{P : g}$ *(i.e.* $\boxed{\top : g}$ *is erased and, for every edge* e, $e(i) =$ $\boxed{\top : g}$ *is substituted by* $e(i) =$ $\boxed{P : g}$*).*
  Two vertices in the same cut and with the same reference may be merged.
- **reverse merging of two vertices**
  A merging of two vertices may be reversed.

– **rules of identity**
  - *reflexivity*
    For arbitrary vertices v edges e with $\kappa(e) = id$, $cut(e) = cut(v)$ and $e\big|_1 = e\big|_2 = v$ may be inserted or erased.
  - *symmetry*
    If e is an edge with $\kappa(e) = id$, then e may be substituted by an edge e' which fulfills $e'\big|_1 = e\big|_2$, $e'\big|_2 = e\big|_1$ and $cut(e') = cut(e)$.
  - *transitivity*
    If e_1, e_2 are two edges with $\kappa(e_1) = \kappa(e_2) = id$, $cut(e_1) = cut(e_2)$ and $e_1\big|_2 = e_2\big|_1$, then edges e with $\kappa(e) = id$, $cut(e) = cut(e_1)$, $e\big|_1 = e_1\big|_1$ and $e\big|_2 = e_2\big|_2$ may be inserted or erased.
  - *congruence*
    If e is an edge with $\rho(e\big|_1) = g_1$, $\rho(e\big|_1) = g_2$ and $\kappa(e) = id$, then $\rho(e\big|_1) = g_1$ may be substituted by $\rho(e\big|_1) = g_2$.

To see how these rules can be written down mathematically, here are the precise definitions for the rules 'erasure', 'iteration' and 'merging two vertices'.

– If $\mathfrak{G} := (V, E, \nu, \kappa, \rho, Cut)$ is a concept graph with the closed subgraph $\mathfrak{G}_0 := (V_0, E_0, \nu_0, \kappa_0, \rho_0, Cut_0)$ and if $\bar{c}$ is a cut with $\bar{c} \notin Cut_0$, then let $\mathfrak{G}'$ be the following graph:
  - $V' := V \times \{1\} \cup V_0 \times \{2\}$
  - $E' := E \times \{1\} \cup E_0 \times \{2\}$
  - $\nu'((e,i)) = ((v_1,i),\ldots,(v_n,i))$ for $(e,i) \in E'$ and $\nu(e) = (v_1,\ldots,v_n)$
  - $\kappa'((e,i)) := \kappa(e)$ and $\kappa'((v,i)) := \kappa(v)$ for all $(e,i) \in E'$, $(v,i) \in V'$
  - $\rho'((v,i)) = \rho(v)$ for all $(v,i) \in V'$
  - $Cut' := Cut \times \{1\} \cup Cut_0 \times \{2\}$
  - $area'$ is defined as follows: Let $c \in Cut$.
    for $c \in Cut_0$ let $area'((c,2)) := area(c) \times \{2\}$
    for $c \not\geq \bar{c}$ let $area'((c,1)) := area(c) \times \{1\}$
    for $c \geq \bar{c}$ let $area'((c,1)) := area(c) \times \{1\} \cup (V_0 \cup E_0 \cup Cut_0) \times \{2\}$

  Then we say that $\mathfrak{G}'$ is derived from $\mathfrak{G}$ by *iterating the subgraph $\mathfrak{G}_0$ into the cut $\bar{c}$.*
– If $\mathfrak{G} := (V, E, \nu, \kappa, \rho, Cut)$ is a concept graph with the closed subgraph $\mathfrak{G}_0 := (V_0, E_0, \nu_0, \kappa_0, \rho_0, Cut_0)$, then let $\mathfrak{G}'$ be the following graph:
  - $V' := V \backslash V_0$
  - $E' := E \backslash E_0$
  - $\nu' := \nu|_{E'}$
  - $\kappa' := \kappa|_{V' \cup E'}$
  - $\rho' := \rho|_{V'}$
  - $Cut' := Cut \backslash Cut_0$
  - $area'(c') := area(c')\big|_{V' \cup E' Cut'}$

  Then we say that $\mathfrak{G}'$ is derived from $\mathfrak{G}$ by *erasing the subgraph $\mathfrak{G}_0$.*
– If $\mathfrak{G} := (V, E, \nu, \kappa, \rho, Cut)$ is a concept graph with two vertices $v_1, v_2 \in V$, then let $\mathfrak{G}'$ be the following graph:
  - $V' := V \backslash \{v_1\}$
  - $E' := E$

- ν' is defined as follows: For $\nu(e)\big|_i = v$ let $\nu'(e)\big|_i = \begin{cases} v & v \neq v_1 \\ v_2 & v = v_1 \end{cases}$.
- $\kappa' := \kappa|_{V' \cup E}$
- $\rho' := \rho|_{V'}$
- $Cut' = Cut$
- $area'(c') := area(c')\big|_{V' \cup E \cup Cut}$

Then we say that $\mathfrak{G}'$ is derived from $\mathfrak{G}$ by *merging* v_2 *into* v_1.

These rules are sound and complete with respect to the given semantics (see Theorem 1). Instead of proving this theorem formally, some heuristics for the rules are presented.

First note, that all the rules are in some sense dually symmetric with respect to positive and negative cuts. More precisely, every rule which can be applied in one direction in positive cuts can be applied in the opposite direction in negative cuts, and vice versa. So if a rule can only be applied in positive contexts, this rule has a counterpart for negative contexts (like erasure and insertion or like generalization and specialization). All other rules apply both to positive and negative contexts.

The first five rules are sound and complete concerning the classical propositional calculus. If all vertices and edges would be understood as logically independent propositional variables, these rules would be enough. The rules 'generalization', 'specialization' and '$\top$-rule' encompass the orders on the concept and relation names. Note that $\top$ is not only the greatest element of all concepts: The semantics for $\top$ implies that every object belongs to the extension of the concept $\top$. Thus the generalization rule does not encompass all properties of the concept $\top$, and the $\top$-rule is necessary. The same is true for the relation *id*. In fact it is a congruence relation by definition. This is encompassed by the *id*-rules. The specialization rule can be derived from the other rules, but it is added to keep the calculus symmetric. The rules 'merging two vertices' and 'reverse merging of two vertices' deal with the fact that one object may be the reference for different vertices. With these rules it is possible to transform every concept graph into an equivalent graph in which no cut intersects a relation line. More precisely:

Definition 11. *A concept graph is called free of intersections, if it fulfills the following condition:* $\forall e \in E\ \forall v \in V : v \in V_e \Longrightarrow cut(v) = cut(e)$

It follows from the rules 'merging two vertices' and 'reverse merging of two vertices' that every concept graph is equivalent to a graph free of intersections. And these graphs are easy to read: They have a form which is closely related to the existential graphs without lines of identity, and the soundness and completeness of the first five rules concerning existential graphs can be applied now. This leads to the following essential theorem:

Theorem 1 (soundness and completeness of the calculus).
Two nonexistential, simple concept graph $\mathfrak{G}_1$, $\mathfrak{G}_2$ *with cuts over* $\mathcal{A}$ *satisfy*

$$\mathfrak{G}_1 \vdash G_2 \quad \Longleftrightarrow \quad \mathfrak{G}_1 \models G_2$$

5 Future Work

How to procede with this work is clear. First the approach has to be extended to include graphs with generic markers. The Φ-operator for these graphs has to be elaborated and it has to be proven that simple concept graphs with negation ovals and identity are equivalent to first order logic. In part, this has already been done (e.g. [BMT98]). Afterwards, the approach should be extended to the nested case. It seems reasonable that nested graphs are equivalent to a certain class of formulas of modal logic in such a way that nestings will be interpreted as different possible worlds, which are connected by the structure of the nestings. And again, a semantics and a sound and complete calculus have to be developed.

References

BMT98. F. Baader, R. Molitur, S. Tobies: The Guarded Fragment of Conceptual Graphs. RWTH LTCS-Report. `http://www-lti.informatik.rwth-aachen.de/Forschung/Papers.html`

CMS98. M. Chein, M.-L. Mugnier, G. Simonet: Nested Graphs: A Graph-based Knowledge Representation Model with FOL Semantics, Rapport de Recherche, LIRMM, Université Montpellier II, 1998.

GW99a. B. Ganter, R. Wille: Formal Concept Analysis: Mathematical Foundations. Springer, Berlin-Heidelberg-New York 1999.

GW99b. B. Ganter, R. Wille: Contextual attribute logic, in: W. Tepfenhart, W. Cyre (Eds.): Conceptual Structures: Standards and Practices, Springer Verlag, Berlin–New York 1999, 377-388.

LK96. D. Lukose, R. Kremer: Knowledge Engineering: PART A, Knowledge Representation. `http://www.cpsc.ucalgary.ca/~kremer/courses/CG/`

Pe98. C. S. Peirce: Reasoning and the Logic of Things. The Cambridge Conferences Lectures of 1898. Ed. by K. L. Kremer, Harvard Univ. Press, Cambridge 1992

Pr98a. S. Prediger: Kontextuelle Urteilslogik mit Begriffsgraphen. Ein Beitrag zur Restrukturierung der mathematischen Logik, Shaker Verlag 1998.

Pr98b. S. Prediger: Simple Concept Graphs: A Logic Approach, in: M, -L. Mugnier, M. Chein (Eds.): Conceptual Structures: Theory, Tools and Applications, Springer Verlag, Berlin–New York 1998, 225–239.

Ro73. D. D. Roberts: The Existential Graphs of Charles Sanders Peirce, Mouton The Hague – Paris 1973.

So84. J. F. Sowa: Conceptual Structures: Information Processing in Mind and Machine. Addison Wesley Publishing Company Reading, 1984.

So92. J. F. Sowa: Conceptual Graphs Summary, in: T. E. Nagle, J. A. Nagle, L. L. Gerholz, P. W. Eklund (Eds.): Conceptual Structures: current research and practice, Ellis Horwood, 1992, 3–51.

So99. J. F. Sowa: Conceptual Graphs: Draft Proposed American National Standard, in: W. Tepfenhart, W. Cyre (Eds.): Conceptual Structures: Standards and Practices, Springer Verlag, Berlin–New York 1999, 1-65.

So00. J. F. Sowa: Knowledge Representation: Logical, Philosophical, and Computational Foundations. Brooks Cole Publishing Co., Pacific Grove, CA, 2000.

We95. M. Wermelinger: Conceptual Graphs and First-Order Logic, in: G. Ellis et al. (Eds.): Conceptual Structures: Applications, Implementations and Theory, Springer Verlag, Berlin–New York 1995, 323–337.

Extending the CG Model by Simulations

Jean - François Baget

LIRMM (CNRS and Université Montpellier II)
161, rue Ada, 34392 Montpellier - France
baget@lirmm.fr

Abstract. Conceptual graphs (CGs) share with FOL a fundamental expressiveness limitation: only higher-order logics allow assertions of properties on predicates. This paper intends to push back this limit by reifying underlying relations of CGs (*is-a*, *a-kind-of*, *referent*) into first-class objects (i.e. nodes) of an equivalent, labelled graphs (LG) model.
Benefits of this reification, applied on a subset of CGs, namely simple graphs and rules of form "if G then H", are discussed in terms of expressiveness, succintness and robustness. We show that using the LG model as an *interpreter* allows us to improve and extend the results in [2].

1 Introduction

Labelled Graphs have long been used as a natural and readable way to represent symbolic knowledge. The *Conceptual Graphs* model [11] can be seen as a higher level abstraction of Representation Networks [9,6], benefiting from further developments in Knowledge Representation: a clear distinction between *entities* and *relations*, and between *factual knowledge* and *background knowledge* (Fig. 1).

Though the CG model adds structuration and clarity to the represented knowledge, this improvement has a subtle trade-off: the relations s (for *subset of*, *a kind of*) and e (for *element of*, *is a*), explicit in representation networks, become implicit in CGs. Indeed, *a kind of* is encoded in the type hierarchies defining background knowledge, and *is a* is encoded in concept node labels.

The drawback is that CGs cannot handle these relations as "first-class objects", they are used in the deduction mechanism, but cannot be the *object* of reasonings. From a FOL point of view, it is possible to assert properties on objects, but not on relations (predicates) between objects. Some consequences are pointed out in [2], in a CG model restricted to simple graphs (SGs) and "if ... then" rules (SG rules). Namely:

1. It is not possible to express that *"If two concept nodes have the same individual marker, they represent the same entity"*. Instead, [2] uses one rule for each individual marker in the support (i.e. for each individual marker m, "if two concept nodes are marked by m, then they represent the same entity").
2. It is not possible to define the relation type EQUIVALENCE as a subtype of REFLEXIVE, SYMMETRICAL and TRANSITIVE, such that these properties (themselves encoded in rules) are inherited by EQUIVALENCE and all its subtypes. Instead, [2] defines a family of rules expressing the equivalence property, that must be implemented for *every* equivalence relation.

B. Ganter and G.W. Mineau (Eds.): ICCS 2000, LNAI 1867, pp. 277–291, 2000.
© Springer-Verlag Berlin Heidelberg 2000

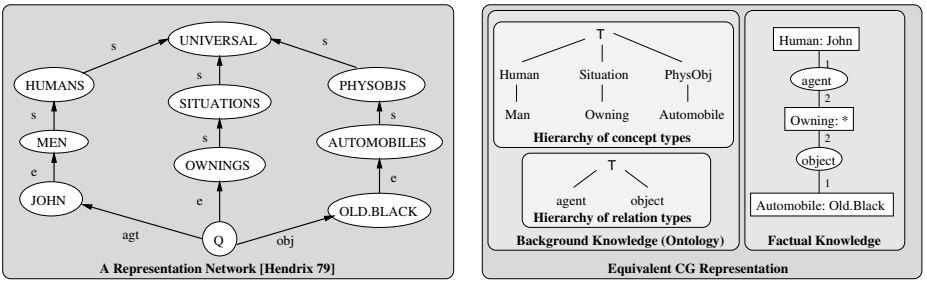


Fig. 1. Representation Networks and Conceptual Graphs Formalisms

These solutions are not satisfying in terms of succintness (families of rules defined for each marker and/or type) neither in terms of robustness (when a new type or marker is added, it is necessary to add new rules). Instead, we propose to reify implicit relations of the SG model (IS-A, A-KIND-OF, REFERENT) into first-class objects of a model that *simulates* SGs.

In Sect. 2 we present a general framework for simulating a reasoning model into another one. This framework will then be applied for simulating the {SG, rules} model (recalled in Sect.3) into a {labelled graphs, rules} model. Sect. 4 and 5 study this latter model, its semantics, and its expressiveness with respect to {SG, rules}. In Sect. 6, we build step by step a simulation of the {SG, rules} model into the {labelled graphs, rules} model, where implicit notions of the simulated model are reified. Finally, in Sect. 7, we discuss our gains in terms of expressiveness and present an extension of the {SG, rules} model, obtained from this simulation.

2 A Simulation Framework for Reasoning Models

In this paper, we study how a reasoning model can gain expressiveness by a simulation into another one. As [7], we define a reasoning model by: a *language* or *syntax* that specifies the objects we manipulate; a *deduction system* that computes the relation $\vDash$ between objects; and *valuation rules* or *semantics* that are given here through a translation into FOL formulas. To compare different knowledge representation formalisms, [5] gives several evaluation criteria:

1. Does the formalism support efficient reasonings?
2. How expressive is it?
3. How succinctly can the formalism express the sets of models that it can?
4. How does the knowledge representation form fare in the face of change?

Comparison of reasoning models in terms of complexity or decidability of the deduction problem (1.) is only slightly discussed in this paper. To compare the expressiveness of two reasoning models (2.), we define the notion of *simulation* of a reasoning model into another one, which is basically a reduction of a decision problem to another. We propose here some criteria to evaluate the very subjective notion of a *good* simulation, and discuss in the final part of this paper how

a simulation that reifies implicit notions of a model can add expressiveness, succintness (3.) and robustness (4.) to this model.

Definition (Simulation). *A* simulation *of a reasoning model $\mathcal{M}$ into a model $\mathcal{N}$ is a mapping Γ from the objects of $\mathcal{M}$ into the objects of $\mathcal{N}$ such that A can be deduced from B in $\mathcal{M}$ if and only if $\Gamma(A)$ can be deduced from $\Gamma(B)$ in $\mathcal{N}$.*

2.1 Complexity of Γ: Importance of a Linear Simulation

We want a simulation to prove that *all reasonings in the model $\mathcal{M}$ can be performed in $\mathcal{N}$*, and wish to forbid part of the deduction mechanism in $\mathcal{M}$ to be encoded in the mapping Γ. The usual way to assert that $\mathcal{M}$ is "simpler" than $\mathcal{N}$ is to find a *polynomial time* simulation of $\mathcal{M}$ into $\mathcal{N}$. But a polynomial time mapping could be complex enough to encode in itself mechanisms of $\mathcal{M}$ that cannot be represented in $\mathcal{N}$ (Sect. 6.1). We want a simulation to compute a *syntactic translation* only. That is why we are interested in simulations that fulfill a much stronger constraint: *linear time mappings*. If there exists a polynomial time (resp. linear time) simulation of $\mathcal{M}$ into $\mathcal{N}$, we say that $\mathcal{N}$ is a *generalization* (resp. *strong generalization*) of $\mathcal{M}$. Reasoning models that generalize each other are said *equivalent* (resp. *strongly equivalent*).

2.2 Backward Translation of Objects and Proofs

It is important, for every object A in $\mathcal{N}$, to compute efficiently if $A \in \Gamma(\mathcal{M})$ (i.e. if A is the representation of an object of $\mathcal{M}$) and, in that case, to translate back A into the *unique* object A' of $\mathcal{M}$ such that $\Gamma(A') = A$; thus Γ should be injective. This feature allows to develop a reasoning model on top of an existing one by implementing only the simulation Γ and its backward translation in such a way that this translation is invisible to the end user: he only needs to manipulate objects of the model $\mathcal{M}$. The interest of backward translation of objects extends to backward translation of *proofs*. Intuitively, if A and B are objects of $\mathcal{M}$, a proof of $A \vDash B$ in the model $\mathcal{M}$ can be seen as a sequence of operations allowed by the deduction system of $\mathcal{M}$. Consider a proof $\Gamma(A) \vDash \Gamma(B)$ of the model $\mathcal{N}$. A proof $A \vDash B$ of $\mathcal{M}$ can be built if there exists a *surjective mapping* of the proofs in $\Gamma(\mathcal{M})$ into the proofs in $\mathcal{M}$. This property would allow an end user to visualize all proofs in the model $\mathcal{M}$, even if reasonings are *interpreted* into the model $\mathcal{N}$. Even more, should we want an application that compute deduction in $\mathcal{M}$ with some user interaction features (step by step visualization of proof, user assisted deduction), this property should extend to subsequences of a proof.

2.3 Preservation of Complexity Classes

Let $C_{\mathcal{M}}$ and $C_{\mathcal{N}}$ be the complexity classes of the deduction problems in $\mathcal{M}$, and $\mathcal{N}$; let $\mathcal{A}_{\mathcal{M}}$ and $\mathcal{A}_{\mathcal{N}}$ be algorithms solving these problems ("respecting" the complexity classes $C_{\mathcal{M}}$ and $C_{\mathcal{N}}$); and let Γ be a simulation of $\mathcal{M}$ into $\mathcal{N}$. We want to keep reasonings in the model $\mathcal{M}$ *as efficient as possible*, even when they are interpreted through the translation in the model $\mathcal{N}$. That is why $\mathcal{A}_{\mathcal{N}}$, applied

to objects of $\Gamma(\mathcal{M})$, should satisfy $C_{\mathcal{M}}$. Note that, if it is not the case, $\mathcal{A}_{\mathcal{M}}$ can still be used, assuming that we have an efficient reconnaissance algorithm for $\Gamma(\mathcal{M})$, or can even give new insights for a general improvement of $\mathcal{A}_{\mathcal{N}}$. This discussion could also be extended to the preservation of complexity classes for particular subsets of $\mathcal{M}$.

In this paper, we propose a linear simulation Θ from the {SG, rules} model into a "low-level" model. Since there also exists a linear simulation from the latter model into {SG, rules}, both models are strongly equivalent. This simulation ensures backward translations of objects as well as proofs, and, since the two models are strongly equivalent, complexity classes of deduction problems are preserved. The simulation Θ reifies implicit notions of the CG model (types, markers) into first-class objects of the "low-level" model. Benefits of this reification will be further discussed in the last section (Sect. 7) of this paper.

3 The Simple Graphs and {SG, Rules} Models

Here we recall fundamental definitions and results on a subset of CGs: simple graphs (SGs) [4], and simple graphs rules [8,2].

3.1 Simple Graphs

Syntax. Basic ontological knowledge is encoded in a structure called a *support*. A support $\mathcal{S} = (T_C, T_R, \mathcal{I})$ is given by two finite partially ordered sets T_C and T_R, respectively the set of *concept types* and the set of binary[1] *relation types*, and a set of *individual markers* $\mathcal{I}$. T_C (resp. T_R) admits a greatest element, denoted by $\top_C$ (resp. $\top_R$). A partial order is defined on $\mathcal{I} \cup \{*\}$: the *generic marker* $*$ is the greatest element and elements of $\mathcal{I}$ are pairwise non comparable.

We denote by $G = (V_C, V_R, E, \text{label}, \text{co-ref})$ a *simple graph* defined on a support, where V_C and V_R are respectively the sets of *concept nodes* and *relation nodes*; E is the set of *edges* (the two edges incident to a relation node are labelled by 1 and 2); *co-ref*, the co-reference relation, is an equivalence relation on the set of generic concept nodes. Two concept nodes are said *co-identical* if they have the same individual marker or if they belong to the same *co-reference* class. Intuitively, it means that these two concept nodes represent the same entity. In what follows, we impose that *co-identical* concept nodes have the same type.

Deduction System. Let $\mathcal{S}$ be a support, and G and H be two SGs defined on $\mathcal{S}$. A projection Π from H into G is a mapping from $V_C(H)$ into $V_C(G)$ and from $V_R(H)$ into $V_R(G)$ that preserves edges and their labelling, may restrict labels of concept and relation nodes, and preserves co-identity:

1. $\forall e = (x_1, x_2) \in E(H), e' = (\Pi(x_1), \Pi(x_2)) \in E(G)$ and $\text{label}(e) = \text{label}(e')$
2. $\forall x \in V_C(H) \cup V_R(H), \Pi(x) = x' \Rightarrow \text{label}(x') \leq \text{label}(x)$
3. $\forall x, y \in V_C(H), \text{co-ref}(x, y) \Rightarrow \text{co-ident}(\Pi(x), \Pi(y))$

[1] For the sake of simplicity, we restrict these definitions to binary relation types but all results could easily be extended to n-ary relation types.

A SG is said in *normal form* if all co-identity classes are restricted to the trivial ones (co-ident$(x, y) \Rightarrow x = y$). The normal form $NF(G)$ of a graph G is obtained by fusionning all nodes that belong to the same co-identity class.

We consider two deduction systems in the SG model (effectively defining two reasoning models): we note $\mathcal{S}, G \vDash H$ if there exists a projection from H into G, and $\mathcal{S}, G \vDash_{NF} H$ if there exists a projection of H into $NF(G)$ (remark that if $\mathcal{S}, G \vDash H$, then $\mathcal{S}, G \vDash_{NF} H$, but the reverse does not hold).

Semantics. The deduction system $\vDash_{NF}$ is sound and complete w. r. t. the FOL semantics Φ [4], i. e. $\mathcal{S}, G \vDash_{NF} H$ iff $\Phi(\mathcal{S}), \Phi(G) \vDash \Phi(H)$, while the deduction system $\vDash$ is sound and complete w. r. t. the FOL semantics Ψ (an alternative to Φ proposed in [10]). A formal presentation of these semantics can be found in [10], while [2] presents their consequences on the different reasonings allowed.

3.2 Simple Graph Rules

Simple graph rules (SG rules) can be seen as graphs that embody knowledge of the form "if *hypothesis* can be deduced from a graph, so can *conclusion*".

Syntax. Let $\mathcal{S}$ be a support. A *simple graph rule* is given by a simple graph R defined on $\mathcal{S}$ and a mapping color: $V(R) \rightarrow \{0, 1\}$. The subgraph of R generated by 0-colored (resp. 1-colored) nodes is called the *hypothesis* (resp. *conclusion*) of the rule. Furthermore, the following constraints ensure that the application of a rule on a SG always generates a SG: 1) the subgraph of R generated by 0-colored nodes is a SG (but without restrictions on *co-ref*); 2) co-identity classes whose members are of different colors are forbidden; 3) co-identity classes whose members are 0-colored nodes are without restriction; 4) co-identity classes whose members are 1-colored nodes suffer the same restrictions as for SGs.

Deduction System. Let $\mathcal{S}$ be a support, G be a SG defined on $\mathcal{S}$, and R be a SG rule defined on $\mathcal{S}$. We say that R is *applicable* to G iff there exists a projection from the hypothesis of R into G. Let Π be such a projection. An *immediate derivation* of G by the application of R following Π is a graph G' obtained by 1) making the disjoint union of G and a copy of the conclusion of R; 2) for each edge (r, c) labelled n in R, where c is a concept node of color 0 and r is a relation node of color 1, linking the copy of r to $\Pi(c)$ by an edge labelled n.

Let $\mathcal{R}$ be a set of SG rules defined on $\mathcal{S}$. We say that $G, \mathcal{R}$ *derives* a SG G' if there exists a sequence of immediate derivations leading to G' by application of rules in $\mathcal{R}$. We say that $G, \mathcal{R}$ *normally derives* G' if each immediate derivation is followed by a normalization of the obtained graph.

Then again, we obtain two deduction systems in the {SG, rules} model. We note $\mathcal{S}, \mathcal{R}, G \vDash H$ if there exists a graph G' such that $G, \mathcal{R}$ derives G' and H can be projected into G'. We note $\mathcal{S}, \mathcal{R}, G \vDash_{NF} H$ if there exists a graph G' such that $NF(G), \mathcal{R}$ normally derives G' and H can be projected into G'.

Semantics. The FOL semantics Φ and Ψ are extended to SG rules: $\vDash_{NF}$ is sound and complete w. r. t. Φ [8], and $\vDash$ is sound and complete w. r. t. Ψ [2].

4 The Labelled Graphs Model

The labelled graphs (LG) model is very similar to the SG model, but does not take into account the notions of support or type: we consider it as a "syntactically neutral" version of SGs. We first define formally the objects we manipulate, then propose two deduction mechanisms based on projection whose semantics mimic the ones defined in the SG model. It is not surprising, this model being simpler, that there exists a linear simulation of labelled graphs into SGs.

4.1 Syntax and Deduction Mechanisms

Let $\mathcal{A}$ be a set of symbols, and $* \notin \mathcal{A}$ a special symbol named the *generic label*. A labelled graph (LG) $G = (V, E, \text{label})$ is given by a set of nodes V, a symmetrical relation E on $V \times V$, and a mapping label from V into $\mathcal{A} \cup \{*\}$.

Let H and G be two LGs, a *LG-projection* (projection in the LG model) is a mapping Π from $V(H)$ into $V(G)$ that preserves edges and non-generic labels:

1. $\forall (x_1, x_2) \in E(H), (\Pi(x_1), \Pi(x_2)) \in E(G)$
2. $\forall x \in V(H), \text{label}(x) \neq * \Rightarrow \text{label}(\Pi(x)) = \text{label}(x)$

A LG is said in *normal form* if all non-generic labels are different. The normal form $NF(G)$ of a LG G is obtained by fusionning nodes having the same, non-generic label. We note $H \sqsubseteq G$ if there exists a LG-projection from H into G.

We note $G \models H$ if there exists a LG-projection from H into G, and note $G \models_{NF} H$ if there exists a LG-projection from H into $NF(G)$. Unless otherwise noted, we will consider $\models$ when referring to the LG model.

4.2 Linear Simulations of Labelled Graphs into SGs

Lemma 1. *The* $(SGs, \models)$ *model is a strong generalization of the* $(LGs, \models)$ *model and the* $(SGs, \models_{NF})$ *model is a strong generalization of the* $(LGs, \models_{NF})$ *model.*

Proof. This result is not surprising, but the simulations Γ of $(\text{LGs}, \models)$ into $(\text{SGs}, \models)$ and Γ_{NF} of $(\text{LGs}, \models_{NF})$ into $(\text{SGs}, \models_{NF})$ will be used in Theor.1.

The simulation Γ is illustrated in Fig. 2 by the transformation of the LG G_1 into the SG G_2. We define a support $\Gamma(\mathcal{A})$ as follows: $*$ is associated to the greatest element of T_C, each label of $\mathcal{A}$ is associated to a distinct type of T_C (denoted by the same symbol). All types corresponding to $\mathcal{A}$ are pairwise non-comparable. T_R is restricted to one relation type, named *link*. $\mathcal{I}$ is empty. Each node of an LG is transformed into a concept node whose type is the one associated with its label and whose marker is generic. Each edge ab is transformed into two symmetrical relation nodes typed *link*, linking $\Gamma(a)$ and $\Gamma(b)$.

Γ_{NF} is illustrated in Fig. 2 by the transformation G_1 into G_3. The only concept type in $\Gamma_{NF}(\mathcal{A})$ is *node*, T_R is restricted to the relation type *link*, and $\mathcal{I}$ is equal to $\mathcal{A}$. The difference with Γ consists in concept node labeling: a node labelled m is transformed into a concept node whose type is *node*, and whose marker is generic if m is generic, or the element of $\mathcal{I}$ associated to m otherwise.

We now check that Γ and Γ_{NF} are linear transformations, that $G \models H$ iff $\Gamma(\mathcal{A}), \Gamma(G) \models \Gamma(H)$, and that $G \models_{NF} H$ iff $\Gamma_{NF}(\mathcal{A}), \Gamma_{NF}(G) \models_{NF} \Gamma_{NF}(H)$. □

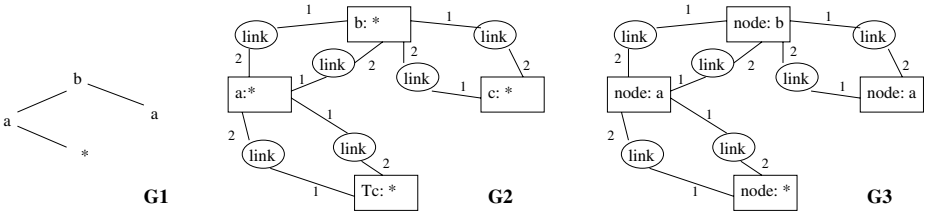


Fig. 2. Simulations of LG models in SGs

4.3 Semantics

Semantics ϕ and ψ for the two labelled graph models are defined as:

- $\psi(\mathcal{A}) = \Phi(\Gamma(\mathcal{A}))$ and $\psi(G) = \Phi(\Gamma(G))$
- $\phi(\mathcal{A}) = \Phi(\Gamma_{NF}(\mathcal{A}))$ and $\phi(G) = \Phi(\Gamma_{NF}(G))$

Theorem 1 (Soundness and completeness). *The deduction system* $\vDash$ *(resp.* $\vDash_{NF}$*) in the LG model is sound and complete w. r. t. the semantics* ψ *(resp.* ϕ*).*

Proof. This is a corollary of Lem. 1, since Γ only creates SGs in normal form. □

5 The {Labelled Graphs, Rules} Model

Labelled graph rules (LG rules) are designed as a "light syntax" version of SG rules. Extending the results in Sect. 4, we simulate the {LG, rules} model into the {SG, rules} model, and give sound and complete semantics.

5.1 Syntax and Deduction Mechanisms

A *LG rule* $R = (V, E, \text{label}, \text{color})$, is given by a LG (V, E, label) and a mapping color from V into $\{0, 1\}$. The subgraph $\text{hyp}(R)$ (resp. $\text{conc}(R)$) generated by 0-colored (resp. 1-colored) nodes, circled in white (resp. gray) in the representation of the graph, is called the *hypothesis* (resp. *conclusion*) of the rule (Fig. 3).

A LG rule R is *applicable* to a LG G if there exists a LG-projection (say Π) from the hypothesis of R into G. In that case, a LG G' is an *immediate derivation* of G following R and Π if G' is obtained by making the disjoint union of G and a copy of $\text{conc}(R)$, then, for each edge $hc \in E(R)$ such that $h \in \text{hyp}(R)$ and $c \in \text{conc}(R)$, adding an edge between $\Pi(h)$ and $\text{copy}(c)$.

The mechanism of rule application is illustrated in Fig. 3, where $G_1, \ldots G_5$ are the immediate derivations of G following R. The notions of *derivation* and *normal derivation* are defined as in Sect. 3.2, as a sequence of immediate derivations (each one followed by a normalization in the latter case). We note $\mathcal{R}, G \vDash H$ if there exists a graph G' such that $G, \mathcal{R}$ derives G' and there exists a LG projection of H into G'. We note $\mathcal{R}, G \vDash_{NF} H$ if there exists a graph G' such that $NF(G), \mathcal{R}$ normally derives G' and there exists a LG projection of H into G'. Unless otherwise noted, we will use $\vDash$ when considering the {LG, rules} model.

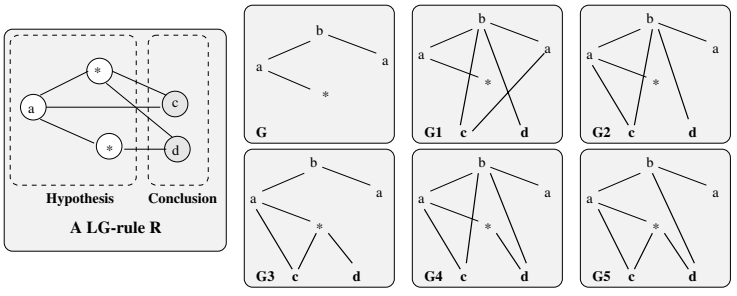


Fig. 3. Five immediate derivations of a graph G

5.2 Linear Simulations of Labelled Graphs Rules into SG Rules

Lemma 2. *The* $(\{SGs,\ rules\}, \vDash)$ *(resp.* $(\{SGs,\ rules\}, \vDash_{NF})$*) model is a strong generalization of the* $(\{LGs,\ rules\}, \vDash)$ *(resp.* $(\{LGs,\ rules\}, \vDash_{NF})$*) model.*

Proof. As in the proof of lemma 1, we exhibit two simulations Δ and Δ_{NF}. Let $R = (G, \text{color})$ be a LG rule. $\Delta(R)$ (resp. $\Delta_{NF}(R)$) is obtained by adding to $\Gamma(G)$ (resp. $\Gamma_{NF}(G)$) the following coloration: the color 0 is assigned to nodes issued from the hypothesis of $\mathcal{R}$, the color 1 to all others. We can verify that these simulations are linear, that, thanks to lemma 1, a LG rule R is applicable to a labelled graph G iff $\Delta(R)$ (resp. $\Delta_{NF}(R)$) is applicable to $\Gamma(G)$ (resp. $\Gamma_{NF}(G)$), and that, to every immediate derivation of a LG G into G' following R corresponds an immediate derivation of $\Gamma(G)$ into $\Gamma(G')$ following $\Delta(R)$. □

5.3 Semantics

We extend the semantics ϕ and ψ as in Sect. 4. Let R be a LG rule:

- $\psi(R) = \Phi(\Delta(R))$
- $\phi(R) = \Phi(\Delta_{NF}(R))$

Theorem 2 (Soundness and completeness). *The deduction system* $\vDash$ *(resp.* $\vDash_{NF}$*) in the* $\{LG,\ rules\}$ *model is sound and complete w. r. t.* ψ *(resp.* ϕ*).*

Proof. Lemma 2, soundness and completeness for the {SG, rules} models. □

6 Simulations of {SG, Rules} into {LG, Rules}

Having defined the LG and {LG, rules} models as particular cases of SG and {SG, rules} (Lem. 1 and 2), we now simulate various SG models (beginning with a basic one) into {LG, rules} models by reifying implicit relations of SGs.

6.1 Basic Simple Graphs

We call basic simple graph (BSG) a SG defined on a support $\mathcal{S}$ without co-reference links (all co-reference classes are trivial), nor individual markers ($\mathcal{I} = \emptyset$). BSGs being in normal form, the Φ and Ψ semantics are equivalent.

Proposition 1. *The BSG model is equivalent to the LG model.*

Proof. We have proven (lemma 1) that the BSG model is a strong generalization of the LG model, since Γ produces only BSGs. We now exhibit a *polynomial* simulation of the BSG model into the LG model. Fig.4 represents the simulation, by the mapping Θ_1, of the graph in Fig. 1 (without its individual markers).

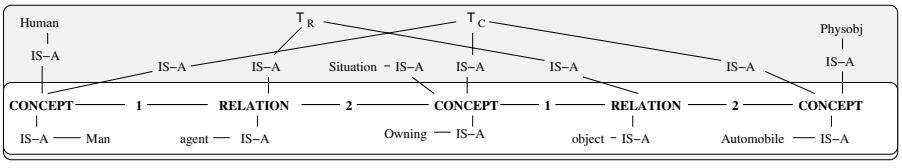


Fig. 4. Simulation of the BSG model into the LG model via Θ_1

We first define skel(G), a LG called the skeleton of G (boldface in Fig. 4), obtained by creating a node skel(c) labelled by CONCEPT (resp. RELATION) for each concept node (resp. relation node) x in G, and, for each edge $rc \in G$, a node labelled by the label of this edge, whose neighbors are skel(r) and skel(c).

The LG $\Theta_1(G, \mathcal{S})$ is then obtained from skel(G), by adding a node labelled by t for each type $t \in T_C \cup T_R$ (there must be no identical symbol in these two sets), then for each node x typed t in G, for each type t' in $T_C \cup T_R$ such that $t \leq t'$, adding a node labelled IS-A linking skel(x) to the node labelled by t'. We denote by $\Theta_1^-(G)$ a subgraph of Θ_1 where nodes are only linked by IS-A to their most specific type (the subgraph in the white rectangle of Fig. 4).

We check that Θ_1 is a *polynomial* simulation. For computational efficiency, we point out that, G and H being two BSGs defined on $\mathcal{S}$: $\mathcal{S}, G \vDash H$ iff $\Theta_1(G, \mathcal{S}) \vDash \Theta_1^-(H)$ (leading to a smaller graph to project). □

We doubt it is possible to find a *linear* simulation of the BSG model into the LG model. Intuitively, reasonings by LG-projection and reasonings on the hierarchy of types have a fundamentally different nature: one is concerned with existence of objects, the other embodies knowledge on all types that verify some property. Reasonings on a hierarchy are basically rules. The following simulations will be given by a linear mapping translating only syntactic information of a SG, and by a constant set of rules (considered as a *library* for the LG interpreter) that encode reasonings that cannot be achieved by mere LG-projection.

Proposition 2. *The {LG, rules} model is a strong generalization of the BSG model.*

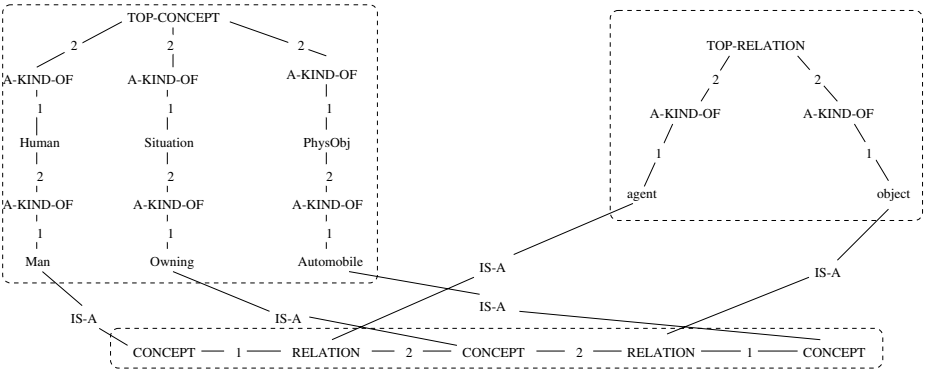


Fig. 5. Transformation via Θ_2 of the BSG and support in Fig. 1 into a LG

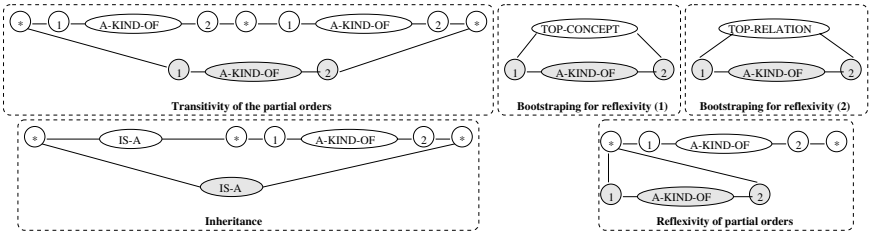


Fig. 6. Library $\mathcal{R}_s$ of LG-rules used for the transformation Θ_2

Proof. Let G be a BSG defined on $\mathcal{S}$. We first encode the support in a LG $\text{enc}(\mathcal{S})$. This graph contains two particular nodes, labelled TOP-CONCEPT and TOP-RELATION, that represent the types $\top_C$ and $\top_R$. Then, for each type t being a *direct* subtype of types $t_1, \ldots, t_k$, we add a new node labelled t that is linked by a chain – 1 – A-KIND-OF – 2 – to the representation of each of the t_i. The LG $\Theta_2(G, \mathcal{S})$ is obtained by the disjoint union of $\text{skel}(G)$ (see proof of Prop. 1) and $\text{enc}(\mathcal{S})$, then, for each node labelled CONCEPT or RELATION in this skeleton, linking it by a chain[2] – IS-A – to the node labelled by *its most specific type*.

The graph in Fig. 5 illustrates this transformation: note that, this time, transformation of the support has been purely syntactic. No information on the properties of IS-A or A-KIND-OF has been needed to generate this graph. These properties are given by the rules $\mathcal{R}_s$ in Fig. 6. These rules express the transitivity and reflexivity of the partial order on types, and that an entity or relation inherits all the super-types of its given type. Though only this last rule is necessary to prove the proposition, they will all be used when simulating co-reference.

We must now check that Θ_2 is linear and verify the following equivalences (the last one being presented in an optimization perspective): $\mathcal{S}, G \models H$ iff $\Theta_2(G, \mathcal{S}), \mathcal{R}_s \models \Theta_2(H, \mathcal{S})$ iff $\Theta_2(G, \mathcal{S}), \mathcal{R}_s \models \Theta_1^-(H)$ □

[2] As for *a-kind-of*, we could have used a chain – 1 – IS-A – 2 –. This is not necessary here, since the "orientation" is implicitly given, from the object to its type.

Remark 1. All rules presented here require there is a *unique* node representing each type (same for individual markers). All LG rules to be applied on SGs must be designed in such a way that the graphs they generate keep this invariant true.

6.2 Introducing Co-reference and Individual Markers

Proposition 3. *The {LG, rules} model is a strong generalization of the (SG, $\models$) and the (SG, $\models_{NF}$) models.*

Proof. Let G be a SG on a support $\mathcal{S}$. As in [2], reification of co-identity is done by simulation into the SG model itself. $\theta(\mathcal{S})$ is the support obtained by adding to $\mathcal{S}$ four new relation types, namely REFLEXIVE, SYMMETRICAL, TRANSITIVE and EQUIVALENCE, the first being subtypes of $\top_R$, the latter a subtype of the three others, then adding CO-IDENT as a subtype of EQUIVALENCE, and CO-REF as a subtype of CO-IDENT. $\theta(G)$ is obtained by adding a relation node typed CO-REF between nodes in the same co-reference class (this simulation is linear, we only need to generate $n-1$ nodes for a co-reference class of size n). It remains now to handle individual markers. The mapping Θ_3 is an extension of Θ_2. The graph obtained with Θ_3 contains one node labelled m for each individual marker m in $\mathcal{I}$ (required in Sect. 6.3), all linked to a unique node labelled MARKERS. Each node labelled CONCEPT obtained from an individual node marked m is linked by a node labelled REFERENT to the node representing m. In Θ_3^-, only the most specific types and the individual markers present in the graph are represented.

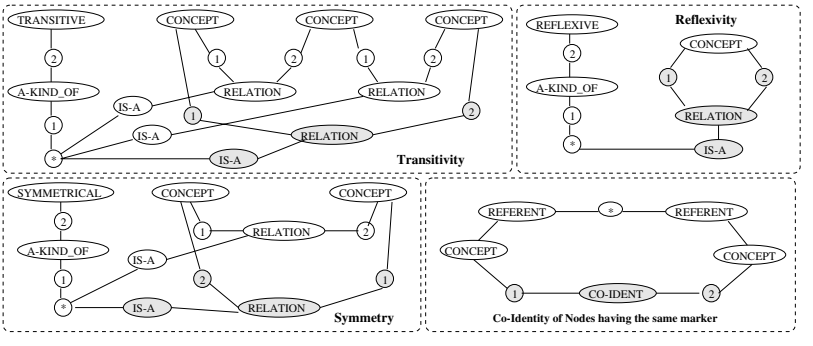


Fig. 7. Library $\mathcal{R}_c$ of LG-rules defining the co-identity relation

Fig. 7 defines the set of rules $\mathcal{R}_c$: the three first rules ensure that, for every relation type declared as a subtype of EQUIVALENCE (and in particular CO-REF and CO-IDENT), this relation behaves as an equivalence relation. The last one indicates that concept nodes having the same marker are co-ident. These rules are sufficient to simulate the (SG, $\models$) model, i.e.: $\mathcal{S}, G \models H$ iff $\Theta_3(\theta(G), \theta(\mathcal{S})), \mathcal{R}_s \cup \mathcal{R}_c \models \Theta_3^-(\theta(H))$ (proof of this assertion is given by Prop. 2, pointing out that these rules are a "higher-order version" of the ones presented in [2]).

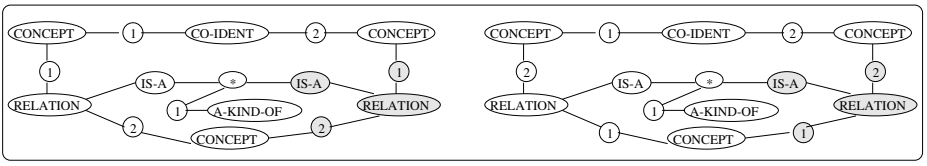


Fig. 8. Library $\mathcal{R}_n$ of LG-rules handling the normalization process

To simulate the normalization step for deduction in the (SG, $\vDash_{NF}$) model, we add another set of rules, $\mathcal{R}_n$ (Fig. 8). Again, thanks to Prop. 2, by verifying that these rules are a "higher-order version" of the ones presented in [2], we prove that $\mathcal{S}, G \vDash_{NF} H$ iff $\Theta_3(\theta(G), \theta(\mathcal{S})), \mathcal{R}_s \cup \mathcal{R}_c \cup \mathcal{R}_n \vDash \Theta_3^-(\theta(H))$. □

6.3 Introducing Rules

Theorem 3. *The* $\{LG, rules\}$, $(\{SG, rules\}, \vDash)$ *and* $(\{SG, rules\}, \vDash_{NF})$ *models are strongly equivalent.*

Proof. Sect. 5 proves one part of the equivalence, we must now prove that the {LG, rules} model is a strong generalization of both the ({SG, rules}, $\vDash$) and the ({SG, rules}, $\vDash_{NF}$) models. Let $\Upsilon(R)$ be a linear time mapping from SG rules into LG rules. Let $R = (G, color)$ be a SG rule defined on $\mathcal{S}$. $\Upsilon(R)$ is obtained from $\Theta_3^-(\theta(G))$ (the simulation of a SG into a LG) by assigning the color 0 to *all* nodes defining the support and to nodes representing the hypothesis of R, and the color 1 to all others (Fig. 9). This translation is designed in such a way that the "unique type and marker" invariant (see Rem. 1) is preserved by derivation. Let $\mathcal{S}$ be a support, G and H be two SGs defined on $\mathcal{S}$, we prove that:

1. $\mathcal{S}, \mathcal{R}, G \vDash H$ iff $\Theta_3(\theta(G), \theta(\mathcal{S})), \Upsilon(\mathcal{R}) \cup \mathcal{R}_s \cup \mathcal{R}_c \vDash \Theta_3^-(\theta(H))$
2. $\mathcal{S}, \mathcal{R}, G \vDash_{NF} H$ iff $\Theta_3(\theta(G), \theta(\mathcal{S})), \Upsilon(\mathcal{R}) \cup \mathcal{R}_s \cup \mathcal{R}_c \cup \mathcal{R}_n \vDash \Theta_3^-(\theta(H))$

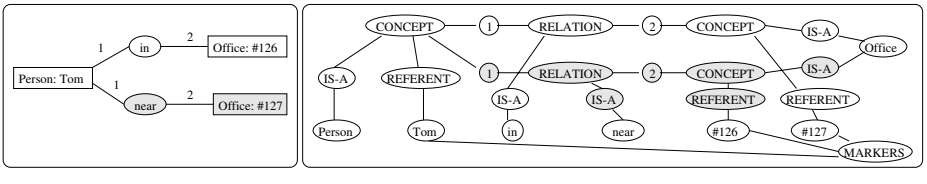


Fig. 9. A SG rule and the obtained transformation by Υ

Check that the application of a SG rule R on a SG G gives a SG G' iff the simulation of G derives the simulation of G' by some applications of the LG rules $\mathcal{R}_s \cup \mathcal{R}_c$, then an application of $\Upsilon(R)$. □

Corollary 1. *The* $\{Nested\ Graphs, rules\}$ *model is strongly equivalent to the* $\{LG, rules\}$ *model.*

Proof. [2] simulates of nested graphs into SGs, using a set $\mathcal{N}$ of SG rules. We compose this *linear* simulation with Θ_3, and translate $\mathcal{N}$ by Υ. □

7 Conclusion

In this paper, we have explored a basic projection-based reasoning model, the LG model, *a priori* less expressive than the SG model. However, we have proven that the {LG, rules} model is *strongly equivalent* to the {SG, rules} and even the {Nested graphs, rules} models. The simulation Θ exhibited to translate an instance of {SG, rules} deduction problem into an instance of the {LG, rules} deduction problem possess all "good properties" discussed in Sect. 2:

1. Θ is linear;
2. We prove, though only hints are included in this paper, there exists a *linear* (resp. *polynomial*) backward translation of objects (resp. proofs) of the {LG, rules} model into objects (resp. proofs) of the {SG, rules} model [3];
3. Since the two models are strongly equivalent, they are both in the same complexity class (i.e. deduction problems are *semi-decidable* in both models).

Let $\mathcal{M}$ be a reasoning model: the {LG, rules} model can be seen as an *interpreter* for $\mathcal{M}$ if it is provided with a simulation of $\mathcal{M}$, backward translations mechanisms, and a *library* of LG-rules that mimics specific reasonings of $\mathcal{M}$. But, having built an interpreter for the {SG, rules} and {NG, rules} models by designing a single, easier to implement model, what are the benefits of this simulation? We first discuss its added expressiveness, in terms of succintness and robustness, then show how the simulation can be used to extend the model.

7.1 The Succintness and Robustness Criteria

Let us first point out the "rules factorization" gained by the simulation. Though [2] gives SG rules for the same result, it requires: $3 \times k$ rules to indicate that $r_1, \ldots, r_k$ are equivalence relations; $|\mathcal{I}|$ rules to indicate that nodes sharing an individual marker are co-identical; $2 \times |\top_R|$ rules to simulate normalization.

Using our LG rules library, that use types and markers as "first-class objects" of the model, we need only a constant number of rules (11), before declaring $r_1, \ldots, r_k$ as subtypes of EQUIVALENCE.

But there is another benefit: using the rules defined in [2], one must add new rules each time a new type or a new marker is added to the support, otherwise reasonings are not complete. And keeping trace of all the rules that must be added when updating the support in a complex modelization soon becomes difficult. Now, reification allows to express properties on types and markers. These properties can be encoded in libraries of LG rules, which implement various SGs semantics, in such a way that it can be invisible to the end-user. The interest is that these properties can be *inherited*, and do not need to be encoded again by the end-user. Reification of types and markers indeed adds expressiveness, as well in terms of succintness (factorization of rules) as in terms of robustness (resistance to changes of data).

Finally, we point out that our simulation is complete only for LGs verifying the "unique type and marker" assumption (see Rem. 1). The problem is we have represented knowledge "two entities have the same type" by "the types of these two entities are represented by the same node". Should we want to overcome this limitation, we could define a "meta-relation" expressing that knowledge.

7.2 An Extended Simple Graphs Model

The SG model imposes strong syntactic conditions on co-reference links: for example, fusionning two concept nodes having different types or different individual markers is impossible in the model, hence the draconian constraints on SG rules, designed in such a way that never will co-referent nodes have different types or markers. But the deduction mechanism of the {LG, rules} model, applied to the simulation of a SG, ignores such constraints. Then a natural question is: "what happens if we remove all constraints on co-reference, and make reasonings on the simulation of these objects? How does the result of these reasonings translate back to the SG model?"

The first problem is to handle the case of co-identical nodes having different types or markers. The set of rules $\mathcal{R}_d$ presented in Fig. 10 are an answer to that problem. They indicate that if two nodes having type t and t' represent the same entity, then the type of this entity is some subtype of t and t'; and the marker of a node can be considered as a marker for all its co-ident nodes. We consider $\mathcal{R}$ the library of LG rules consisting of $\mathcal{R}_s \cup \mathcal{R}_c \cup \mathcal{R}_n \cup \mathcal{R}_d$. We can extend the {SG, rules} model by dropping off all constraints on co-reference, and simulate it (using the transformation Θ) into the {LG, rules} model with the library $\mathcal{R}$.

Let us now outline an extended simple graph (ESG) model, that allows one to translate back those reasonings. An ESG $G = (V_C, V_R, E, \text{label}, \text{co-ref})$, defined on a support $\mathcal{S}$, can be seen as a simple graph where:

- the type of a concept node is a non-empty *subset* of T_C;
- the marker of a concept node is a *subset* of $\mathcal{I}$ or the generic marker $*$;
- there is no constraint on concept nodes that can be declared co-referent;
- two *individual* concept nodes x, y are co-ident if $\text{marker}(x) \cap \text{marker}(y) \neq \emptyset$.

Intuitively, a set of types $\{t_1, \ldots, t_k\}$ (as used, by example, in [1]) can be seen as the conjunction of types $t_1 \sqcap \cdots \sqcap t_k$, and a set of individual markers as aliases for the same entity. The partial ordering $\leq_e$ on labels is used for projection:

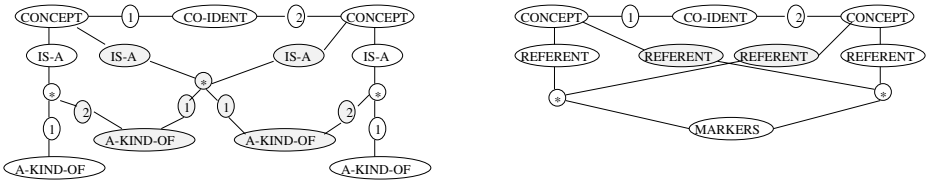


Fig. 10. Library $\mathcal{R}_d$ of LG-rules handling co-identity without restrictions

- Let t and t' be two types. Then $t \leq_e t'$ iff $\forall t'_i \in t',\ \exists t_j \in t$ such that $t_j \leq t'_i$.
- Let m and m' be two individual markers. Then $m \leq_e m'$ (and $m' \leq_e m$) iff $m \cap m' \neq \emptyset$. Moreover, for every marker m, $m \leq_e *$.

An ESG is in *normal form* if all its co-identity classes are trivial. An ESG is put into normal form by fusionning co-identical nodes. The label resulting

from the fusion of `[t: m]` and `[t': m']` is `[t ∪ t': M]`, where $M = m \cup m'$ if both markers are individual, $*$ if both are generic, otherwise M is the individual marker m or m'. The deduction system in the {ESG, rules} model is based upon *normal derivation*. Sound and complete FOL semantics can be given, using the semantics of the equivalent {LG, rules} model, or directly extending Φ (see [3]).

This example shows that, not only simulation adds robustness with respect to changes in data (e.g. changes in the support, as discussed above) but also with respect to changes in the model. We believe that a {LG, rules}-based interpreter, provided with macros that describe syntactic translations of graphs, can be the basis of a good prototyping and development tool for different graph-based reasoning models.

References

1. Franz Baader, Ralf Molitor, and Stephan Tobies. Tractable and decidable fragments of conceptual graphs. In *Proc. of ICCS'99*, pages 480–493. Springer, 1999.
2. Jean-François Baget. A Simulation Of Co-Identity with Rules in Simple and Nested Graphs. In *Proc. of ICCS'99*, pages 442–455. Springer, 1999.
3. Jean-François Baget. Extending the CG Model by Simulations. Research report, LIRMM, 2000.
4. Michel Chein and Marie-Laure Mugnier. Conceptual Graphs are also Graphs. Research Report, 1995.
5. G. Gogic, H. Kautz, C. Papadimitriou, and B. Selman. The Comparative Linguistics of Knowledge Representation. In *Proc. of IJCAI'95*, pages 862–869, 1995.
6. Gary G. Hendrix. Encoding Knowledge in Partitioned Networks. In *Associative Networks*, pages 51–92. Academic Press, 1979.
7. D. Kayser. *La représentation des connaissances.* HERMES, Paris, 1997.
8. E. Salvat and M-L. Mugnier. Sound and Complete Forward and Backward Chainings of Graph Rules. In *Proc. of ICCS'96*. Springer, 1996.
9. Stuart C. Shapiro. A net structure for semantic information storage, deduction and retrieval. In *Advance Papers of IJCAI'71*, pages 512–523, 1971.
10. G. Simonet. Two FOL Semantics for Simple and Nested Conceptual Graphs. In *Proc. of ICCS'98*. Springer, 1998.
11. John. F. Sowa. *Conceptual structures : Information processing in mind and machine.* Addison-Wesley, 1984.

Building and Structuring Description Logic Knowledge Bases Using Least Common Subsumers and Concept Analysis*

Franz Baader and Ralf Molitor

Lehr- und Forschungsgebiet Theoretische Informatik, RWTH Aachen,
Ahornstraße 55, 52074 Aachen, Germany,
{baader,molitor}@informatik.rwth-aachen.de

Abstract. Given a finite set $\mathcal{C} := \{C_1, \ldots, C_n\}$ of description logic concepts, we are interested in computing the subsumption hierarchy of all least common subsumers of subsets of $\mathcal{C}$. This hierarchy can be used to support the bottom-up construction and the structuring of description logic knowledge bases. The point is to compute this hierarchy without having to compute the least common subsumer for all subsets of $\mathcal{C}$. We will show that methods from formal concept analysis developed for computing concept lattices can be employed for this purpose.

1 Introduction

Knowledge representation systems based on description logics (DL) can be used to describe the knowledge of an application domain in a structured and formally well-understood way. Traditionally, the knowledge base of a DL system is built in a *top-down* manner. First, the relevant concepts of the application domain (its terminology) are formalized by *concept descriptions*, i.e., expressions that are built from atomic concepts (unary predicates) and atomic roles (binary predicates) using the concept constructors provided by the DL language. In a second step, these concept descriptions are then used to specify properties of *objects* occurring in the domain. The *standard inference procedures* provided by DL systems (like computing the subsumption hierarchy between concepts, and testing for implied instance relationships between objects and concepts) support this traditional approach to building DL knowledge bases.

The main problem with the top-down approach is that it presupposes a knowledge engineer that is both, an expert in description logics, and in the application domain. Less experienced knowledge engineers encounter (at least one of) the following two problems: (1) it is often not clear which are the relevant concepts in an application; and (2) even if it is clear which (intuitive) concepts should be introduced, it is sometimes difficult to come up with formal definitions

* This work was partially supported by the *Deutsche Forschungsgemeinschaft* Grant No. GRK 185/3-98.

B. Ganter and G.W. Mineau (Eds.): ICCS 2000, LNAI 1867, pp. 292–305, 2000.
© Springer-Verlag Berlin Heidelberg 2000

of these concepts in the given DL language. It has turned out that providing adequate support for overcoming these problems requires additional (non-standard) inference procedures for DL systems.

In [2,3], we propose to support the construction of DL knowledge bases in a *bottom-up* fashion: instead of directly defining a new concept, the knowledge engineer introduces several typical examples as objects, which are then automatically generalized into a concept description by the system. This description is offered to the knowledge engineer as a possible candidate for a definition of the concept. The task of computing such a concept description can be split into two subtasks: computing the most specific concepts of the given objects, and then computing the least common subsumer of these concepts. The *most specific concept* (msc) of an object o (the *least common subsumer* (lcs) of concept descriptions $C_1, \ldots, C_n$) is the most specific concept description C expressible in the given DL language that has o as an instance (that subsumes $C_1, \ldots, C_n$). The problem of computing the lcs and (to a limited extent) the msc has already been investigated in the literature [5,9,2,3].

Here, we will address an additional problem that occurs in the bottom-up approach: obviously, the *choice of the examples* is crucial for the quality of the result. If the examples are too similar, the resulting concept might be too specific. Conversely, if the examples are too different, the resulting concept is likely to be too general. Our goal in this paper is to support the process of choosing an appropriate set of objects as examples. Assume that $C_1, \ldots, C_n$ are the most specific concepts of a given collection of objects $o_1, \ldots, o_n$, and that we intend to use subsets of this collection for constructing new concepts. In order to avoid obtaining concepts that are too general or too specific, it would be good to know the position of the corresponding lcs in the subsumption hierarchy of all least common subsumers of subsets of $\{C_1, \ldots, C_n\}$. Since there are exponentially many subsets to be considered, and (depending on the DL language) both, computing the lcs and testing for subsumption, can be expensive operations, we want to obtain complete information on how this hierarchy looks like without computing the least common subsumers of all subsets of $\{C_1, \ldots, C_n\}$, and without explicitly making all the subsumption tests between these least common subsumers.

This is where methods from *formal concept analysis* [11] come into play. We shall show that the *dual* of the *attribute exploration* algorithm [10,16,11] (called *object exploration* in the following) can be adapted to our purposes. To be more precise, given a formal context, the attribute (object) exploration algorithm computes the concept lattice as well as a minimal implication base, the so-called (dual) Duquenne-Guigues base [8], of this context. For a given set of concept descriptions $\{C_1, \ldots, C_n\}$, we will define a formal context that has the property that its concept lattice is isomorphic to the subsumption hierarchy of all least common subsumers of subsets of $\{C_1, \ldots, C_n\}$. Thus, standard tools for drawing concept lattices [19] can be employed to show the hierarchy to the user. In addition, the dual Duquenne-Guigues base provides a small representation of this hierarchy. From this representation, all subsumption relationships can be

deduced in time linear in the size of the representation. To compute the concept lattices and the Duquenne-Guigues base, the exploration algorithm employs an algorithm for computing the lcs and a decision procedure for subsumption as sub-procedures. Although, in the worst case, an exponential number of calls to these sub-procedures cannot be avoided, experiences from applications of formal concept analysis [18] indicate that the exploration algorithm usually does a lot better in practice.

Another application for this method is *structuring of DL knowledge bases.* DL knowledge bases encountered in applications are often rather broad in the sense that a given concept can have a large number of direct successors in the subsumption hierarchy. Narrower and thus deeper hierarchies would be more convenient when browsing the knowledge base along the hierarchy, and they would make searching more efficient. Given a concept C with direct sub-concepts $\{C_1, \ldots, C_n\}$, one could use least common subsumers of selected subsets to provide a better structuring of the knowledge base by inserting additional layers between C and its sub-concepts $C_1, \ldots, C_n$. Again, knowing the hierarchy of all least common subsumers of subsets of $\{C_1, \ldots, C_n\}$ can support the knowledge engineer in choosing the right subsets.

In the next section, we introduce the DL languages $\mathcal{ALE}$ and $\mathcal{ALN}$, define the subsumption problem as well as the notion "least common subsumer", and recall results from the literature for deciding subsumption and computing the least common subsumer. In Section 3, we introduce as many of the basic notions of formal concept analysis as are necessary for our purposes. In particular, we sketch the object exploration algorithm. Section 4 applies this technique to our problem of computing the subsumption hierarchy between least common subsumers of subsets of a given collection of concepts. In Section 5, we provide some experimental results that support our thesis that object exploration is an appropriate tool for this purpose. Section 6 concludes with some comments on related and future work.

2 The Description Logics $\mathcal{ALE}$ and $\mathcal{ALN}$

For the purpose of this paper, it is sufficient to restrict the attention to the formalism for defining concept descriptions (i.e., we need not introduce TBoxes, which allow to abbreviate complex descriptions by names, and ABoxes, which introduce objects and their properties). In order to define concepts in a DL knowledge base, one starts with a set N_C of concept names (unary predicates) and a set N_R of role names (binary predicates), and defines more complex *concept descriptions* using the operations provided by the DL language of the particular system. In this paper, we consider the languages $\mathcal{ALE}$ and $\mathcal{ALN}$,[1] which allow for concept descriptions built from the indicated subsets of the constructors

[1] It should be noted, however, that the methods developed in this paper apply to arbitrary concept descriptions languages, as long as they are equipped with a subsumption algorithm and an algorithm for computing least common subsumers.

Table 1. Syntax and semantics of concept descriptions.

name of constructor	Syntax	Semantics	$\mathcal{ALE}$	$\mathcal{ALN}$
top-concept	$\top$	$\Delta^{\mathcal{I}}$	x	x
bottom-concept	$\bot$	$\emptyset$	x	x
primitive negation	$\neg P$	$\Delta^{\mathcal{I}} \setminus P^{\mathcal{I}}$	x	x
conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$	x	x
value restriction	$\forall r.C$	$\{x \in \Delta^{\mathcal{I}} \mid \forall y : (x,y) \in r^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$	x	x
existential restriction	$\exists r.C$	$\{x \in \Delta^{\mathcal{I}} \mid \exists y : (x,y) \in r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$	x	
number restriction	$(\geq n\ r)$	$\{x \in \Delta^{\mathcal{I}} \mid \#\{y \in \Delta^{\mathcal{I}} \mid (x,y) \in r^{I}\} \geq n\}$		x
number restriction	$(\leq n\ r)$	$\{x \in \Delta^{\mathcal{I}} \mid \#\{y \in \Delta^{\mathcal{I}} \mid (x,y) \in r^{I}\} \leq n\}$		x

shown in Table 1. In this table, P stands for a concept name, r for a role name, n for a nonnegative integer, and C, D for arbitrary concept descriptions.

The semantics of concept descriptions is defined in terms of an *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$. The domain $\Delta^{\mathcal{I}}$ of $\mathcal{I}$ is a non-empty set and the interpretation function $\cdot^{\mathcal{I}}$ maps each concept name $P \in N_C$ to a set $P^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ and each role name $r \in N_R$ to a binary relation $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. The extension of $\cdot^{\mathcal{I}}$ to arbitrary concept descriptions is inductively defined, as shown in the third column of Table 1.

One of the most important traditional inference services provided by DL-systems is computing subconcept/superconcept relationships (so-called *subsumption* relationships) between concept descriptions. The concept description C_2 *subsumes* the concept description C_1 ($C_1 \sqsubseteq C_2$) iff $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$ for all interpretations $\mathcal{I}$; C_2 is *equivalent* to C_1 ($C_1 \equiv C_2$) iff $C_1 \sqsubseteq C_2$ and $C_2 \sqsubseteq C_1$. The subsumption relation $\sqsubseteq$ is a quasi order (i.e., reflexive and transitive), but in general not a partial order since it need not be antisymmetric (i.e., there may exist equivalent description that are not syntactically equal). As usual, the quasi order $\sqsubseteq$ induces a partial order $\sqsubseteq_{\equiv}$ on the equivalence classes of concept descriptions:

$$[C_1]_{\equiv} \sqsubseteq_{\equiv} [C_2]_{\equiv} \quad \text{iff} \quad C_1 \sqsubseteq C_2,$$

where $[C_i]_{\equiv} := \{D \mid C_i \equiv D\}$ is the equivalence class of C_i ($i = 1, 2$). When talking about the *subsumption hierarchy* of a set of descriptions, one means this induced partial order.

Deciding subsumption between $\mathcal{ALN}$-concept descriptions is polynomial [4], whereas the subsumption problem for $\mathcal{ALE}$ is NP-complete [6].

In addition to subsumption, we are interested in the non-standard inference problem of computing the least common subsumer of concept descriptions.

Definition 1 *Given $n \geq 2$ concept descriptions $C_1, \ldots, C_n$ in a DL language $\mathcal{L}$, the concept description C of $\mathcal{L}$ is an lcs of $C_1, \ldots, C_n$ ($C = \mathsf{lcs}(C_1, \ldots, C_n)$) iff (i) $C_i \sqsubseteq C$ for all $1 \leq i \leq n$, and (ii) C is the least concept description with this property, i.e., if C' satisfies $C_i \sqsubseteq C'$ for all $1 \leq i \leq n$, then $C \sqsubseteq C'$.*

As an example, consider the $\mathcal{ALE}$-concept descriptions

$$C := \exists \mathsf{has\text{-}child}.\top \sqcap \forall \mathsf{has\text{-}child}.(\mathsf{Male} \sqcap \mathsf{Doctor}) \quad \text{and}$$

$$D := \exists \mathsf{has\text{-}child}.(\mathsf{Male} \sqcap \mathsf{Mechanic}) \sqcap \exists \mathsf{has\text{-}child}.(\mathsf{Female} \sqcap \mathsf{Doctor})$$

respectively describing parents whose children are all male doctors, and parents having a son that is a mechanic and a daughter that is a doctor. The lcs of C and D is given by the $\mathcal{ALE}$-concept description

$$\mathsf{lcs}(C, D) = \exists \mathsf{has\text{-}child}.\mathsf{Male} \sqcap \exists \mathsf{has\text{-}child}.\mathsf{Doctor}.$$

It describes all parents having at least one son and at least one child that is a doctor (see [3] for an algorithm for computing such an lcs).

Depending on the DL under consideration, the lcs of two or more descriptions need not always exist, but if it exists, then it is unique up to equivalence. It is also easy to see that one can restrict the attention to the problem of computing the lcs of two concept descriptions, since the lcs of $n > 2$ descriptions can be obtained by iterated application of the binary lcs operation.

In [3], we have shown that the lcs of two $\mathcal{ALE}$-concept descriptions always exists and that it can effectively be computed; however, the size of the lcs can be exponential in the size of the input descriptions. For $\mathcal{ALN}$, the lcs of two descriptions also always exists and it can be computed in polynomial time. In addition, the size of the lcs is polynomial in the size of the input descriptions, even if one considers $n > 2$ descriptions (these results for $\mathcal{ALN}$ can easily be obtained by restricting the results in [2] to the acyclic case).

3 Formal Concept Analysis

We shall introduce only those notions and results from formal concept analysis that are necessary for our application. We will describe how the object exploration algorithm works, but note that explaining why it works is beyond the scope of this paper (see [11] for more information on formal concept analysis).

Definition 2 *A* formal context *is a triple $\mathcal{K} = (\mathcal{O}, \mathcal{P}, \mathcal{S})$, where $\mathcal{O}$ is a set of objects, $\mathcal{P}$ is a set of attributes (or properties), and $\mathcal{S} \subseteq \mathcal{O} \times \mathcal{P}$ is a relation that connects each object o with the attributes satisfied by o.*

Let $\mathcal{K} = (\mathcal{O}, \mathcal{P}, \mathcal{S})$ be a formal context. For a set of objects $A \subseteq \mathcal{O}$, the *intent* A' of A is the set of attributes that are satisfied by all objects in A, i.e.,

$$A' := \{p \in \mathcal{P} \mid \forall a \in A\colon (a, p) \in \mathcal{S}\}.$$

Similarly, for a set of attributes $B \subseteq \mathcal{P}$, the *extent* B' of B is the set of objects that satisfy all attributes in B, i.e.,

$$B' := \{o \in \mathcal{O} \mid \forall b \in B\colon (o, b) \in \mathcal{S}\}.$$

It is easy to see that, for $A_1 \subseteq A_2 \subseteq \mathcal{O}$ (resp. $B_1 \subseteq B_2 \subseteq \mathcal{P}$), we have

- $A_2' \subseteq A_1'$ (resp. $B_2' \subseteq B_1'$),
- $A_1 \subseteq A_1''$ and $A_1' = A_1'''$ (resp. $B_1 \subseteq B_1''$ and $B_1' = B_1'''$).

A *formal concept* is a pair (A, B) consisting of an extent $A \subseteq \mathcal{O}$ and an intent $B \subseteq \mathcal{P}$ such that $A' = B$ and $B' = A$. Such formal concepts can be hierarchically ordered by inclusion of their extents, and this order induces a complete lattice, the *concept lattice* of the context. Given a formal context, the first step for analyzing this context is usually to compute the concept lattice.

The following are easy consequences of the definition of formal concepts and the properties of the $\cdot'$ operation mentioned above:

Lemma 3 *All formal concepts are of the form* (A'', A') *for a subset* A *of* $\mathcal{O}$*, and any such pair is a formal concept. In addition,* $(A_1'', A_1') \leq (A_2'', A_2')$ *iff* $A_2' \subseteq A_1'$.

Thus, if the context is finite, the concept lattices can in principle be computed by enumerating the subsets A of $\mathcal{O}$, and applying the operations $\cdot'$ and $\cdot''$. However, this naïve algorithm is usually very inefficient. In many applications [18], one has a large (or even infinite) set of objects, but only a relatively small set of attributes. In such a situation, Ganter's *attribute exploration* algorithm [10,11] has turned out to be an efficient approach for computing the concept lattice.

In the application considered in this paper, we are faced with the *dual* situation: the set of attributes will be the *infinite* set of all possible concept descriptions of the DL under consideration, and the set of objects will be the *finite* collection of concept descriptions for which we want to compute the subsumption hierarchy of least common subsumers. Consequently, we must dualize Ganter's algorithm and the notions on which it depends. Alternatively, we could have considered the dual context (which is obtained by transposing the matrix corresponding to $\mathcal{S}$) and employed the usual attribute exploration for this context. The correctness of the dual version is an immediate consequence of the fact that its results coincide with the results of the usual algorithm on the dual context.

Object Exploration

Before we can describe the dual version of Ganter's algorithm, called object exploration in the following, we must introduce some notation. The most important notion for the algorithm is the one of an implication between sets of objects. Intuitively, such an implication $A_1 \to A_2$ holds if any attribute satisfied by all elements of A_1 is also satisfied by all elements of A_2.

Definition 4 *Let* $\mathcal{K}$ *be a formal context and* A_1*,* A_2 *be subsets of* $\mathcal{O}$*. The* object implication $A_1 \to A_2$ *holds in* $\mathcal{K}$ *(*$\mathcal{K} \models A_1 \to A_2$*) iff* $A_1' \subseteq A_2'$*. An attribute* p violates *the implication* $A_1 \to A_2$ *iff* $p \in A_1' \setminus A_2'$.

It is easy to see that an implication $A_1 \to A_2$ holds in $\mathcal{K}$ iff $A_2 \subseteq A_1''$. In particular, given a set of objects A, the implications $A \to (A'' \setminus A)$ always holds in $\mathcal{K}$. We denote the set of all object implications that hold in $\mathcal{K}$ by $Imp_{\mathcal{O}}(\mathcal{K})$. This set can be very large, and thus one is interested in (small) generating sets.

Definition 5 *Let $\mathcal{J}$ be a set of object implications, i.e., the elements of $\mathcal{J}$ are of the form $A_1 \to A_2$ for sets of objects $A_1, A_2 \subseteq \mathcal{O}$. For a subset A of $\mathcal{O}$, the* implication hull *of A with respect to $\mathcal{J}$ is denoted by $\mathcal{J}(A)$. It is the smallest subset H of $\mathcal{O}$ such that*

- *$A \subseteq H$, and*
- *$A_1 \to A_2 \in \mathcal{J}$ and $A_1 \subseteq H$ imply $A_2 \subseteq H$.*

The set of object implications generated by $\mathcal{J}$ *consists of all implications $A_1 \to A_2$ such that $A_2 \subseteq \mathcal{J}(A_1)$. It will be denoted by $Cons(\mathcal{J})$. We say that a set of implications $\mathcal{J}$ is a* base *of $Imp_{\mathcal{O}}(\mathcal{K})$ iff $Cons(\mathcal{J}) = Imp_{\mathcal{O}}(\mathcal{K})$ and no proper subset of $\mathcal{J}$ satisfies this property.*

If $\mathcal{J}$ is a base for $Imp_{\mathcal{O}}(\mathcal{K})$, then it can be shown that $A'' = \mathcal{J}(A)$ for all $A \subseteq \mathcal{O}$. The implication hull $\mathcal{J}(A)$ of a set of objects A can be computed in time linear in the size of $\mathcal{J}$ and A using , for example, methods for deciding satisfiability of sets of propositional Horn clauses [7]. Consequently, given a base $\mathcal{J}$ for $Imp_{\mathcal{O}}(\mathcal{K})$, any question of the form "$A_1 \to A_2 \in Imp_{\mathcal{O}}(\mathcal{K})$?" can be answered in time linear in the size of $\mathcal{J} \cup \{A_1 \to A_2\}$.

There may exist different implication bases of $Imp_{\mathcal{O}}(\mathcal{K})$, and not all of them need to be of minimal cardinality. A base $\mathcal{J}$ of $Imp_{\mathcal{O}}(\mathcal{K})$ is called *minimal base* iff no base of $Imp_{\mathcal{O}}(\mathcal{K})$ has a cardinality smaller than the cardinality of $\mathcal{J}$. Duquenne and Guigues have given a description of such a minimal base [8] for the dual case of attribute implications. Ganter's attribute exploration algorithm computes this minimal base as a by-product. In the following, we introduce the *dual Duquenne-Guigues base* and show how it can be computed using the object exploration algorithm.

The definition of the dual Duquenne-Guigues base given below is based on a modification of the closure operator $A \mapsto \mathcal{J}(A)$ defined by a set $\mathcal{J}$ of object implications. For a subset A of $\mathcal{O}$, the *implication pseudo-hull* of A with respect to $\mathcal{J}$ is denoted by $\mathcal{J}^*(A)$. It is the smallest subset H of $\mathcal{O}$ such that

- $A \subseteq H$, and
- $A_1 \to A_2 \in \mathcal{J}$ and $A_1 \subset H$ (strict subset) imply $A_2 \subseteq H$.

Given $\mathcal{J}$, the pseudo-hull of a set $A \subseteq \mathcal{O}$ can again be computed in time linear in the size of $\mathcal{J}$ and A (e.g., by adapting the algorithm in [7] appropriately). A subset A of $\mathcal{O}$ is called *pseudo-closed* in a formal context $\mathcal{K}$ iff $Imp_{\mathcal{O}}(\mathcal{K})^*(A) = A$ and $A'' \neq A$.

Definition 6 *The* dual Duquenne-Guigues base *of a formal context $\mathcal{K}$ consists of all object implications $A_1 \to A_2$ where $A_1 \subseteq \mathcal{O}$ is pseudo-closed in $\mathcal{K}$ and $A_2 = A_1'' \setminus A_1$.*

When trying to use this definition for actually *computing* the dual Duquenne-Guigues base of a formal context, one encounters two problems:

1. The definition of pseudo-closed refers to the set of all valid implications $Imp_{\mathcal{O}}(\mathcal{K})$, and our goal is to avoid explicitly computing all of them.

2. The closure operator $A \mapsto A''$ is used, and computing it via $A \mapsto A' \mapsto A''$ may not be feasible for a context with an infinite set of attributes.

Ganter solves the first problem by enumerating the pseudo-closed sets of $\mathcal{K}$ in a particular order, called lectic order. This order makes sure that it is sufficient to use the already computed part $\mathcal{J}$ of the base when computing the pseudo-hull. To define the lectic order, fix an arbitrary linear order on the set of objects $\mathcal{O} = \{o_1, \ldots, o_n\}$, say $o_1 < \cdots < o_n$. For all $j, 1 \leq j \leq n$, and $A_1, A_2 \subseteq \mathcal{O}$ we define

$$A_1 <_j A_2 \text{ iff } o_j \in A_2 \setminus A_1 \text{ and } A_1 \cap \{o_1, \ldots, o_{j-1}\} = A_2 \cap \{o_1, \ldots, o_{j-1}\}.$$

The lectic order $<$ is the union of all relations $<_j$ for $j = 1, \ldots, n$. It is a linear order on the powerset of $\mathcal{O}$. The lectic smallest subset of $\mathcal{O}$ is the empty set.

The second problem is solved by constructing an increasing chain of finite subcontexts of $\mathcal{K}$. The context $\mathcal{K}_i = (\mathcal{O}_i, \mathcal{P}_i, \mathcal{S}_i)$ is a *subcontext* of $\mathcal{K}$ iff $\mathcal{O}_i = \mathcal{O}$, $\mathcal{P}_i \subseteq \mathcal{P}$, and $\mathcal{S}_i = \mathcal{S} \cap (\mathcal{O} \times \mathcal{P}_i)$. The closure operator $A \mapsto A''$ is always computed with respect to the current finite subcontext $\mathcal{K}_i$. To avoid adding a wrong implication, an "expert" is asked whether the implication $A \to A'' \setminus A$ really holds in the whole $\mathcal{K}$. If it does not hold, the expert must provide a counterexample, i.e., an attribute p from $\mathcal{P} \setminus \mathcal{P}_i$ that violates the implication. This attribute is then added to the current context. Technically, this means that the expert must provide an attribute p, and must say which of the objects of $\mathcal{O}$ satisfy this attribute and which don't.

The following algorithm computes the set of all extents of formal concepts of $\mathcal{K}$ as well as the dual Duquenne-Guigues base of $\mathcal{K}$. The concept lattice is then given by the usual inclusion ordering between the extents.

Algorithm 7 (Object exploration) Initialization: *One starts with the empty set of object implications, i.e., $\mathcal{J}_0 := \emptyset$, the empty set of concept extents $\mathcal{E}_0 := \emptyset$, and the empty subcontext $\mathcal{K}_0$ of $\mathcal{K}$, i.e., $\mathcal{P}_0 := \emptyset$. The lectic smallest subset of $\mathcal{O}$ is $A_0 := \emptyset$.*

Iteration: *Assume that $\mathcal{K}_i$, $\mathcal{J}_i$, $\mathcal{E}_i$, and A_i ($i \geq 0$) are already computed. Compute A_i'' with respect to the current subcontext $\mathcal{K}_i$. Now the expert is asked whether the implication $A_i \to A_i'' \setminus A_i$ holds in $\mathcal{K}$.*[2]

If the answer is "no", then let $p_i \in \mathcal{P}$ be the counterexample provided by the expert. Let $A_{i+1} := A_i$, $\mathcal{J}_{i+1} := \mathcal{J}_i$, and let $\mathcal{K}_{i+1}$ be the subcontext of $\mathcal{K}$ with $\mathcal{P}_{i+1} := \mathcal{P}_i \cup \{p_i\}$.

If the answer is "yes", then $\mathcal{K}_{i+1} := \mathcal{K}_i$ and

$$(\mathcal{E}_{i+1}, \mathcal{J}_{i+1}) := \begin{cases} (\mathcal{E}_i, \mathcal{J}_i \cup \{A_i \to A_i'' \setminus A_i\}) & \text{if } A_i'' \neq A_i, \\ (\mathcal{E}_i \cup \{A_i\}, \mathcal{J}_i) & \text{if } A_i'' = A_i. \end{cases}$$

To find the new set A_{i+1}, we start with $j = n$, and test whether

$$(*) \quad A_i <_j \mathcal{J}_{i+1}^*((A_i \cap \{o_1, \ldots, o_{j-1}\}) \cup \{o_j\})$$

[2] If $A_i'' \setminus A_i = \emptyset$, then it is not really necessary to ask the expert because implications with empty right-hand side hold in any context.

holds. The index j is decreased until one of the following cases occurs:
(1) $j = 0$: In this case, $\mathcal{E}_{i+1}$ is the set of all concept extents and $\mathcal{J}_{i+1}$ the dual Duquenne-Guigues base of $\mathcal{K}$, and the algorithm stops.
(2) $(*)$ holds for $j > 0$: In this case, $A_{i+1} := \mathcal{J}^*_{i+1}((A_i \cap \{o_1, \ldots, o_{j-1}\}) \cup \{o_j\})$, and the iteration is continued.

4 Computing the Hierarchy of Least Common Subsumers

Given a finite set $\mathcal{O} := \{C_1, \ldots, C_n\}$ of concept descriptions, we are interested in the subsumption hierarchy between all least common subsumers of subsets of $\mathcal{O}$. For sets $A \subseteq \mathcal{O}$ of cardinality ≥ 2, we have already defined the notion $\mathsf{lcs}(A)$. We extend this notion to the empty set and singletons in the obvious way: $\mathsf{lcs}(\emptyset) := \bot$ and $\mathsf{lcs}(\{C_i\}) := C_i$.

Our goal is to compute the subsumption hierarchy between all concept descriptions $\mathsf{lcs}(A)$ for subsets A of $\mathcal{O}$ without explicitly computing all these least common subsumers. This is achieved by defining a formal context with objects $\mathcal{O}$ such that the concept lattice of this context is isomorphic to the subsumption hierarchy we are interested in.

Definition 8 *Given a DL language $\mathcal{L}$ and a finite set $\mathcal{O} := \{C_1, \ldots, C_n\}$ of $\mathcal{L}$-concept descriptions, the corresponding formal context $\mathcal{K}_{\mathcal{L}}(\mathcal{O}) = (\mathcal{O}, \mathcal{P}, \mathcal{S})$ is defined as follows:*

$$\begin{aligned}
&\mathcal{O} := \{C_1, \ldots, C_n\},\\
&\mathcal{P} := \{D \mid D \text{ is an } \mathcal{L}\text{-concept description}\},\\
&\mathcal{S} := \{(C, D) \mid C \sqsubseteq D\}.
\end{aligned}$$

As an easy consequence of the definition of $\mathcal{K}_{\mathcal{L}}(\mathcal{O})$ and of the lcs, we obtain that the intent of a set $A \subseteq \mathcal{O}$ is closely related to the lcs of this set:

Lemma 9 *Let A_1, A_2 be subsets of $\mathcal{O}$.*

1. *$A' = \{D \in \mathcal{P} \mid \mathsf{lcs}(A) \sqsubseteq D\}$;*
2. *$A'_1 \subseteq A'_2$ iff $\mathsf{lcs}(A_2) \sqsubseteq \mathsf{lcs}(A_1)$;*
3. *the implication $A_1 \to A_2$ holds in $\mathcal{K}_{\mathcal{L}}(\mathcal{O})$ iff $\mathsf{lcs}(A_2) \sqsubseteq \mathsf{lcs}(A_1)$.*

As an immediate consequence of 3. of this lemma, the dual Duquenne-Guigues base $\mathcal{J}$ of $\mathcal{K}_{\mathcal{L}}(\mathcal{O})$ yields a representation of all subsumption relationships of the form $\mathsf{lcs}(A_1) \sqsubseteq \mathsf{lcs}(A_2)$ for subsets A_1, A_2 of $\mathcal{O}$. Given this base $\mathcal{J}$, any question of the form "$\mathsf{lcs}(A_1) \sqsubseteq \mathsf{lcs}(A_2)$?" can then be answered in time linear in the size of $\mathcal{J} \cup \{A_1 \to A_2\}$. Another easy consequence of the lemma is that the concept lattice of $\mathcal{K}_{\mathcal{L}}(\mathcal{O})$ coincides with the subsumption hierarchy of all least common subsumers of subsets of $\mathcal{O}$.

Theorem 10 *The concept lattice of $\mathcal{K}_{\mathcal{L}}(\mathcal{O})$ is isomorphic to the subsumption hierarchy of all least common subsumers of subsets of $\mathcal{O}$.*

Proof. We define the mapping π from the formal concepts of $\mathcal{K}_{\mathcal{L}}(\mathcal{O})$ to the set of (equivalence classes of) least common subsumers of subsets of $\mathcal{O}$ as follows:

$$\pi(A, B) := [\mathsf{lcs}(A)]_{\equiv}.$$

For formal concepts $(A_1, B_1), (A_2, B_2)$ we have $(A_1, B_1) \leq (A_2, B_2)$ iff $A_1 \subseteq A_2$ iff $A_2' \subseteq A_1'$ iff $\mathsf{lcs}(A_1) \sqsubseteq \mathsf{lcs}(A_2)$. As an easy consequence we obtain that π is order preserving (and thus also injective): $(A_1, B_1) \leq (A_2, B_2)$ iff $[\mathsf{lcs}(A_1)]_{\equiv} \sqsubseteq_{\equiv} [\mathsf{lcs}(A_2)]_{\equiv}$.

It remains to be shown that π is surjective as well. Let A be an arbitrary subset of $\mathcal{O}$. We must show that $[\mathsf{lcs}(A)]_{\equiv}$ can be obtained as an image of the mapping π. By Lemma 3, (A'', A') is a formal concept, and thus it is sufficient to show that $\mathsf{lcs}(A) \equiv \mathsf{lcs}(A'')$. Obviously, $A \subseteq A''$ implies $\mathsf{lcs}(A) \sqsubseteq \mathsf{lcs}(A'')$ (by definition of the lcs). To see the converse, note that, for all $C_i \in \mathcal{O}$, we have

$$\begin{aligned} C_i \in A'' \quad &\text{iff} \quad C_i \sqsubseteq D \text{ for all } D \in A' && \text{(def. of } \cdot' \text{ and } \mathcal{K}_{\mathcal{L}}(\mathcal{O})) \\ &\text{iff} \quad C_i \sqsubseteq D \text{ for all } D \text{ such that } \mathsf{lcs}(A) \sqsubseteq D && \text{(Lemma 9)} \\ &\text{iff} \quad C_i \sqsubseteq \mathsf{lcs}(A). && \text{(def. of the lcs)} \end{aligned}$$

Obviously, this implies $\mathsf{lcs}(A'') \sqsubseteq \mathsf{lcs}(A)$. □

If we want to apply Algorithm 7 to compute the concept lattice and the dual Duquenne-Guigues base, we need an "expert" for the context $\mathcal{K}_{\mathcal{L}}(\mathcal{O})$. This expert must be able to answer the questions asked by the object exploration algorithm, i.e., given an object implication $A_1 \to A_2$, it must be able to decide whether this implication holds in $\mathcal{K}_{\mathcal{L}}(\mathcal{O})$. If the implication does not hold, it must be able to compute a counterexample, i.e., an attribute $p \in A_1' \setminus A_2'$.

If the language $\mathcal{L}$ is such that the lcs is computable and subsumption is decidable (which is, e.g., the case for $\mathcal{L} = \mathcal{ALE}$ or $\mathcal{L} = \mathcal{ALN}$), then we can implement such an expert.

Lemma 11 *Given a subsumption algorithm for $\mathcal{L}$ as well as an algorithm for computing the lcs of a finite set of $\mathcal{L}$-concept descriptions, these algorithms can be used to obtain an "expert" for the context $\mathcal{K}_{\mathcal{L}}(\mathcal{O})$.*

Proof. First, we show how to decide whether a given object implication $A_1 \to A_2$ holds in $\mathcal{K}_{\mathcal{L}}(\mathcal{O})$ or not. By Lemma 9, we know that $A_1 \to A_2$ holds in $\mathcal{K}_{\mathcal{L}}(\mathcal{O})$ iff $\mathsf{lcs}(A_2) \sqsubseteq \mathsf{lcs}(A_1)$. Obviously, $\mathsf{lcs}(A_2) \sqsubseteq \mathsf{lcs}(A_1)$ iff $C_i \sqsubseteq \mathsf{lcs}(A_1)$ for all $C_i \in A_2$. Thus, to answer the question "$A_1 \to A_2$?", we first compute $\mathsf{lcs}(A_1)$ and then use the subsumption algorithm to test whether $C_i \sqsubseteq \mathsf{lcs}(A_1)$ holds for all $C_i \in A_2$.

Second, assume that $A_1 \to A_2$ does not hold in $\mathcal{K}_{\mathcal{L}}(\mathcal{O})$, i.e., $\mathsf{lcs}(A_2) \not\sqsubseteq \mathsf{lcs}(A_1)$. We claim that $\mathsf{lcs}(A_1)$ is a counterexample, i.e., $\mathsf{lcs}(A_1) \in A_1'$ and $\mathsf{lcs}(A_1) \notin A_2'$. This is an immediate consequence of the facts that $A_i' = \{D \in \mathcal{P} \mid \mathsf{lcs}(A_i) \sqsubseteq D\}$ $(i = 1, 2)$ and that $\mathsf{lcs}(A_2) \not\sqsubseteq \mathsf{lcs}(A_1)$.

Of this counterexample, Algorithm 7 really needs the column corresponding to this attribute in the matrix corresponding to $\mathcal{S}$. This column can easily be computed using the subsumption algorithm: for each $C_i \in \mathcal{O}$, we use the subsumption algorithm to test whether $C_i \sqsubseteq \mathsf{lcs}(A)$ holds or not. □

Using this expert, an application of Algorithm 7 yields

- all extents of formal concepts of $\mathcal{K}_{\mathcal{L}}(\mathcal{O})$, and thus the concept lattice of $\mathcal{K}_{\mathcal{L}}(\mathcal{O})$, which coincides with the subsumption hierarchy of all least common subsumers of subsets of $\mathcal{O}$ (by Theorem 10);
- the dual Duquenne-Guigues base of $\mathcal{K}_{\mathcal{L}}(\mathcal{O})$, which yields a compact representation of this hierarchy (by 3. of Lemma 9); and
- a finite subcontext of $\mathcal{K}_{\mathcal{L}}(\mathcal{O})$ that has the same concept extents as $\mathcal{K}_{\mathcal{L}}(\mathcal{O})$ and the same $\cdot''$ operation on sets of objects.

Using the output of Algorithm 7, one can then employ the usual tools for drawing concept lattices [19] in order to present the subsumption hierarchy of all least common subsumers of subsets of $\mathcal{O}$ to the knowledge engineer.

5 First Experimental Results

In the previous section, we have shown that the object exploration algorithm can be used to compute the hierarchy of least common subsumers of a given set of concept descriptions. What remains is to analyze whether object exploration really is a good approach for solving this task. Our reason for trying it in the first place was that computing this hierarchy is the same as computing a certain concept lattice (as shown above), and that Ganter's algorithm is known to be a very good method for doing this. The problem with this generic argument in favour of object exploration is, of course, that we consider a very specific context, and that it might well be that, for this context, object exploration is not the best thing to do. The first experimental results that will be described below are, however, rather encouraging.

We intend to use the bottom-up construction of knowledge bases in a chemical process engineering application [13,15], where the knowledge base describes standard building blocks of process models (such as certain types of reactors). Currently, this knowledge base consists of about 600 definitions of building blocks, which we translate into $\mathcal{ALE}$-concept descriptions.

In order to test the object exploration algorithm, we have taken 7 descriptions of reactors of a similar type, which the process engineers considered to be good examples for generating a new concept. These descriptions were translated into $\mathcal{ALE}$-concept descriptions $R_1, \ldots, R_7$, and we applied the object exploration algorithm to this set of objects. The resulting concept lattice, which coincides with the hierarchy of all least common subsumers of subsets of $\mathcal{O} := \{R_1, \ldots, R_7\}$, is depicted in Figure 1. The top concept corresponds to the lcs obtained from the whole set of examples, and the bottom concept is the lcs obtained from the empty set, i.e., the description $\bot$. The node labelled $\mathsf{lcs}(i_1 \ldots i_m)$ corresponds to the formal concept with extent $R_{i_1}, \ldots, R_{i_m}$, and thus to $\mathsf{lcs}(R_{i_1}, \ldots, R_{i_m})$. Note that in many cases $\mathsf{lcs}(R_{i_1}, \ldots, R_{i_m})$ can also be obtained as the lcs of a strict subset of $\{R_{i_1}, \ldots, R_{i_m}\}$. This can be easily seen by using the least-upper-bound operation of the concept lattice. For example, $\mathsf{lcs}(R_i, R_7) = \mathsf{lcs}(R_1, \ldots, R_7)$ for all $i, 1 \leq i \leq 6$.

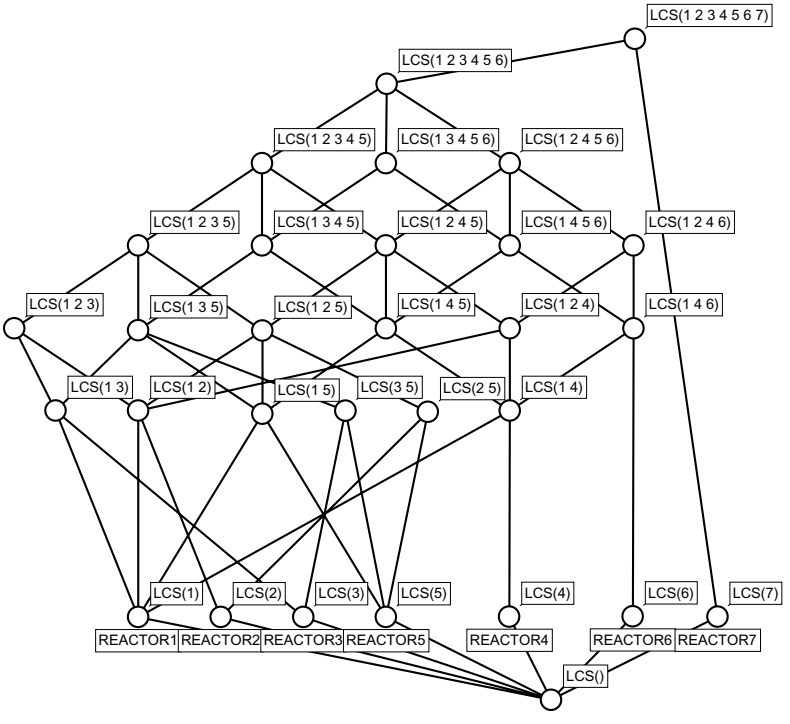


Fig. 1. The hierarchy of least common subsumers of seven reactor descriptions.

Statistical information: The Duquennes-Guigues base of the context consists of 15 implications, and the concept lattice of 30 formal concepts. If we subtract the trivial least common subsumers $\bot, R_1, \ldots, R_7$ as well as $\mathsf{lcs}(R_1, \ldots, R_7)$, which turned out to be equivalent to an already existing description, we end up with 21 candidates for new concepts. Of these 21 interesting least common subsumers, only 10 have explicitly been computed during the exploration.

During the calls of the "expert", 255 subsumption tests and 25 n-ary lcs operations have been executed. Because we re-used already computed least common subsumers, the 25 n-ary lcs operations only required 25 binary lcs operations. The number of counterexamples computed by the expert was also 25.

Finally, we measured the time needed for executing the interesting subtasks, namely computing the lcs, testing subsumption, and realizing the overhead introduced by the object exploration algorithm (e.g., computing the $\cdot''$ operation, the pseudo-hull, etc). It turned out that more than 84% of the time was used for computing least common subsumers, 15% for subsumption tests, and less than 1% for the rest. This shows that, at least for this small example, the exploration algorithm does not introduce any measurable overhead. The fact that computing the lcs needed a lot more time than testing subsumption is probably due to the fact that we used a highly optimized subsumption algorithm [12], but only a first prototypical implementation of the lcs algorithm.

What can be learned from the concept lattice? Two important facts about the reactor descriptions can be read off immediately. First, there is no subsumption relationship between any of the 7 concepts since all singleton sets occur as extents. Second, Reactor 7 is quite different from the other reactors since its lcs with any of the others yields a very general concept description. Thus, it should not be used for generating new concepts together with the other ones. In fact, a closer look at R_7 revealed that, though it describes a reactor of a type similar to that of the other ones, this description was given on a completely different level of abstraction.

Next, let us consider the question of which of the least common subsumers occurring in the lattice appear to be good candidates for providing an interesting new concept. First, the lcs of the whole set is ruled out since it involves Reactor 7, which does not fit well with the other examples (see above). Second, in order to avoid concepts that are too specific, least common subsumers that do not cover more than half of the reactors should also be avoided. If we use these two criteria, then we are left with 9 candidates (the formal concepts with extents of cardinality 4, 5, and 6), which is a number of concepts that can well be inspected by the process engineer. In our example, the 5 least common subsumers on the first layer of these interesting candidates (the formal concepts with extents of cardinality 4) turned out to be the most promising, though this must still be checked in more detail with the process engineers.

6 Related and Future Work

The idea of using tools from formal concept analysis in description logics is not new. In [1], the attribute exploration algorithm was used to compute a small representation of the subsumption hierarchy of all conjunctions of concepts defined in a terminology, and in [17] this approach was extended such that it could handle both, conjunction and disjunction. There are, however, significant differences to the approach considered in the present paper. First, the formal context defined in [1,17] is quite different from the one introduced above: its objects are pairs consisting of an interpretation $\mathcal{I}$ and an element of $\Delta^{\mathcal{I}}$. To be able to compute the counterexamples required by the attribute exploration algorithm, the subsumption algorithm had to be extended such that it computes appropriate finite countermodels [1]. Second, [1] was only interested in the Duquennes-Guigues base computed by the algorithm, and not in the concept lattice. In fact, in the experiments made with the approach introduced in [1], the base turned out to be usually rather small, whereas the lattice was very large (and thus it did not make sense to visualize it) [14].

In the future we will test the approach introduced in this paper with more examples from our chemical process engineering application. In particular, we will more closely analyze which of the least common subsumers yield concepts that make sense in the application domain, with the goal to develop appropriate heuristics for suggesting good candidates based on the computed concept lattice. In addition, we will try to optimize the method, with the goal to avoid even more

explicit lcs computations by using information from the partial implication base and the subcontext already computed.

References

1. F. Baader. Computing a Minimal Representation of the Subsumption Lattice of all Conjunctions of Concepts Defined in a Terminology. In *Proceedings of KRUSE'95*, 1995.
2. F. Baader and R. Küsters. Computing the least common subsumer and the most specific concept in the presence of cyclic $\mathcal{ALN}$-concept descriptions. In *Proceedings of KI'98*, Springer LNCS 1504, 1998.
3. F. Baader, R. Küsters, and R. Molitor. Computing least common subsumers in description logics with existential restrictions. In *Proceedings of IJCAI'99*, Morgan Kaufmann, 1999.
4. A. Borgida and P. Patel-Schneider. A semantics and complete algorithm for subsumption in the CLASSIC description logic. *J. Artificial Intelligence Research*, 1, 1994.
5. W.W. Cohen and H. Hirsh. Learning the CLASSIC description logic: Theoretical and experimental results. In *Proceedings of KR'94*, Morgan Kaufmann, 1994.
6. F.M. Donini, M. Lenzerini, D. Nardi, B. Hollunder, W. Nutt, and A.M. Spaccamela. The complexity of existential quantification in concept languages. *Artificial Intelligence*, 53, 1992.
7. W.F. Dowling and J.H. Gallier. Linear-time algorithms for testing the satisfiability of propositional Horn formulae. *J. Logic Programming*, 3, 1984.
8. V. Duquenne. Contextual implications between attributes and some representational properties for finite lattices. In *Beiträge zur Begriffsanalyse*, B.I. Wissenschaftsverlag, 1987.
9. M. Frazier and L. Pitt. CLASSIC learning. *Machine Learning*, 25, 1996.
10. B. Ganter. Finding all closed sets: A general approach. *Order*, 8, 1991.
11. B. Ganter and R. Wille. *Formal Concept Analysis – Mathematical Foundations*, Springer–Verlag, 1999.
12. I. Horrocks. The FACT system. In *Proceedings of Tableaux'98*, Springer LNCS 1397, 1998.
13. W. Marquardt. Trends in computer-aided process modeling. *Computers and Chemical Engineering*, 20(6/7), 1996.
14. M. Nassiri. Berechnung einer erweiterten Subsumtionshierarchie. Diploma thesis, RWTH Aachen, Germany, 1997.
15. U. Sattler. *Terminological Knowledge Representation Systems in a Process Engineering Application*. PhD thesis, RWTH Aachen, 1998.
16. G. Stumme. *Exploration Tools in Formal Concept Analysis*. In *Proceedings of OSDA'95*. Vol. 8 of *Studies in Classification, Data Analysis, and Knowledge Organization*. Springer-Verlag, 1996.
17. G. Stumme. The concept classification of a terminology extended by conjunction and disjunction. In *Proceedings of PRICAI'96*, Springer LNCS 1114, 1996.
18. G. Stumme and R. Wille. *Begriffliche Wissensverarbeitung — Methoden und Anwendungen*. Springer–Verlag, 2000.
19. F. Vogt and R. Wille. TOSCANA - A graphical tool for analyzing and exploring data. In *Graph Drawing*, Springer LNCS 894, 1995.

On the Contextual Logic of Ordinal Data

Silke Pollandt and Rudolf Wille

Fachbereich Mathematik , Technische Universität Darmstadt,
Schlossgartenstr. 7, D–64289 Darmstadt;
{pollandt,wille}@mathematik.tu-darmstadt.de

Abstract. This paper reports on first attempts to develop a *contextual logic of ordinal data*. The investigations are based on a mathematical theory of *ordinal contexts* which has been developed within Formal Concept Analysis. From ordinal contexts, *binary power context families* are derived as semantic basis of a contextual logic of ordinal data. They are used to characterize *compound attributes* extensionally. In this way, the contextual logic becomes a relational logic within the framework of the *Peircean Algebraic Logic*, as reconstructed by R. W. Burch. The considerations of this papers are discussed through an example of ordinal data investigated toward a meaningful representation in ordered vector spaces.

Contents

1. Ordinal Structures
2. Contextual Logic of Binary Power Context Families
3. Contextual Logic of Ordinal Contexts

1 Ordinal Structures

The aim of this paper is to show how methods of *Contextual Logic* (cf. [Wi97]), [Pr98], [GW99b]) may be introduced into *Ordinal Data Analysis* to support meaningful applications of mathematical investigations of ordinal structures. This shall be understood as a contribution to the development of a basic mathematical theory of ordinal data which is necessary for obtaining relevant interpretations of ordinal data.

According to [Co64], there are two main types of relations between objects in empirical data: dominance (order) and proximity (similarity). With respect to this classification, *ordinal data* may be understood as representations of *dominances*. Although ordinal data are basic and occur in high frequency, the development of a comprehensive structure theory for ordinal data only started in the early 1990's (see [SW92]). An obstacle for the development of a comprehensive structure theory was the absence of a basic notion of *ordinal structures*. For introducing a mathematical definition of ordinal structures, *Representational Measurement Theory* was taken as an adequate framework (cf. [KLST71], [Ro79]).

B. Ganter and G.W. Mineau (Eds.): ICCS 2000, LNAI 1867, pp. 306–316, 2000.
© Springer-Verlag Berlin Heidelberg 2000

In Representational Measurement Theory, empirical data are generally modelled by relational structures which consists of a set and a family of relations on this set; in this setting, dominances are usually mathematized by quasi-orders. Let us recall that a *quasi-order* $\lesssim$ on a set S is a reflexive transitive binary relation between the elements of S, i.e., $x \lesssim x$ for all $x \in S$, and $x \lesssim y$ and $y \lesssim z$ imply $x \lesssim z$ for all $x, y, z \in S$. In [SW92], ordinal structures have been specified by those relational structures in which all relations are quasi-orders. Formally, an *ordinal structure* is a relational structure $(S, (\lesssim_m)_{m\in M})$ for which the $\lesssim_m$ $(m \in M)$ are quasi-orders; in the case of finitely many quasi-orders, an ordinal structure is usually described by $(S, \lesssim_1, \ldots, \lesssim_n)$.

Ordinal structures are often given by object-attribute-tables as in the example of Figure 1: the table shows data from [SF68] describing the amount of absorption for four colour stimuli in the retina of a goldfish. For each colour

Receptor	Violet 430	Blue 458	Blue 485	Blue-Green 498
1	147	153	89	57
2	153	154	110	75
3	145	152	125	100
4	99	101	122	140
5	46	85	103	127
6	73	78	85	121
7	14	2	46	52
8	44	65	77	73
9	87	59	58	52
10	60	27	23	24
11	0	0	40	39

Fig. 1. Amounts of absorption of four colour stimuli by receptors in the retina of a goldfish

stimulus m, we obtain a quasi-order $\lesssim_m$ on the set S of all receptors as follows: $g \lesssim_m h$ if the colour stimulus m does not yield a greater value for the receptor g than for the receptor h. According to this understanding, the table in Figure 1 represents an ordinal structure $(S, \lesssim_{430}, \lesssim_{458}, \lesssim_{485}, \lesssim_{498})$. Notice that the quasi-orders are induced by an order on the attribute values. Thus, a mathematical model of object-attribute-tables for describing ordinal structures should include orderings on the sets of attribute values. An *ordinal context* is therefore defined as a structure $\mathbb{K} := (G, M, (W_m, \leq_m)_{m\in M}, I)$ where G and M are sets, $(W_m, \leq_m)_{m\in M}$ is a family of ordered sets, and I is a ternary relation with $I \subseteq \bigcup_{m\in M} G \times \{m\} \times W_m$ such that, for each $g \in G$ and $m \in M$, there is exactly one $w \in W_m$ with $(g, m, w) \in I$ (see [Wi82]). The elements of G, M,

and W_m ($m \in M$) are called *objects*, *attributes*, and *attribute values*, respectively; if $(g, m, w) \in I$ we say: the object g has the value w for the attribute m. An attribute m of $\mathbb{K}$ can be understood as a mapping from G into W_m; therefore we often write $m(g) = w$ instead of $(g, m, w) \in I$. In many cases, the attribute value sets of an ordinal context are the same set W and even the orderings might be equal; then we write for $\mathbb{K}$ also $(G, M, (W, \leq_m)_{m \in M}, I)$ or even $(G, M, (W, \leq), I)$. For an ordinal context $\mathbb{K} := (G, M, (W_m, \leq_m)_{m \in M}, I)$, the corresponding ordinal structure is defined by $\underline{S}(\mathbb{K}) := (G, (\lesssim_m)_{m \in M})$ where the quasi-orders $\lesssim_m$ are determined by $g \lesssim_m h :\Leftrightarrow m(g) \leq_m m(h)$. Conversely, every ordinal structure can be derived in this way from an ordinal context.

The purpose of a structure theory for ordinal data is to support meaningful interpretations of the data. Since interpretations are always based on concepts and their relationships, suitable conceptual structures are assigned to ordinal structures resp. ordinal contexts by methods of *Formal Concept Analysis* (for its basic notions and results, we refer to [GW99a]). For an ordinal context $\mathbb{K} := (G, M, (W_m, \leq_m)_{m \in M}, I)$, the most elementary method for such an assignment is the *plain ordinal scaling* defined by forming the derived formal context $\mathbb{K}_o := (G, \bigcup_{m \in M} \{m\} \times W_m, I_o)$ with $g I_o (m, w) :\Leftrightarrow m(g) \leq w$ and its concept lattice (cf. [GW89]). Figure 2 represents the formal context derived from the data table in Figure 1 by plain ordinal scaling. The concept lattice of this context is shown

	(430,0)	(430,14)	(430,44)	(430,46)	(430,60)	(430,73)	(430,87)	(430,99)	(430,145)	(430,147)	(430,153)	(458,0)	(458,2)	(458,27)	(458,59)	(458,65)	(458,78)	(458,85)	(458,101)	(458,152)	(458,153)	(458,154)	(485,23)	(485,40)	(485,46)	(485,58)	(485,77)	(485,85)	(485,89)	(485,103)	(485,110)	(485,122)	(485,125)	(498,24)	(498,39)	(498,52)	(498,57)	(498,73)	(498,75)	(498,100)	(498,121)	(498,127)	(498,140)
1										×	×											×	×							×	×	×	×	×			×	×	×	×	×	×	×
2											×											×									×	×	×					×	×	×	×	×
3									×	×	×									×	×	×											×						×	×	×	×
4								×	×	×	×								×	×	×	×											×	×								×
5				×	×	×	×	×	×	×	×							×	×	×	×	×								×	×	×	×								×	×
6						×	×	×	×	×	×						×	×	×	×	×	×						×	×	×	×	×	×							×	×	×
7		×	×	×	×	×	×	×	×	×	×		×	×	×	×	×	×	×	×	×	×			×	×	×	×	×	×	×	×	×			×	×	×	×	×	×	×
8			×	×	×	×	×	×	×	×	×				×	×	×	×	×	×	×	×				×	×	×	×	×	×	×	×				×	×	×	×	×	×
9							×	×	×	×	×				×	×	×	×	×	×	×	×				×	×	×	×	×	×	×	×			×	×	×	×	×	×	×
10					×	×	×	×	×	×	×			×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×
11	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×		×	×	×	×	×	×	×	×	×	×		×	×	×	×	×	×	×	×

Fig. 2. The formal context derived from Figure 1 by plain ordinal scaling

in Figure 3. The attribute concepts derived from the first colour stimulus (Violet 430) form a chain located on the left side of the diagram, while the chain of the attribute concepts derived from the fourth colour stimulus (Blue-Green 498) is on the right side; the chains of the other two stimuli (Blue 458 and Blue 485) are located in between in correspodence to the order of the wave lengths $430 < 458 < 485 < 498$. Is the recognition of this betweeness independent of the choice of the specific diagram representing the concept lattice? Since there is no adequate notion of betweeness of chains in lattices, the question arises how one can derive an ordinally invariant notion of betweeness in ordinal structures,

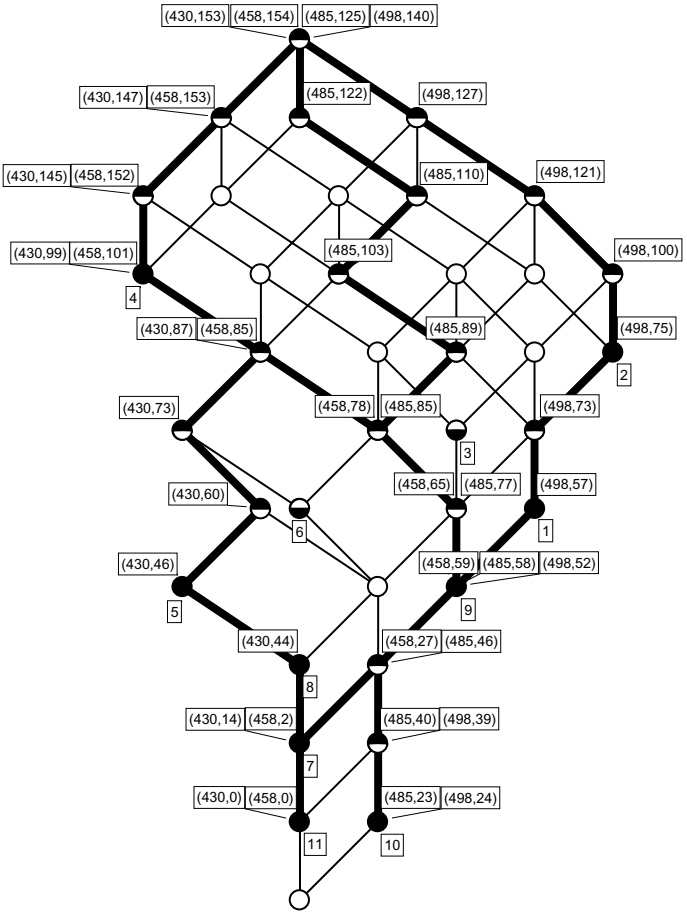


Fig. 3. The concept lattice of the formal context in Figure 2

which allows us, in particular, to justify, by the ordinal data in Figure 1, that the betweeness of the colour stimuli, according to their wave length, is perceived by the goldfish.

In Representational Measurement Theory an ordinally invariant notion of betweeness is derived for ordinal structures by *meaningful representations* of ordinal structures in ordered vector spaces. An elementary approach to such representations is described in [WW96] (see also [Wl95]). The basic step toward an ordered vector space representation consists of constructing an *addition* starting, without loss of generality, with an ordinal context $(G, \{m_0, m_1, \ldots, m_n\}, (W, \leq), I)$. Let us assume that we have already an addition $+$ on W such that $(W, +, \leq)$ is an ordered abelian group with $m_0(g) = m_1(g) + \cdots + m_n(g)$ for all $g \in G$. Obviously, these equalities have the following properties: if all summands on

the right side increase then the left side has to increase, and if all summands except one on the right side increase, but the left side decreases, than the remaining summand on the right side has to decrease too. These properties can be translated to the following symmetrized conditions for the derived ordinal structure $(G, \lesssim_0, \lesssim_1, \ldots, \lesssim_n)$ for which, as defined above, $g \lesssim_i h :\Leftrightarrow m_i(g) \leq m_i(h)$ $(i = 1, \ldots, n)$ but, conversely, $g \lesssim_0 h :\Leftrightarrow m_0(g) \geq m_0(h)$:

$$(A_i) \quad g \lesssim_j h \text{ for all } j \in \{0, 1, \ldots, n\} \setminus \{i\} \text{ implies } g \gtrsim_i h \quad (i = 0, \ldots, n).$$

In the case of the condition (A_i), we say that the quasi-order $\lesssim_i$ is *anti-ordinally dependent* of the quasi-orders $\lesssim_j$ $(j \in \{0, 1, \ldots, n\} \setminus \{i\})$.

It turns out, that the anti-ordinal dependency conditions allow us to construct an *n-quasi-group* operation. For obtaining even an ordered abelian group, we need further structural properties. For their formulation, two types of equivalence relations are defined for ordinal structures $(i, j = 0, 1, \ldots, n)$:

$$\Theta_i := \{(g, h) | g \lesssim_i h \text{ and } h \lesssim_i g\} \text{ and } \Psi_{ij} := \bigcap \{\Theta_k \mid k \in \{0, 1, \ldots, n\} \setminus \{i, j\}\}.$$

Now, we can define the so-called "*shifting conditions*" $(i = 1, \ldots, n$ with $n \geq 3)$:

$$(S_i) \quad g\Psi_{0i}k,\ k\Theta_i l,\ l\Psi_{0i}h,\ h\Theta_i g, \text{ and } k\Theta_0 l \text{ imply } g\Theta_0 h.$$

For an ordinal structure, the conditions of anti-ordinal dependencies and shifting guarantee a representation by an ordered vector space, which is meaningful, and yields an ordinally invariant notion of betweeness (see [WW95], [WW96]). For finding out how to check those kinds of conditions, it is useful to better understand the contextual logic of ordinal structures.

2 Contextual Logic of Binary Power Context Families

Before discussing the contextual logic of ordinal structures specifically, we outline the more general *Contextual Logic of Binary Relations* (cf. [EGSW00]). Semantically, this logic is based on *binary power context families* $\mathbb{K} := (\mathbb{K}_0, \mathbb{K}_1, \mathbb{K}_2)$ consisting of the formal contexts $\mathbb{K}_k := (G_k, M_k, I_k)$ $(k = 0, 1, 2)$ with $G_k \subseteq (G_0)^k$ for $k = 1, 2$ (cf. [Wi97]). The formal concepts of $\mathbb{K}_1$ and $\mathbb{K}_2$ have as extents unary and binary relations and are therefore called *unary* and *binary relational concepts*, respectively.

Now, we start to describe, in analogy to [GW99b], the *Contextual Attribute Logic* of binary power context families $\mathbb{K}$ by recursively defining *compound attributes* for $\mathbb{K}$ with the operational elements $\neg$, $\bigwedge$, $\bigvee$, $*$, and $\circ$:

– Each attribute $m \in M_k$ $(k = 0, 1, 2)$ is a compound attribute and also the "constants" $\bot_k$, $\top_k$, and id_2 with the extents

$$(\bot_k)^{I_k} := \emptyset,\ (\top_k)^{I_k} := G_k, \text{ and } (id_2)^{I_2} := \{(g, g) \mid g \in G_0\}.$$

- For each attribute $m \in M_k$ $(k = 0, 1, 2)$, we define its *negation* $\neg m$ to be a compound attribute with

$$(\neg m)^{I_k} := G_k \setminus m^{I_k}.$$

  Thus, $\boldsymbol{g}$ is in the extent of $\neg m$ if and only if $\boldsymbol{g}$ is not in the extent of m.
- For each set of attributes $A \subseteq M_k$ $(k = 0, 1, 2)$, we define the *conjunction* $\bigwedge A$ and the *disjunction* $\bigvee A$ to be compound attributes for which

$$(\bigwedge A)^{I_k} := \bigcap_{m \in A} m^{I_k} \text{ and } (\bigvee A)^{I_k} := \bigcup_{m \in A} m^{I_k}.$$

  Thus, $\boldsymbol{g}$ is in the extent of $\bigwedge A$ if and only if $\boldsymbol{g}$ is in the extent of all attributes $m \in A$, and $\boldsymbol{g}$ is in the extent of $\bigvee A$ if and only if $\boldsymbol{g}$ is in the extent of some attribute $m \in A$. Important compound attributes are the *sequents* $\bigvee(S \cup \{\neg m \mid m \in A\})$ determined by two subsets $A, S \subseteq M_k$, respectively, and denoted by (A, S), shortly .
- For each binary attribute $m \in M_2$, we define its *conversion* $*m$ to be a compound attribute with

$$(*m)^{I_2} := \{(g, h) \in G_2 | (h, g) \in m^{I_2}\}.$$

  Thus, (g, h) is in the extent of $*m$ if and only if (h, g) is in the extent of m.
- For each set $\mathcal{P} \subseteq \mathfrak{P}(M_0)$ and each two attributes $m, n \in M_2$, we define the *$\mathcal{P}$-concatenation* $m \circ_{\mathcal{P}} n$ to be the compound attribute with

$$(m \circ_{\mathcal{P}} n)^{I_2} := \{(g, h) \in G_2 | \exists_{B \in \mathcal{P}} \exists_{l \in (\neg(B, M_0 \setminus B))^{I_0}} (g, l) I_2 m \text{ and } (l, h) I_2 n\}.$$

  Thus, (g, h) is in the extent of $m \circ_{\mathcal{P}} n$ if and only if there exists a set $B \in \mathcal{P}$ and an element $l \in G_0$ such that (g, l) is in the extent of the attribute m, (l, h) is in the extent of n, and l is not in the extent of the sequent $(B, M_0 \setminus B)$. We write $\circ$ instead of $\circ_{\mathfrak{P}(M_0)}$.
- Iteration of the above compositions leads to further compound attributes, the extents of which are determined in the obvious manner.

The most important operations for binary relations are caputured by the introduced compound attributes; this should be clear, except for the concatenation. Since the notions of contextual attribute logic have to be expressed purely on the attribute level, the reference to objects in the definition of the concatenation has to be replaced by an attribute description. The following propositions show that this may still yield the usual concatenation for binary relations (cf. [GW99b]).

Proposition 1 *In every formal context $\mathbb{K} := (G, M, I)$, for $g \in G$ and $B \subseteq M$, $g \in (\neg(B, M \setminus B))^I$ is equivalent to $g^I = B$, i.e., the object g is, up to object-clarification, uniquely determined by the extent $(\neg(g^I, M \setminus g^I))^I$.*

Proof: Since $(B, M \setminus B) = \bigvee((M \setminus B) \cup \{\neg m \mid m \in B\})$, we obtain $(\neg(B, M \setminus B))^I = (\bigwedge(B \cap \{\neg m \mid m \in M \setminus B\}))^I = B^I \cap \bigcap\{G \setminus m^I \mid m \in M \setminus B\}$. If $g^I = B$ this yields $g \in g^{II} \cap (\neg(B, M \setminus B))^I = \bigcap\{G \setminus m^I \mid m \in M \setminus g^I\}$ because $g \in g^{II}$ and $g \notin m^I$ for each $m \in M \setminus g^I$.
Conversely, $g \in (\neg(B, M \setminus B))^I$ implies $g^I \subseteq B$, because of $g \notin m^I$ for each $m \in M \setminus B$, and $B \subseteq g^I$, because $g \in B^I$ and therefore $g^I \supseteq B^{II} \supseteq B$. □

Proposition 2 *Let* $\mathbb{K} := (\mathbb{K}_0, \mathbb{K}_1, \mathbb{K}_2)$ *be a binary power context family, let* $\mathcal{G}_0 := \{g^{I_0} \mid g \in G_0\}$*, and let* $m, n \in M_0$*. Then*

$$(m \circ_{\mathcal{G}_0} n)^{I_2} = \{(g,h) \in G^2 \mid \exists\, l \in G_0 : (g,l) \in m^{I_0} \text{ and } (l,n) \in n^{I_0}\}.$$

Proof: Let $(g,l) \in m^{I_0}$ and $(l,n) \in n^{I_0}$. Then, by Proposition 1, we have $l \in (\neg(g_0^I, M \backslash g^{I_0}))^{I_0}$; hence $(g,h) \in m \circ_{\mathcal{G}_0} n)^{I_2}$. Conversely, let $(g,h) \in m \circ_{\mathcal{G}_0} n)^{I_2}$. Then there exist a $B \in \mathcal{G}_0$ and $l \in (\neg(B, M \setminus B))^{I_0}$ with $(g,l) I_2 m$ and $(l,h) I_2 n$; hence (m,n) is in the set on the right side of the asserted equality. □

Logically, a basic question concerning binary power context families $\mathbb{K} := (\mathbb{K}_0, \mathbb{K}_1, \mathbb{K}_2)$ is whether the extents of compound attributes are equal or not. Two k-ary compound attributes m and n are called *extensionally equivalent* in $\mathbb{K}$ (in symbols: $m \approx_{\mathbb{K}_k} n$) if they have the same extent in $\mathbb{K}_k$; they are said to be *globally equivalent* (in symbols: $m \approx n$) if they have the same extent in each binary power context family with the same attribute sets M_0, M_1 and M_2. Since $m \approx_{K_k} n$ is equivalent to $(m \vee \neg n) \vee (\neg m \wedge n) \approx_{K_k} \top_k$, we also define the notion of *all-extensionality*: A compound attribute $m \in M_k$ $(k = 0, 1, 2)$ is called *all-extensional* in the binary power context family if $m^{I_k} = G_k$. It is of basic interest to find effective methods for deciding the all-extensionality for compound attributes of binary power context families.

The contextual logic of binary context families is seen in the framework of the *Peircean Algebraic Logic* which R. W. Burch created as "an attempt to amalgamate various systems of logic that Peirce developed over his long career" (see [Bu91]). The relational operations which result from the recursive definition of the compound attributes can all be derived from the basic operators of the Peircean Algebraic Logic (cf. [EGSW00]).

3 Contextual Logic of Ordinal Contexts

For specifying a contextual logic for an ordinal context

$$\mathbb{K} := (G, M, (W_m, \leq_m)_{m \in M}, I),$$

we transform the ordinal context in a binary power context family by the method of *relational scaling*, which was introduced by [PW99]. The simplest way of doing this is to define $\mathbb{K}_0$ as the context $\mathbb{K}_o$ derived from the ordinal context $\mathbb{K}$ by plain ordinal scaling (see Section 1) and $\mathbb{K}_2$ as the formal context $\mathbb{K}_q := (G^2, M, I_2)$ with $(g,h) I_2 m :\Leftrightarrow m(g) \leq_m m(h)$ ($\mathbb{K}_1$ may be assumed to be empty and therefore omitted). The contexual logic of the binary power context family $\mathbb{K}_{oq} := (\mathbb{K}_o, \mathbb{K}_q)$ may be understood as the semantic basis of the "canonical" contextual logic of the ordinal context $\mathbb{K}$. The other methods of ordinal conceptual scaling discussed in [SW92] and [GW99a] may produce richer contexts, but the compound attributes still yield the same extents; hence the logical expressibility keeps constant.

An interesting question is how restricted are the binary relations which are extents of compound attributes of the binary power context families $\mathbb{K}_{oq}$. It

might surprise that one has to expect any binary relation on the set G. This is the content of the following proposition:

Proposition 3 *For each binary relation R on G, there is an ordinal context $\mathbb{K}$ with the object set G such that R is the extent of a compound attribute of $\mathbb{K}_{oq}$.*

Proof: Let R be an arbitrary binary relation on G. We regard the ordinal context $\mathbb{K} := (G, M, (W_m, \leq_m), I)$ with

$$\begin{aligned} M &:= \{m_g | g \in G\} \cup \{m_{gh} | g, h \in G \text{ and } g \neq h\}, \\ W_g &:= \{0, 1\} \text{ with } 0 <_g 1, \\ W_{gh} &:= G \text{ with } \leq_{gh} := id_2 \cup \{(g, h)\}, \\ m_g(k) &:= \begin{cases} 0 & \text{if } k = g \\ 1 & \text{otherwise,} \end{cases} \\ m_{gh}(k) &:= k. \end{aligned}$$

(We write W_g instead of W_{m_g}, $\leq_g$ instead of $\leq_{m_g}$, and $\leq_{gh}$ instead of $\leq_{m_{gh}}$.) Then, in the corresponding formal context $\mathbb{K}_q := (G^2, M, I_2)$, we have

$$\begin{aligned} (m_g)^{I_2} &= ((G \setminus \{g\}) \times G) \cup \{(g, g)\}, \\ (\neg m_g)^{I_2} &= \{g\} \times (G \setminus \{g\}), \\ (*m_g)^{I_2} &= (G \times (G \setminus \{g\})) \cup \{(g, g)\}, \\ (\neg * m_g)^{I_2} &= ((G \setminus \{g\}) \times \{g\}. \end{aligned}$$

Therefore, we obtain $(\neg m_g \circ \neg * m_g)^{I_2} = \{(g, g)\}$ and hence

$$(\bigvee \{(\neg m_g \circ \neg * m_g) | (g, g) \in R\})^{I_2} = \{(g, g) | (g, g) \in R\};$$

furthermore, it follows $(m_{gh})^{I_2} = \{(k, k) | k \in G\} \cup \{(g, h)\}$ and from this $(\neg m_{gh})^{I_2} = \{(k, l) | (k, l) \neq (g, h) \text{ and } k \neq l\}$ and so

$$(\bigwedge \{\neg m_{gh} | (g, h) \notin R\})^{I_2} = \{(k, l) | (k, l) \in R \text{ and } k \neq l\}.$$

Finally, we get $(\bigvee \{(\neg m_g \circ \neg * m_g) | (g, g) \in R\} \vee \bigwedge \{\neg m_{gh} | (g, h) \notin R\})^{I_2} = R$. □

Now, we want to apply the presented language of the contextual logic of ordinal contexts to the spatial representation problem. The question is still open whether the ordinal structure given by the ordinal data in Figure 1 has a meaningful representation in an ordered vector space or not. For a binary power context family $\mathbb{K}_{oq} := (\mathbb{K}_o, \mathbb{K}_q)$, obtained by relational scaling of an ordinal context, the conditions (A_i) of anti-ordinal dependency $(i = 0, 1, \ldots, n)$ lead in general to the compound attributes

$$a_i := *q_i \vee \bigvee \{\neg q_k \mid k \in \{0, 1, \ldots, n\} \setminus \{i\}\},$$

$(q_{0,\ldots,n} \in M \cup *M)$ such that q_i is anti-ordinally dependent of the q_k with $k \in \{0, 1, \ldots, n\} \setminus \{i\}$ if and only if a_i is all-extensional in $\mathbb{K}_q$. For translating the

shifting conditions (S_i), it is more convenient to use the equivalent *permutability conditions* $(\Theta_i \cap \Theta_0) \circ \Psi_{0i} = \Psi_{0i} \circ (\Theta_i \cap \Theta_0)$ $(i = 1, \ldots, n)$. Let us define the following abreviations $(i, j = 0, 1, \ldots n)$:

$$e_i := q_i \wedge *q_i \quad \text{and} \quad e_{ij} := \bigwedge \{e_k \mid k \in \{0, 1, \ldots, n\} \setminus \{i, j\}\}.$$

For the binary power context family K_{oq} and $\mathcal{G}_o := \{g^{I_o} \mid g \in G_o\}$, the terms of the permutability conditions lead to the compound attributes

$$s_i := (e_i \wedge e_0) \circ_{\mathcal{G}_o} e_{0i} \quad \text{and} \quad t_i := e_{0i} \circ_{\mathcal{G}_o} (e_i \wedge e_0),$$

such that $(\{e_i\}^{I_o} \wedge \{e_0\}^{I_o}) \circ_{\mathcal{G}_o} \{e_{0i}\}^{I_o} = \{e_{0i}\}^{I_o} \circ_{\mathcal{G}_o} (\{e_i\}^{I_o} \wedge \{e_0\}^{I_o})$ if and only if s_i and t_i are extensionally equivalent in $\mathbb{K}_q$.

What help do the described compound attributes give to answer the representation question for our example? By relational scaling of the ordinal context represented in Figure 1, we obtain the binary power context family $\mathbb{K}_{oq}^{col} := (\mathbb{K}_o^{col}, \mathbb{K}_q^{col})$ with $G_o^{col} := \{1, \ldots, 11\}$, $M_o^{col} := \{(430, 0), (430, 14), \ldots, (498, 140)\}$, $G_q^{col} := (G_o^{col})^2$, and $M^{col} := \{q_{430}, q_{458}, q_{485}, q_{498}\}$. For our example, the conditions (A_i) of anti-ordinal dependency lead to the compound attributes

$$\neg q_j \vee \neg q_k \vee q_i$$
$$\text{with } (j, i, k) \in \{(430, 458, 485), (430, 458, 498), (430, 485, 498), (458, 485, 498)\},$$

such that $*q_i$ is anti-ordinally dependent of q_j and q_k if and only if $\neg q_j \vee \neg q_k \vee q_i$ is all-extensional in $\mathbb{K}_q^{col}$. Obviously, it is not difficult to check the all-extensionality algorithmically; hence one can easily confirm that, for any choice of three attributes in M_q, the described anti-ordinal dependency holds for those attributes.

Since the shifting conditions are only non-trivial for $n \geq 3$, they are of no help for anti-ordinal triples of attributes. For this case, the *Thomsen condition* (see [KLST71]) is meaningful, but there are not enough coincidences between the values of the attributes in our example, so that a direct application of the Thomsen condition does not work either. Nevertheless, after the positive answer concerning the anti-ordinal dependencies, it is not difficult to find a representation of our data in the euclidean plane; such a representation is shown in Figure 4 (cf. [WW96]).

Each colour stimulus is represented by a set of parallel lines: Violet 430 by the vertical lines, Blue-Green 498 by the horizontal lines, and the other two colour stimuli in between so that part of the colour circle becomes apparent. The visible betweeness is independent of the specific representation of the stimuli by sets of parallel lines in the euclidean plane as proved in [WW95]. Thus, the ordinal data confirm the hypothesis that the goldfisch perceives at least part of the colour circle according to the wave lengths of the chosen colour stimuli.

The example may indicate how the contextual logic of ordinal contexts can support the interpretation of ordinal data. Expressing structural properties by compound attributes allows us to activate general algorithms for determining extensional equivalence or all-extensionality to check those properties in ordinal

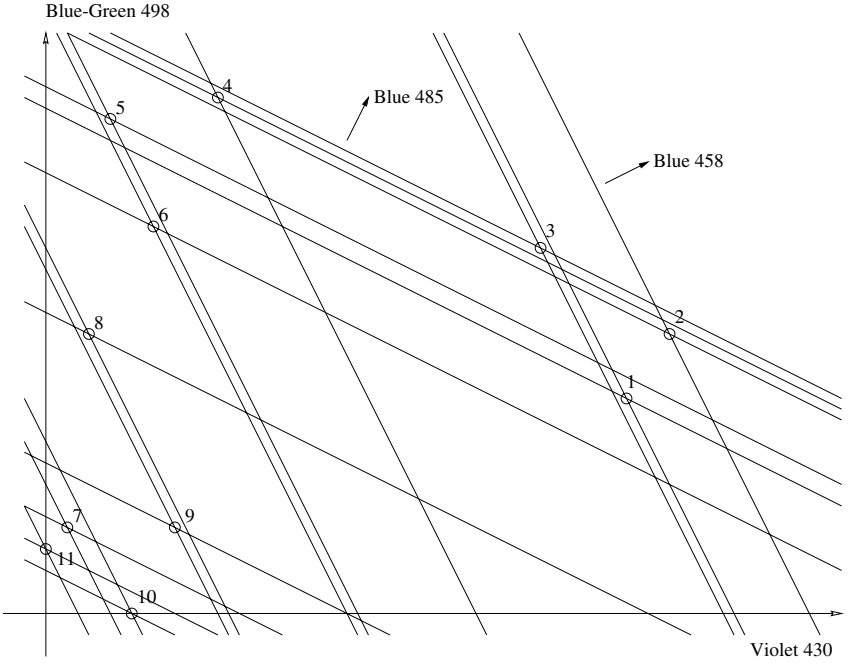


Fig. 4. A geometric representation of the ordinal data in Figure 1 in the euclidean plane

contexts. Of course, efficient algorithms have still to be developed in contextual logic of ordinal data, but at least it became clear in which directions further developments should go. Many research problems should be attacked as, for instance, the normal form and the word problem for compound attributes with respect to the extensional equivalence. In general, the contextual logic of ordinal data should be elaborated within the framework of Peircean Algebraic Logic.

References

Bu91. R. W. Burch: *A Peircean reduction thesis.* Texas Tech University Press, Lubbock 1991.

Co64. C. H. Coombs: *A theory of data.* Wiley, New York 1964.

EGSW00. P. Eklund, B. Groh, G. Stumme, R. Wille: A contextual-logic extension of TOSCANA. FB4-Preprint, TU Darmstadt 2000.

GW89. B. Ganter, R. Wille: Conceptual scaling. In: F. S. Roberts (ed.): *Applications of combinatorics and graph theory to the biological and social sciences.* Springer, New York, 1989, 139–167.

GW99a. B. Ganter, R. Wille: *Formal Concept Analysis: Mathematical Foundations.* Springer, Heidelberg 1999. (Translation of: *Formale Begriffsanalyse: Mathematische Grundlagen.* Springer, Heidelberg 1996.)

GW99b. B. Ganter, R. Wille: Contextual attribute logic. In: W. Tepfenhart, W. Cyre (eds.): *Conceptual structures: standards and practices*. LNAI **1640**. Springer, Heidelberg 1999, 377–388.

KLST71. D. H. Krantz, R. D. Luce, P. Suppes, A. Tversky: *Foundations of measurement*. AcademicPress, San Diego 1971 (vol.1), 1989 (vol.2), 1990 (vol.3).

Pr97. S. Prediger: Logical scaling in Formal Concept Analysis. In: D. Lukose, H. Delugach, M. Keeler, L. Searle, J. F. Sowa (eds.): *Conceptual structures: fulfilling Peirce's dream*. LNAI **1257**. Springer, Heidelberg 1997, 332–341.

Pr98. S. Prediger: *Kontextuelle Urteilslogik mit Begriffsgraphen. Ein Beitrag zur Restrukturierung der mathematischen Logik*. Dissertation, TU Darmstadt. Shaker Verlag, Aachen 1998.

PW99. S. Prediger, R. Wille: The lattice of concept graphs of a relationally scaled context. In: W. Tepfenhart, W. Cyre (eds.): *Conceptual structures: standards and practices*. LNAI **1640**. Springer, Heidelberg 1999, 401–414.

Ro79. F. S. Roberts: *Measurement theory with applications to decision making, utility, and the social sciences*. Addison-Wesley, Reading/MA 1979.

SF68. H. Schiffmann, Ph. Falkenberg: The organization of stimuli and sensory neurons. *Physiology and Behavior* **3** (1968), 197–201.

So84. J. F. Sowa: *Conceptual structures: Information processing in mind and machine*. Adison-Wesley, Reading 1984.

SW92. S. Strahringer, R. Wille: Towards a structure theory of ordinal data. In: M. Schader (ed.): *Analyzing and modeling data and knowledge*. Springer, Heidelberg 1992, 129–139.

Vo96. F. Vogt: *Datenstrukturen und Algorithmen zur Formalen Begriffsanalyse: Eine C++–Klassenbibliothek*. Springer, Heidelberg 1996.

VW95. F. Vogt, R. Wille: TOSCANA – A graphical tool for analyzing and exploring data. In: R. Tamassia, I. G. Tollis (eds.): *GraphDrawing '94*. LNCS **894**. Springer, Heidelberg 1995, 226–233.

Wi82. R. Wille: Restructuring lattice theory: an approach based on hierarchies of concepts. In: I. Rival (ed.): *Ordered sets*. Reidel, Dordrecht-Boston 1982, 445–470.

Wi97. R. Wille: Conceptual Graphs and Formal Concept Analysis. In: D. Lukose, H. Delugach, M. Keeler, L. Searle, J. F. Sowa (eds.): *Conceptual structures: fulfilling Peirce's dream*. LNAI **1257**. Springer, Heidelberg 1997, 290–303.

WW95. R. Wille, U. Wille: Uniqueness of coordinatizations of ordinal structures. In: *Contributions of general algebra*. Vol.9. Hölder-Pichler-Temsky, Wien 1995, 321–324.

WW96. R. Wille, U. Wille: Coordinatization of ordinal structures. *Order* **13**, 281–294.

Wl95. U. Wille: Geometric representation of ordinal contexts. Dissertation, Univ. Gießen 1995. Shaker Verlag, Aachen 1996.

Wl96. U. Wille: Representation of ordinal contexts by ordered n-quasigroups. *Europ.J.Combinatorics* **17** (1996), 317–333.

Boolean Concept Logic

Rudolf Wille

Technische Universität Darmstadt, Fachbereich Mathematik,
Schloßgartenstr. 7, D–64289 Darmstadt; wille@mathematik.tu-darmstadt.de

Abstract. The aim of this paper is to show how a *Boolean Concept Logic* may be elaborated as a mathematical theory based on *Formal Concept Analysis* [GW96]. For this purpose, concept lattices are extended by further operations, mainly *negation* and *opposition*. Two extensions are discussed which lead, on the one hand, to *algebras of protoconcepts* equationally equivalent to *double Boolean algebras* and, on the other hand, to *concept algebras* quasi-equationally equivalent to *dicomplemented lattices*. In both cases, basic representation theorems are proved. These results are not only basic for *Contextual Concept Logic* but also for *Contextual Judgment Logic* with its theory of concept graphs.

Contents

1. Contextual Concept Logic
2. Algebras of Protoconcepts
3. Double Boolean Algebras
4. Concept Algebras
5. Further Research

1 Contextual Concept Logic

George Boole presented in his book *"An Investigation of the Laws of Thought"* [Bo54] a mathematical theory of logic based on the notions of *signs* and *classes*, where signs represent classes of objects. An important aim of his investigation is "the derivation of the laws of the symbols of logic from the laws of the operations of the human mind" [Bo54; p.39]. Boole argues that the symbolic laws of logic, "determined *á posteriori* from the constitution of language, [...] are in reality the laws of that mental operations" which he describes in his book [Bo54; p.44]. Mathematically, Boole developed his logic as a theory of symbolic operations applied to classes of objects. This approach - today called the *Boolean Class Logic* - became basic for modern mathematical logic.

In his book [Bo54; p.42], Boole pointed out that, in every discourse, "there is an assumed or expressed limit within which the subjects of its operations are confined"; he called the field within which all the objects of such discourse are found: the *universe of discourse*. In *Contextual Attribute Logic* [GW99], in which signs are understood as attributes, Boole's idea of a limited universe of

B. Ganter and G.W. Mineau (Eds.): ICCS 2000, LNAI 1867, pp. 317–331, 2000.
© Springer-Verlag Berlin Heidelberg 2000

discourse finds its mathematical representation in the notion of a "formal context", the basic notion of Formal Concept Analysis (see [GW96]). A *formal context* is defined as a tripel $\mathbb{K} := (G, M, I)$ combining two sets G and M with a binary relation I between them, i.e. $I \subseteq G \times M$; the elements of G and M are called *objects* and *attributes*, respectively. For $Y \subseteq M$, the *derivation* is defined by $Y' := \{g \in G \mid gIm \text{ for all } m \in Y\}$ (and dually, for $X \subseteq G$, by $X' := \{m \in M \mid gIm \text{ for all } g \in X\}$); in particular, for $m \in M$, we get $m' := \{m\}' = \{g \in G \mid gIm\}$, the so-called *attribute extent* of m which may be understood as the Boolean class belonging to the "sign" m. In [GW99], it is outlined how this correspondence may lead to a development of a *Contextual Attribute Logic* in the spirit of Boole's approach to mathematical logic.

The aim of this paper is to show how a *Boolean Concept Logic* may be elaborated as part of a *Contextual Concept Logic*, which is based on the mathematical theory of formal contexts and concepts presented in [GW96]. In general, the Contextual Concept Logic shall be developed together with a Contextual Judgment Logic as a mathematization of the traditional philosophical logic with the main aim to support conceptual knowledge representation and processing (see [Wi96],[Wi97a],[Wi97b],[Pr98],[GW99]).

A *formal concept* in Contextual Concept Logic is defined with respect to a formal context $\mathbb{K} := (G, M, I)$, namely as a pair (A, B) with $A \subseteq G$ and $B \subseteq M$ such that $A' = B$ and $B' = A$; A and B are called the *extent* and the *intent* of the formal concept (A, B), respectively. The formal concepts of $\mathbb{K}$ can be derived by forming (X'', X') for $X \subseteq G$ and (Y', Y'') for $Y \subseteq M$, respectively, because of $X''' = X'$ and $Y' = Y'''$. The set $\mathfrak{B}(\mathbb{K})$ of all formal concepts of the formal context $\mathbb{K}$ carries the natural *subconcept-superconcept-*order given by $(A_1, B_1) \leq (A_2, B_2) :\Leftrightarrow A_1 \subseteq A_2$ ($\Leftrightarrow B_1 \supseteq B_2$). $\mathfrak{B}(\mathbb{K})$ with the defined subconcept-superconcept-order is always a complete lattice, called the *concept lattice* of the formal context $\mathbb{K}$ and denoted by $\underline{\mathfrak{B}}(\mathbb{K})$. Since $\mathbb{K}$ can be reconstructed from the concept lattice by determining the *object concepts* $\gamma g := (\{g\}'', \{g\}')$ ($g \in G$) and the *attribute concepts* $\mu m := (\{m\}', \{m\}'')$ ($m \in M$), which satisfy $\gamma g \leq \mu m \Leftrightarrow gIm$, the Contextual Attribute Logic may be also expressed in terms of concept lattices of formal contexts and so be understood as part of Contextual Concept Logic.

For extending the Boolean Attribute Logic to a Boolean Concept Logic, the main obstacle is the absence of formal negations on the conceptual level. Therefore we first concentrate on the question how to introduce formal negations in conceptual structures based on formal contexts. In the history of logic one can distinguish essentially four forms of negations (cf. [Me84]): (1) negative act of judgment, (2) negation of propositional logic, (3) negative copula, (4) negative concept or property. The first three forms of negations have to be discussed with respect to Contextual Judgment Logic which is not in the scope of this paper. Thus, we will focus on the form of negative concepts and properties. Boole understood a negated sign as the represention of the complement of the class represented by the original sign in the given universe of discourse; he writes: "if from the conception of the universe, as consisting of "men" and "not-men",

we exclude the conception of "men", the resulting conception is that of the contrary class, "not-men"." [Bo94; p.48] Mathematically, Boole takes, within the class of the considered universe, the complement of the class represented by the origininal sign as the class of the negated sign. If this is analogously done with a formal concept (A, B) in a formal context (G, M, I), the problem arises that the complement $G \setminus A$ of the extent A need not to be an extent again (and that the complement $M \setminus B$ of the intent B need not be an intent again). For obtaining the negation as an operation on the set of formal concepts of the formal context, we may take as the negation of (A, B) the formal concept generated by $G \setminus A$, namely $((G \setminus A)'', (G \setminus A)')$. But then we have the disadvantage that the extents of the formal concept and its negation might not be disjoint; nevertheless, we will investigate further this "weak negation" by studying so-called "concept algebras" in Section 4.

The only possible way for keeping the correspondence between negation and set-complement is to generalize the notion of formal concepts. This has already been done more than a decade ago by introducing the notion of a *semiconcept* (see [LW91],[Wi92]) which algebraically led to the so-called *double Boolean algebras* (see [HLSW99]). The approach in this paper is based on the slightly more general notion of a protoconcept: A *protoconcept* of a formal context $\mathbb{K} := (G, M, I)$ is defined as a pair (A, B) with $A \subseteq G$ and $B \subseteq M$ such that $A' = B''$, which is equivalent to $B' = A''$. The protoconcepts of a formal context $\mathbb{K}$ may be understood as those formal concepts (A, B) of subcontexts of $\mathbb{K}$ for which the same formal concept of $\mathbb{K}$ is derived from A and from B by applying the derivations of $\mathbb{K}$, i.e. $(A'', A') = (B', B'')$. Thus, a formal concept (A, B) of a subcontext is a protoconcept of the whole context if and only if it extends uniquely to a formal concept (C, D) of the whole context for which $C \times D$ is smallest extent-intent-product containing $A \times B$, i.e. the protoconcept (A, B) represents the essential information about the formal concept (C, D). The notion of a protoconcept is therefore helpful for understanding which conceptual information carries over from a formal context to some of its contextual extensions. An extreme case is the extension of a formal context (G, M, I) to the context $(G \dot\cup \{h\}, M \dot\cup \{n\}, I \dot\cup (G \times \{n\}) \dot\cup (\{h\} \times M) \dot\cup \{(h, n)\})$ because each formal concept of (G, M, I) is a "proper" protoconcept of the extended context and each formal concept of the extended context, which is neither the smallest nor the greatest, is the unique extension of a formal concept of (G, M, I).

Basic definitions and results concerning *algebras of protoconcepts* are presented in the next section. The considered operations on the set of all protoconcepts of a formal context are *meet, join, negation, opposition, nothing*, and *all*. These operations give rise to algebraic structures: the so-called *double Boolean algebras*. A main result of this paper, proved in Section 3, states that each double Boolean algebra is embeddable into some algebra of protoconcepts. This representation theorem has as consequence that the equational axioms of double Boolean algebras determine the equational theory of the algebras of protoconcepts (in analogy to the wellknown result that the equational axioms of Boolean algebras determine the equational theory of powerset algebras).

2 Algebras of Protoconcepts

The set $\mathfrak{P}(\mathbb{K})$ of all protoconcepts of a formal context $\mathbb{K} := (G, M, I)$ is structured by the *generalization order* $\sqsubseteq$ which is defined by

$$(A_1, B_1) \sqsubseteq (A_2, B_2) :\Longleftrightarrow A_1 \subseteq A_2 \text{ and } B_1 \supseteq B_2.$$

In general, this order does not yield a lattice structure. However, there are operations of logical nature on $\mathfrak{P}(\mathbb{K})$ which can be defined as follows:

$$\begin{aligned}(A_1, B_1) \sqcap (A_2, B_2) &:= (A_1 \cap A_2, (A_1 \cap A_2)') \\ (A_1, B_1) \sqcup (A_2, B_2) &:= ((B_1 \cap B_2)', B_1 \cap B_2) \\ \neg(A, B) &:= (G \setminus A, (G \setminus A)') \\ \multimap(A, B) &:= ((M \setminus B)', M \setminus B) \\ \bot &:= (\emptyset, M) \\ \top &:= (G, \emptyset)\end{aligned}$$

The set $\mathfrak{P}(\mathbb{K})$ together with the operations $\sqcap, \sqcup, \neg, \multimap, \bot$, and $\top$ is called the *algebra of protoconcepts* of $\mathbb{K}$ and is denoted by $\underline{\mathfrak{P}}(\mathbb{K})$; the operations are named *"meet", "join", "negation", "opposition", "nothing"*, and *"all"*. The *meet* is determined by the intersection of object sets (extensional conjunction) and the *join* by the intersection of attribute sets (intensional conjunction); the *negation* is determined by the complement of an object set (Boole's contrary class) and the *opposition* by the complement of an attribute set (Boethius' opposita contraria); the constants *nothing* and *all* represent the extreme states of an object set (contradictory and universal state).

The logical meaning of five of the defined operations should be clear, but the *opposition* needs further explanations: Already Aristotle discussed the oposition in his "Organon", generally, and distinguished between four kinds of opposition for which Boethius introduced the following Latin terms: "opposita relativa", "opposita contraria", "habitus et privatio", "opposita contradictoria" (cf. [BM74]). The negation is defined according to the meaning of "opposita contradictoria", while the opposition may be understood as an operational formalization of the meaning of "opposita contraria" as discussed in philosophical logic until today. In [KL67], for instance, predicates P and Q are called "contrary" if $P(x)$ always implies $\neg Q(x)$; this condition analogously holds for protoconcepts (A, B) and $\multimap(A, B)$ of formal contexts (G, M, I) with $M' = \emptyset$. The special case of "polar-contrary" opposition (also discussed in [KL67]) may occur especially in formal contexts (G, M, I) for which M consists of dichotomic attribute pairs m_j and n_j $(j \in J)$, namely as contrary pairs of protoconcepts with the attribute sets $\{m_j \mid j \in J_0\} \cup \{n_j \mid j \in J \setminus J_0\}$ and $\{n_j \mid j \in J_0\} \cup \{m_j \mid j \in J \setminus J_0\}$.

The Pythagoreans considered the oppositions as the origin of every being. The pre-Socratic philosopher Empedocles incorporated Pythagorean ideas in his philosophy and systematized the Pythagorean conception of the oppositions by tracing them back to the two oppositions *moist*$\leftrightarrow$*dry* and *cold*$\leftrightarrow$*warm*. By these oppositions, Empedocles explained especially the doctrine of the four elements, as described by the formal context in Figure 1; Figure 2 shows the ordered set of all protoconcepts of the formal context in Figure 1.

	cold	moist	dry	warm
water	×	×		
earth	×		×	
air		×		×
fire			×	×

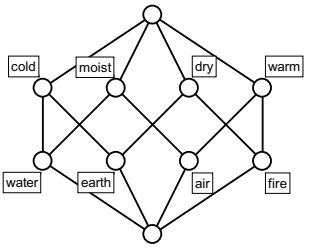


Fig. 1. A context and its concept lattice representing Empedocles' conception of the four elements

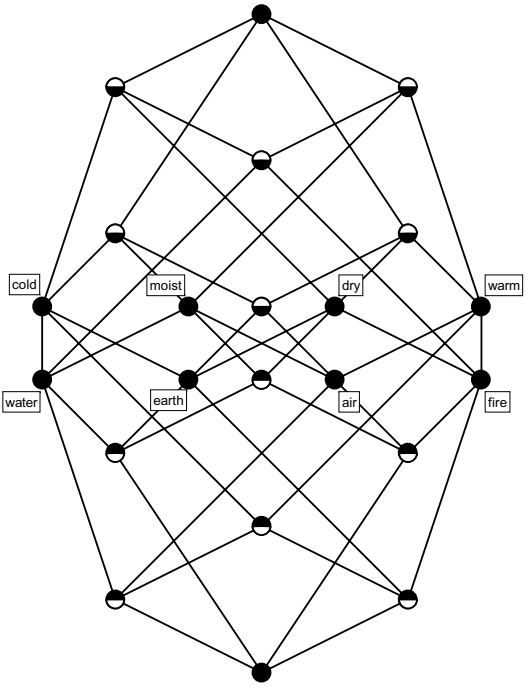


Fig. 2. The algebra of protoconcepts of the formal context in Figure 1

Clearly, the opposition of the concept "water" yields the concept "fire" (and vice versa), while the opposition of the concept "earth" yields the concept "air" (and vice versa). To read this from the diagram in Figure 2, one has to understand how to determine the object and attribute set of a protoconcept represented by a particular circle in the diagram: those objects (resp. attributes) belong to the protoconcept whose names are written on circles on a path of line segments descending (resp. ascending) from the circle representing the protoconcept (notice that, for some protoconcepts, the object resp. attribute set is empty).

The protoconcepts of the formal context in Figure 1 have the special property that one of their sets is the derivation of the other. This property also holds for all protoconcepts which result from an operation in the algebras of protoconcepts as the above definitions show. It is useful to conceptualize this property: A *semiconcept* of a formal context $\mathbb{K} := (G, M, I)$ is defined as a pair (A, B) with $A \subseteq G$ and $B \subseteq M$ such that $A' = B$ or $B' = A$. Obviously, each semiconcept is a protoconcept. For a better understanding of the structure of the protoconcept algebra $\underline{\mathfrak{P}}(\mathbb{K})$, we consider the two types of semiconcepts collected in the following two sets:

$$\mathfrak{P}(\mathbb{K})_{\sqcap} := \{(A, A') \mid A \subseteq G\} \text{ and } \mathfrak{P}(\mathbb{K})_{\sqcup} := \{(B', B) \mid B \subseteq M\}.$$

Then $\mathfrak{H}(\mathbb{K}) := \mathfrak{P}(\mathbb{K})_{\sqcap} \cup \mathfrak{P}(\mathbb{K})_{\sqcup}$ is the set of all semiconcepts of $\mathbb{K}$ which, together with the restrictions of the operations $\sqcap, \sqcup, \neg, \multimap, \bot$, and $\top$ is a subalgebra of $\underline{\mathfrak{P}}(\mathbb{K})$, denoted by $\underline{\mathfrak{H}}(\mathbb{K})$. Furthermore, $\mathfrak{B}(\mathbb{K}) = \mathfrak{P}(\mathbb{K})_{\sqcap} \cap \mathfrak{P}(\mathbb{K})_{\sqcup}$, and $(\mathfrak{B}(\mathbb{K}), \sqsubseteq)$ is the concept lattice of $\mathbb{K}$ in which the operations $\sqcap$ and $\sqcup$ yield the infimum and the supremum of two formal concepts, respectively. For the structural analysis of $\underline{\mathfrak{P}}(\mathbb{K})$, it is useful to define additional operations on $\mathfrak{P}(\mathbb{K})$:

$$\mathfrak{x} \sqcup\!\!\!\!\cdot\; \mathfrak{y} := \neg(\neg\mathfrak{x} \sqcap \neg\mathfrak{y}) \text{ and } \mathfrak{x} \barwedge \mathfrak{y} := \multimap(\multimap\mathfrak{x} \sqcup \multimap\mathfrak{y}),$$
$$\overline{\top} := \neg\bot \text{ and } \underline{\bot} := \multimap\top.$$

Obviously, $\mathfrak{P}(\mathbb{K})_{\sqcap}$ together with the restrictions of the operations $\sqcap, \sqcup\!\!\!\!\cdot\;, \neg, \bot, \overline{\top}$ is a Boolean algebra $\underline{\mathfrak{P}}(\mathbb{K})_{\sqcap}$, isomorphic to the Boolean algebra of all subsets of G; dually, $\mathfrak{P}(\mathbb{K})_{\sqcup}$ together with the restrictions of the operations $\barwedge, \sqcup, \multimap, \underline{\bot}, \top$ is a Boolean algebra $\underline{\mathfrak{P}}(\mathbb{K})_{\sqcup}$, antiisomorphic to the Boolean algebra of all subsets of M. The order relation on $\underline{\mathfrak{P}}(\mathbb{K})$ agrees with the derivable order relations of the two Boolean algebras $\underline{\mathfrak{P}}(\mathbb{K})_{\sqcap}$ and $\underline{\mathfrak{P}}(\mathbb{K})_{\sqcup}$.

Theorem 1 *The following equations are valid in* $\underline{\mathfrak{P}}(\mathbb{K})$*:*

1a)	$(x \sqcap x) \sqcap y = x \sqcap y$	*1b)*	$(x \sqcup x) \sqcup y = x \sqcup y$
2a)	$x \sqcap y = y \sqcap x$	*2b)*	$x \sqcup y = y \sqcup x$
3a)	$x \sqcap (y \sqcap z) = (x \sqcap y) \sqcap z$	*3b)*	$x \sqcup (y \sqcup z) = (x \sqcup y) \sqcup z$
4a)	$x \sqcap (x \sqcup y) = x \sqcap x$	*4b)*	$x \sqcup (x \sqcap y) = x \sqcup x$
5a)	$x \sqcap (x \sqcup\!\!\!\!\cdot\; y) = x \sqcap x$	*5b)*	$x \sqcup (x \barwedge y) = x \sqcup x$
6a)	$x \sqcap (y \sqcup\!\!\!\!\cdot\; z) = (x \sqcap y) \sqcup\!\!\!\!\cdot\; (x \sqcap z)$	*6b)*	$x \sqcup (y \barwedge z) = (x \sqcup y) \barwedge (x \sqcup z)$
7a)	$\neg\neg(x \sqcap y) = x \sqcap y$	*7b)*	$\multimap\multimap(x \sqcup y) = x \sqcup y$
8a)	$\neg(x \sqcap x) = \neg x$	*8b)*	$\multimap(x \sqcup x) = \multimap x$
9a)	$x \sqcap \neg x = \bot$	*9b)*	$x \sqcup \multimap x = \top$
10a)	$\neg\bot = \top \sqcap \top$	*10b)*	$\multimap\top = \bot \sqcup \bot$
11a)	$\neg\top = \bot$	*11b)*	$\multimap\bot = \top$
12)	$(x \sqcap x) \sqcup (x \sqcap x) = (x \sqcup x) \sqcap (x \sqcup x)$.		

Proof: The equations of the theorem can be easily verified in algebras of protoconcepts. We only demonstrate this by proving 12): Let $x := (A, B) \in \mathfrak{P}(\mathbb{K})$. Then $x \sqcap x = (A, A')$ and $x \sqcup x = (B', B)$. It follows that $(x \sqcap x) \sqcup (x \sqcap x) = (A'', A')$

and $(x \sqcup x) \sqcap (x \sqcup x) = (B', B'')$. By the definition of a protoconcept, we have $A' = B''$ and $B' = A''$. Hence $(A'', A') = (B', B'')$, i. e., the equation 12) holds in $\underline{\mathfrak{P}}(\mathbb{K})$. □

The question arises whether the equations of Theorem 1 are enough for determining the equational theory of algebras of protoconcepts, i.e., whether each equation valid in all algebras of protoconcepts can be entailed by the equations of Theorem 1. This is proved in the next section by investigating so-called "double Boolean algebras".

3 Double Boolean Algebras

For investigating the equational theory of algebras of protoconcepts, we introduce the notion of a *double Boolean algebra* which is an algebraic structure $\underline{D} := (D, \sqcap, \sqcup, \neg, \lrcorner, \bot, \top)$ of type $(2, 2, 1, 1, 0, 0)$ satisfying the equations 1a) to 11a), 1b) to 11b), and 12) of Theorem 1, where the further operations are defined as in Section 2 by

$$x \sqcup\!\!\sqcup y := \neg(\neg x \sqcap \neg y) \text{ and } x \sqcap\!\!\sqcap y := \lrcorner(\lrcorner x \sqcup \lrcorner y),$$
$$\overline{\top} := \neg\bot \text{ and } \underline{\bot} := \lrcorner\top.$$

In particular, each algebra of protoconcepts is a double Boolean algebra. Algebras of semiconcepts additionally satisfy the following condition:

$$13)\ x \sqcap x = x \text{ or } x \sqcup x = x.$$

A double Boolean algebra $\underline{D}$ satisfying the condition 13) is called *pure*, because it only consists of the two subsets $D_\sqcap := \{x \in D \mid x \sqcap x = x\}$ and $D_\sqcup := \{x \in D \mid x \sqcup x = x\}$ which both carry a Boolean structure. A detailed investigation of the structure of algebras of semiconcepts and double Boolean algebras is presented in [HLSW99]. Here we concentrate on questions concerning the equational theory of algebras of protoconcepts.

For introducing an order on a double Boolean algebra, we imitate the order definition in Section 2:

$$x \sqsubseteq y :\Leftrightarrow x \sqcap y = x \sqcap x \text{ and } x \sqcup y = y \sqcup y$$

Lemma 1 *In a double Boolean algebra the following conditions hold:*
(1) $x \sqcap y \sqsubseteq x \sqsubseteq x \sqcup y$,
(2) the mapping $x \to x \sqcap y$ *preserves* $\sqsubseteq$ *and* $\sqcap$,
(3) the mapping $x \to x \sqcup y$ *preserves* $\sqsubseteq$ *and* $\sqcup$.

Proof: (1): 2a), 3a), and 1a) yield $(x \sqcap y) \sqcap x = y \sqcap (x \sqcap x) = y \sqcap (y \sqcap (x \sqcap x)) = (x \sqcap y) \sqcap (x \sqcap y)$. By 2b) and 4b) it follows $(x \sqcap y) \sqcup x = x \sqcup x$. Hence $x \sqcap y \sqsubseteq x$. Dually, we obtain $x \sqsubseteq x \sqcup y$. (2) and (3) follows straight forward. Let us only mention that $x \sqsubseteq y$ implies $(x \sqcap z) \sqcup (y \sqcap z) = (x \sqcap x \sqcap z) \sqcup (y \sqcap z) = (x \sqcap y \sqcap z) \sqcup (y \sqcap z) = (y \sqcap z) \sqcup (y \sqcap z)$ by 4b). □

For a double Boolean algebra $\underline{D} := (D, \sqcap, \sqcup, \neg, \dashv, \bot, \top)$ let $x_\sqcap := x \sqcap x$ and $x_\sqcup := x \sqcup x$. We obtain that $D_\sqcap = \{x_\sqcap \mid x \in D\}$ and $D_\sqcup = \{x_\sqcup \mid x \in D\}$, and that $\underline{D}_\sqcap := (D_\sqcap, \sqcap, \sqcup, \neg, \bot, \overline{\top})$ and $\underline{D}_\sqcup := (D_\sqcup, \overline{\sqcap}, \sqcup, \dashv, \underline{\bot}, \top)$ are Boolean algebras.

For proving that each double Boolean algebra can be quasi-embedded into the algebra of protoconcepts of a suitable context, we define the notion of a filter and an ideal of a double Boolean algebra $\underline{D}$: A subset F of $\underline{D}$ is called a *filter* if $x \in F$ and $x \sqsubseteq y$ in $\underline{D}$ imply $y \in F$ and if $x \in F$ and $y \in F$ imply $x \sqcap y \in F$; an *ideal* of $\underline{D}$ is dually defined. A subset F_0 is called a *base* of the filter F if $F = \{y \in D \mid x \sqsubseteq y \text{ for some } x \in F_0\}$; again, a base of an ideal is dually defined.

Lemma 2 *Let F be a filter of a double Boolean algebra $\underline{D}$.*
(1) $F \cap D_\sqcap$ and $F \cap D_\sqcup$ are filters of the Boolean algebras $\underline{D}_\sqcap$ and $\underline{D}_\sqcup$, respectively.
(2) Each filter of the Boolean algebra $\underline{D}_\sqcap$ is a base of some filter of $\underline{D}$; in particular, $F \cap D_\sqcap$ is a base of F.

Proof: Since the restrictions of $\sqcap$ and $\sqsubseteq$ to $D_\sqcap$ are the meet operation and the order relation of the Boolean algebra $\underline{D}_\sqcap$, $F \cap D_\sqcap$ is obviously a filter of $\underline{D}_\sqcap$. $F \cap D_\sqcup$ is a filter of the Boolean algebra $\underline{D}_\sqcup$ because $x \sqcap y \sqsubseteq x \overline{\sqcap}\ y$ for arbitrary $x, y \in D_\sqcup$, namely $x \sqcap y$ is a lower bound of x and y by Lemma 1(1) and $x \overline{\sqcap}\ y$ is the greatest lower bound of x and y since the restriction of $\sqsubseteq$ to $D_\sqcup$ is the order of the Boolean algebra $\underline{D}_\sqcup$. Now, let E be a filter of $\underline{D}_\sqcap$. For $x_1 \sqsubseteq y_1$ and $x_2 \sqsubseteq y_2$ with $x_1, x_2 \in E$, we obtain $x_1 \sqcap x_2 \sqsubseteq x_1 \sqcap y_2 \sqsubseteq y_1 \sqcap y_2$ by Lemma 1(2); hence $\{y \in D \mid x \sqsubseteq y \text{ for some } x \in E\}$ is a filter in $\underline{D}$ with E as a base. For $y \in F$ we have that $y \sqcap y \in F \cap D_\sqcap$ and $y \sqcap y \sqsubseteq y$ by Lemma 1(1). Thus, $F \cap D_\sqcap$ is a base of F. □

Let $\mathfrak{F}_p(\underline{D})$ be the set of all filters F of the double Boolean algebra $\underline{D}$ for which $F \cap D_\sqcap$ is a prime filter of the Boolean algebra $\underline{D}_\sqcap$, and let $\mathfrak{I}_p(\underline{D})$ be the set of all ideals I of $\underline{D}$ for which $I \cap D_\sqcap$ is a prime ideal of the Boolean algebra $\underline{D}_\sqcup$ (for the definitions and results concerning prime filters and prime ideals see [DP92], p.185ff.). Now, we define the *standard context* for a double Boolean algebra $\underline{D}$ by

$$\mathbb{K}(\underline{D}) := (\mathfrak{F}_p(\underline{D}), \mathfrak{I}_p(\underline{D}), \triangle)$$

where $F \triangle I :\Leftrightarrow F \cap I \neq \emptyset$ for $F \in \mathfrak{F}_p(\underline{D})$ and $I \in \mathfrak{I}_p(\underline{D})$. For $x \in D$, let $\mathfrak{F}_x := \{F \in \mathfrak{F}_p(\underline{D}) \mid x \in F\}$ and let $\mathfrak{I}_x := \{I \in \mathfrak{I}_p(\underline{D}) \mid x \in I\}$.

Lemma 3 *The derivations in $\mathbb{K}(\underline{D})$ yield:*
(1) $\mathfrak{F}'_x = \mathfrak{I}_x = \mathfrak{I}_{x_\sqcup}$ for all $x \in D_\sqcap$,
(2) $\mathfrak{I}'_y = \mathfrak{F}_y = \mathfrak{F}_{y_\sqcap}$ for all $y \in D_\sqcup$,
(3) $\mathfrak{F}'_z = \mathfrak{I}_{z_\sqcap} = \mathfrak{I}_{z_{\sqcap\sqcup}}$ and $\mathfrak{I}'_z = \mathfrak{F}_{z_\sqcup} = \mathfrak{F}_{z_{\sqcup\sqcap}}$ for all $z \in D$.

Proof: (1): Let $x \in D_\sqcap$ and let $I \in \mathfrak{I}_x$. Then $x \in F \cap I$ for all $F \in \mathfrak{F}_x$; hence $I \in \mathfrak{F}'_x$. Conversely, let $I \in \mathfrak{F}'_x$. Suppose $x \notin I$. Then $I \cap D_\sqcap$ is an ideal of $\underline{D}_\sqcap$, by the dual of Lemma 2(1), and $x \in D_\sqcap \setminus I$. The ideal $I \cap D_\sqcap$ is contained in

a prime ideal of $\underline{D}_\sqcap$ the complement of which in $D_\sqcap$ is a prime filter E of $\underline{D}_\sqcap$ containing x. By Lemma 2(2), $F := \{y \in D \mid w \sqsubseteq y \text{ for some } w \in E\}$ is a filter of $\underline{D}$ with $F \cap D_\sqcap = E$; hence $F \in \mathfrak{F}_x$. But $F \cap I = \emptyset$ which contradicts $I \in \mathfrak{F}'_x$. Thus, $x \in I$ and so $I \in \mathfrak{I}_x$. This proves $\mathfrak{F}'_x = \mathfrak{I}_x$. The corresponding equality in (2) follows dually. (3): Let $z \in D$. Since $\mathfrak{F}_z = \mathfrak{F}_{z_\sqcap}$, we obtain $\mathfrak{F}'_z = \mathfrak{F}'_{z_\sqcap} = \mathfrak{I}_{z_\sqcap}$ by (1). The second half of (3) follows dually. □

The equivalence relation $\underline{\sqcap}$ on a double Boolean algebra $\underline{D}$ defined by $x \underline{\sqcap} y :\Leftrightarrow x \sqsubseteq y$ and $y \sqsubseteq x$ ($\Leftrightarrow x_\sqcap = y_\sqcap$ and $x_\sqcup = y_\sqcup$) has as restriction the identity on $D_p := D_\sqcap \cup D_\sqcup$, which is the largest pure subalgebra of $\underline{D}$. Especially, $\underline{\sqcap}$ is a congruence relation of $\underline{D}$ because, for each proper algebraic term $t(x_1, \ldots, x_n)$, the relationships $a_1 \underline{\sqcap} b_1, \ldots, a_n \underline{\sqcap} b_n$ even imply $t(a_1, \ldots, a_n) = t(b_1, \ldots, b_n)$. For formalizing the desired representation theorem we need the following notion: a map α between quasi-ordered sets is called *quasi-injective* if it satisfies the equivalence $x \sqsubseteq y \Leftrightarrow \alpha(x) \sqsubseteq \alpha(y)$.

Theorem 2 *Let $\underline{D}$ be a double Boolean algebra. Then $x \mapsto (\mathfrak{F}_x, \mathfrak{I}_x)$ describes a quasi-injective homomorphism ι from $\underline{D}$ to $\underline{\mathfrak{P}}(\mathbb{K}(\underline{D}))$ having $\underline{\sqcap}$ as kernel, and each map $\beta : \iota(D) \longrightarrow D$ satisfying $\iota\beta(\mathfrak{x}) = \mathfrak{x}$ is an injective homomorphism from the image of ι into the double Boolean algebra $\underline{D}$.*

Proof: By Lemma 3, we obtain $\mathfrak{F}'_x = \mathfrak{I}''_x$ and $\mathfrak{I}'_x = \mathfrak{F}''_x$; hence $(\mathfrak{F}_x, \mathfrak{I}_x)$ is a protoconcept of $\mathbb{K}(\underline{D})$ for all $x \in D$. For $x_\sqcap \not\sqsubseteq y_\sqcap$ in $\underline{D}_\sqcap$ there exists always an $F \in \mathfrak{F}_p(\underline{D})$ with $x_\sqcap \in F$ but $y_\sqcap \notin F$; hence $\mathfrak{F}_x = \mathfrak{F}_{x_\sqcap} \neq \mathfrak{F}_{y_\sqcap} = \mathfrak{F}_y$ and so $(\mathfrak{F}_x, \mathfrak{I}_x) \neq (\mathfrak{F}_y, \mathfrak{I}_y)$. Such inequality can be dually obtained for $y_\sqcup \not\sqsubseteq x_\sqcup$ in $\underline{D}_\sqcup$. If $x_\sqcap \sqsubseteq y_\sqcap$ and $y_\sqcap \sqsubseteq x_\sqcap$, then $x_\sqcap = y_\sqcap$ and $x_\sqcup = y_\sqcup$; hence $(\mathfrak{F}_x, \mathfrak{I}_x) = (\mathfrak{F}_y, \mathfrak{I}_y)$. Therefore $x \mapsto (\mathfrak{F}_x, \mathfrak{I}_x)$ describes a quasi-injective map ι from $\underline{D}$ to $\underline{\mathfrak{P}}(\mathbb{K}(\underline{D}))$ having $\underline{\sqcap}$ as kernel. It is even a homomorphism because we can show that $\mathfrak{F}_{x \sqcap y} = \mathfrak{F}_x \cap \mathfrak{F}_y$, $\mathfrak{I}_{x \sqcup y} = \mathfrak{I}_x \cap \mathfrak{I}_y$, $\mathfrak{F}_{\neg x} = \mathfrak{F}_p(\underline{D}) \setminus \mathfrak{F}_x$, and $\mathfrak{I}_{\lrcorner x} = \mathfrak{I}_p(\underline{D}) \setminus \mathfrak{I}_x$. These equalities result from the following equivalences and their duals: $F \in \mathfrak{F}_{x \sqcap y} \Leftrightarrow x \sqcap y \in F \Leftrightarrow x, y \in F \Leftrightarrow F \in \mathfrak{F}_x \cap \mathfrak{F}_y$ and $F \in \mathfrak{F}_{\neg x} \Leftrightarrow \neg x \in F \Leftrightarrow \neg(x_\sqcap) \in F \Leftrightarrow x_\sqcap \notin F \Leftrightarrow x \notin F \Leftrightarrow F \in \mathfrak{F}_p(\underline{D}) \setminus \mathfrak{F}_x$. Now, let β be a map from $\iota(D)$ into D satisfying $\iota\beta(\mathfrak{x}) = \mathfrak{x}$. Obviously, β is injective and even bijective on $\iota(D_p)$. Since the operations of $\underline{D}$ always result in elements of D_p, we obtain $\beta\iota(x) \sqcap \beta\iota(y) = \beta\iota(\beta\iota(x) \sqcap \beta\iota(y)) = \beta(\iota\beta\iota(x) \sqcap \iota\beta\iota(y)) = \beta(\iota(x) \sqcap \iota(y))$ and, analogously, the compatibility conditions for the other operations; hence β is an injective homomorphism. □

By Theorem 2, if $t_1(x_1, \ldots, x_n) = t_2(x_1, \ldots, x_n)$ is a non-trivial equation valid in all algebras of protoconcepts, then, for $a_1, \ldots, a_n \in D$, we get $t_1(\iota(a_1), \ldots, \iota(a_n)) = t_2(\iota(a_1), \ldots, \iota(a_n))$ and even
$t_1(a_1, \ldots, a_n) = t_1(\beta\iota(a_1), \ldots, \beta\iota(a_n)) = t_2(\beta\iota(a_1), \ldots, \beta\iota(a_n)) = t_2(a_1, \ldots, a_n)$ if in additon $\beta\iota(a_i) = a_i$ for $i = 1, \ldots, n$, the concequene of which is that $a_i \underline{\sqcap} a_j$ has to imply $a_i = a_j$. Since $a_i \underline{\sqcap} a_j$ $(i < j)$ generally yields the equality $t_k(a_1, \ldots, a_i, \ldots, a_j, \ldots, a_n) = t_k(a_1, \ldots, a_i, \ldots, a_i, \ldots, a_n)$ for $k = 1, 2$, the equation $t_1(x_1, \ldots, x_n) = t_2(x_1, \ldots, x_n)$ is valid in all double Boolean algebras too. Thus, as a corollary of Theorem 1 and Theorem 2, we obtain the following basic result of Boolean Concept Logic:

Corollary 1 *The equational axioms of double Boolean algebras generate the equational theory of the algebras of protoconcepts.*

For a finite double Boolean algebra $\underline{D}$, the elements of $\mathfrak{F}_p(\underline{D})$ are just the principal filters $[a) := \{x \in D \mid a \sqsubseteq x\}$ where a is an atom of the Boolean algebra $\underline{D}_\sqcap$, and the elements of $\mathfrak{I}_p(\underline{D})$ are just the principal ideals $(c] := \{x \in D \mid x \sqsubseteq c\}$ where c is a coatom of the Boolean algebra $\underline{D}_\sqcup$; furthermore, $[a) \triangle (c] \Leftrightarrow a \sqsubseteq c$. Therefore, the formal context $(A(\underline{D}_\sqcap), C(\underline{D}_\sqcup), \sqsubseteq)$, for which $A(\underline{D}_\sqcap)$ is the set of all atoms of $\underline{D}_\sqcap$ and $C(\underline{D}_\sqcup)$ is the set of all coatoms of $\underline{D}_\sqcup$, can be viewed as a simplified version of the standard context of $\underline{D}$. Substituting this simplified version in Theorem 2 yields the following corollary:

Corollary 2 *Let $\underline{D}$ be a finite double Boolean algebra. Then $x \mapsto (\{a \in A(\underline{D}_\sqcap) \mid a \sqsubseteq x\}, \{c \in C(\underline{D}_\sqcup) \mid x \sqsubseteq c\})$ describes a quasi-injective homomorphism ι from $\underline{D}$ to $\underline{\mathfrak{P}}(A(\underline{D}_\sqcap), C(\underline{D}_\sqcup), \sqsubseteq)$ which maps D_p bijectively onto $\mathfrak{H}(A(\underline{D}_\sqcap), C(\underline{D}_\sqcup), \sqsubseteq)$.*

4 Concept Algebras

As already discussed in Section 1, within a concept lattice $\underline{\mathfrak{B}}(\mathbb{K})$ with $\mathbb{K} := (G, M, I)$, there is only a *weak negation* of formal concepts (A, B) defined by $\neg_\sqcup(A, B) := ((G \setminus A)'', (G \setminus A)')$, which equals $(\neg(A, B))_\sqcup$ in $\underline{\mathfrak{P}}(\mathbb{K})$; dually, we also define a *weak opposition* by $\multimap_\sqcap(A, B) := ((M \setminus B)', (M \setminus B)'')$, which equals $(\multimap(A, B))_\sqcap$ in $\underline{\mathfrak{P}}(\mathbb{K})$. It follows that $A \cup (G \setminus A)'' = G$, but $A \cap (G \setminus A)''$ might not be disjoint and, dually, that $B \cap (M \setminus B)'' = \emptyset$, but $B \cup (M \setminus B)''$ might not be equal to M. In case the incidences gIm in $\mathbb{K}$ mean that the object g may have the attribute m, the elements of $A \cap (G \setminus A)''$ may be interpreted as those objects which may or may not have all attributes of B and the elements of $B \cup (M \setminus B)''$ may be interpreted as those attributes which all objects of A may or may not have. The concept lattice $\underline{\mathfrak{B}}(\mathbb{K})$ together with the weak negation $\neg_\sqcup$ and the weak opposition $\multimap_\sqcap$ as unary operations and the constants $\underline{\bot}$ and $\overline{\top}$ as nullary operations shall be viewed as the algebra $\underline{\mathfrak{A}}(\mathbb{K}) := (\underline{\mathfrak{B}}(\mathbb{K}), \sqcap, \sqcup, \neg_\sqcup, \multimap_\sqcap, \underline{\bot}, \overline{\top})$, called the *concept algebra* of the formal context $\mathbb{K}$.

To obtain a better structural understanding of concept algebras, we introduce the notion of dicomplemented lattices by the following definition: A *dicomplemented lattice* is defined as an algebra $\underline{L} := (L, \wedge, \vee, ^\Delta, ^\nabla, 0, 1)$ of type $(2, 2, 1, 1, 0, 0)$ for which $(L, \wedge, \vee, 0, 1)$ is a bounded lattice and the following conditions hold:

14a) $x^{\Delta\Delta} \leq x$ 14b) $x \leq x^{\nabla\nabla}$

15a) $(x \vee y)^\Delta \leq x^\Delta$ 15b) $x^\nabla \leq (x \wedge y)^\nabla$

16a) $(x \wedge y) \vee (x \wedge y^\Delta) = x$ 16b) $x = (x \vee y) \wedge (x \vee y^\nabla)$

17a) $w \wedge x^\Delta \wedge y_1^\Delta \wedge \cdots \wedge y_n^\Delta \leq x$ and $w^\Delta \wedge x^\Delta \wedge y_1^\Delta \wedge \cdots \wedge y_n^\Delta \leq y_i$ imply $y_1^\Delta \wedge \cdots \wedge y_n^\Delta \leq x$ for each $i \in \{1, \ldots, n\}$.

17b) $w \vee x^\nabla \vee y_1^\nabla \vee \cdots \vee y_n^\nabla \geq x$ and $w^\nabla \vee x^\nabla \vee y_1^\nabla \vee \cdots \vee y_n^\nabla \geq y_i$ imply $y_1^\nabla \vee \cdots \vee y_n^\nabla \geq x$ for each $i \in \{1, \ldots, n\}$.

The conditions 14), 15) and 16) present (up to equivalence) equations, but 17) yields only "quasi-equations" where a *quasi-equation* is defined as an implication between equations. Thus, the dicomplemented lattices form a quasi-equational class of algebraic structures. Clearly, $x \mapsto x^{\Delta\Delta}$ is an interior operator on the underlying set L by 14a),15a) and, dually, $x \mapsto x^{\nabla\nabla}$ is a closure operator on L by 14b),15b); in particular, also by 16),

$$x^{\nabla\nabla\nabla} = x^{\nabla} \leq x^{\Delta} = x^{\Delta\Delta\Delta} \text{ and } x^{\Delta\nabla} \leq x^{\Delta\Delta} \leq x \leq x^{\nabla\nabla} \leq x^{\nabla\Delta}.$$

Furthermore, 16a) and 16b) yield $x \vee x^{\Delta} = 1$ and $x \wedge x^{\nabla} = 0$, $0^{\Delta} = 1$ and $1^{\nabla} = 0$, respectively. The pair $x^{\diamond} := (x^{\nabla}, x^{\Delta})$ is called the *dicomplement* of x. Every bounded lattice can trivially be made to a dicomplemented lattice by defining $(1,1)$, $(0,0)$, and $(0,1)$ as the dicomplements of 0, of 1, and of each element unequal 0 and 1, respectively. Less trivial examples are given by finite distributive lattices in which x^{∇} and x^{Δ} are chosen as the pseudo-complement and the dual pseudo-complement of x. The algebraic structures of both classes can be verified as examples of dicomplemented lattices by the following theorem using as formal contexts $(L, L, \leq)$ for bounded lattices $\underline{L}$ and $(J(\underline{D}), M(\underline{D}), \leq)$ for finite distributive lattices $\underline{D}$.

Theorem 3 *For each formal context $\mathbb{K} := (G, M, I)$, the concept algebra $\underline{\mathfrak{A}}(\mathbb{K})$ $:= (\underline{\mathfrak{B}}(\mathbb{K}), \sqcap, \sqcup, \neg_{\sqcup}, \neg_{\sqcap}, \underline{\perp}, \overline{\top})$ is a dicomplemented lattice.*

Proof: Clearly, $(\underline{\mathfrak{B}}(\mathbb{K}), \sqcap, \sqcup, \underline{\perp}, \overline{\top})$ is a bounded lattice. The conditions 14a), 14b), 15a), and 15b) follow directly from the definition of the unary operations. For proving 16a) we consider $(A,B),(C,D) \in \underline{\mathfrak{B}}(\mathbb{K})$; for the extent of $((A,B) \sqcap (C,D)) \sqcup ((A,B) \sqcap \neg_{\sqcup}(C,D))$ it can be easily seen that $((A \cap C) \cup (A \cap (G \setminus C)''))'' \subseteq A$ and $((A \cap C) \cup (A \cap (G \setminus C)''))'' \supseteq ((A \cap C) \cup (A \cap (G \setminus C)))'' = A$. This yields the desired equality. For proving 17a) we consider $(A,B),(C,D),(E_1,F_1),\ldots(E_n,F_n) \in \underline{\mathfrak{B}}(\mathbb{K})$ with $\neg_{\sqcup}(E_1,F_1) \wedge \cdots \wedge \neg_{\sqcup}(E_n,F_n) \not\leq (C,D)$. Then, for each $i \in \{1,\ldots,n\}$, there exists a $g_i \in \overline{E_i}''$ with $g_i \notin C$ and $g_i \notin E_i$ because $\overline{E_i} \subseteq C$ would imply $\overline{E_i}'' \subseteq C$ which contradicts $\overline{E_1}'' \cap \cdots \cap \overline{E_n}'' \not\subseteq C$. Since $g_i \in \overline{C}''$, we obtain $g_i \in \overline{C}'' \cap \overline{E_1}'' \cap \cdots \cap \overline{E_n}''$ and hence, by $g_i \in A$ or $g_i \in \overline{A}''$, the alternative $g_i \in A \cap \overline{C}'' \cap \overline{E_1}'' \cap \cdots \cap \overline{E_n}''$ or $g_i \in \overline{A}'' \cap \overline{C}'' \cap \overline{E_1}'' \cap \cdots \cap \overline{E_n}''$. Because of $g_i \notin C$ and $g_i \notin E_i$, we can now conclude that $(A,B) \wedge \neg_{\sqcup}(C,D) \wedge \neg_{\sqcup}(E_1,F_1) \wedge \cdots \wedge \neg_{\sqcup}(E_n,F_n) \not\leq (C,D)$ or $\neg_{\sqcup}(A,B) \wedge \neg_{\sqcap}(C,D) \wedge \neg_{\sqcup}(E_1,F_1) \wedge \cdots \wedge \neg_{\sqcup}(E_n,F_n) \not\leq (E_i,F_i)$. Thus, 17a) is proved for concept algebras. 16b) and 17b) follow dually. □

Conversely, it can be shown that each dicomplemented lattice can be embedded into the algebra of protoconcepts of some suitable context. For the construction of those contexts, we need the notion of a primary filter and a primary ideal in a dicomplemented lattice $\underline{L} := (L, \wedge, \vee, ^{\Delta}, ^{\nabla}, 0, 1)$: A subset F of L is called a *filter* of $\underline{L}$ if $x \in F$ and $x \leq y$ imply $y \in F$ and if $x \in F$ and $y \in F$ imply $x \wedge y \in F$; an *ideal* is dually defined. A subset F_0 is called a *base* of the filter F if $F = \{y \in L \mid x \leq y \text{ for some } x \in F_0\}$; again, a base of an ideal is dually defined. A filter F of $\underline{L}$ is called *primary* if there is an ideal I and an element x of $\underline{L}$

with $x^\Delta \notin I$ such that F is maximal under all filters which are disjoint from I and do not contain x, but contain w or w^Δ for all $w \in L$. The set of all primary filters of $\underline{L}$ is denoted by $\mathfrak{F}_{pr}(\underline{L})$. Dually, primary ideals of $\underline{L}$ are defined; the set of all those ideals is denoted by $\mathfrak{I}_{pr}(\underline{L})$. The desired context for representing $\underline{L}$ can now be defined by $\mathbb{K}(\underline{L}) := (\mathfrak{F}_{pr}(\underline{L}), \mathfrak{I}_{pr}(\underline{L}), \triangle)$ with $F \triangle I :\Leftrightarrow F \cap I = \emptyset$; $\mathbb{K}(\underline{L})$ is called the *standard context* of the dicomplemented lattice $\underline{L}$. For $x \in L$, let $\mathfrak{F}_x := \{F \in \mathfrak{F}_{pr}(\underline{L}) \mid x \in F\}$ and $\mathfrak{I}_x := \{I \in \mathfrak{I}_{pr}(\underline{L}) \mid x \in I\}$. We prepare the desired representation theorem by three lemmas:

Lemma 4 *For elements x and y in $\underline{L}$ the following conditions hold:*
(1) $x \wedge y^\Delta \leq y \Leftrightarrow x \leq y$
(2) $x^\Delta \leq y \Leftrightarrow y^\Delta \leq x$

Proof: (1): Let $x \wedge y^\Delta \leq y$. Together with 16a) it follows $x = (x \wedge y) \vee (x \wedge y^\Delta) = x \wedge y$ and hence $x \leq y$. The converse is obvious.
(2): 14a) and 15a) yield the following equivalences: $x^\Delta \leq y \Leftrightarrow y^\Delta \leq x^{\Delta\Delta} \Leftrightarrow y^\Delta \leq x$. □

Lemma 5 *For any ideal I and any element $x \in L \setminus I$ with $x^\Delta \notin I$, there exists a filter F with $F \cap (\{x\} \cup I) = \emptyset$ and $w^\Delta \in F$ for all $w \in L \setminus F$.*

Proof: Let I be an ideal and $x \in L \setminus I$ with $x^\Delta \notin I$. For $S \subseteq L$ we generally define $S^\Delta := \{s^\Delta \mid s \in S\}$ and $S^{\underline{\Delta}} := \{\bigwedge T^\Delta \mid T \text{ is a finite subset of } S\}$. Since $(y \vee z)^\Delta \leq y^\Delta \wedge z^\Delta$ by 15a), $I^\Delta (= I^{\underline{\Delta}})$ is a base of a filter. This filter is disjoint from I and does not contain x because $y^\Delta \leq z$ for some $y, z \in I$ would imply $1 = y \vee y^\Delta \in I$ which contradicts $x^\Delta \notin I$, and $y^\Delta \leq x$ for some $y \in I$ would imply $x^\Delta \leq y$ by Lemma 4(2) which also contradicts $x^\Delta \notin I$.

In general, for each $J \subseteq L$, we define the filter $F_J := \{y \in L \mid y \geq z \text{ for some } z \in J^{\underline{\Delta}}\}$ and $\mathfrak{F}(x, I)$ to be the set of all those filters F_J which satisfy $\{x\} \cup I \subseteq J$ and $(\{x\} \cup I) \cap F_J = \emptyset$; furthermore, let $\mathfrak{J}(x, I)$ be the set consisting of all such J with $F_J \in \mathfrak{F}(x, I)$. By Lemma 4, for $y, z \in I$, $x^\Delta \wedge y^\Delta \leq x$ would imply $y^\Delta \leq x$ and hence $x^\Delta \leq y \in I$ and $x^\Delta \wedge y^\Delta \leq z$ would imply $x^\Delta \wedge (y \vee z)^\Delta \leq y \vee z$ and hence $x^\Delta \leq y \vee z \in I$ which both contradict the assumption $x^\Delta \notin I$. Therefore, $y \in (\{x\} \cup I) \cap F_{(\{x\} \cup I)} = \emptyset$ and hence $\{x\} \cup I \in \mathfrak{J}(x, I)$. Thus, $\mathfrak{J}(x, I)$ is not empty. Let $\mathfrak{C}$ be a non-empty chain in $(\mathfrak{J}(x, I), \subseteq)$. It can be easily checked that $F_{\bigcup \mathfrak{C}} \in \mathfrak{F}(x, I)$. Thus, by Zorn's Lemma, there exits a maximal set $\tilde{J}$ in $(\mathfrak{J}(x, I), \subseteq)$.

Let $w \in L$ and let Y be a finite subset of $\tilde{J}$. Because of $\bigwedge Y^\Delta \not\leq x$, we obtain by 17a) for each $y \in Y$ the alternatives

$$(w^\Delta \wedge x^\Delta \wedge \bigwedge Y^\Delta \not\leq x \text{ or } w^{\Delta\Delta} \wedge x^\Delta \wedge \bigwedge Y^\Delta \not\leq y) \text{ and}$$
$$(w^\Delta \wedge x^\Delta \wedge \bigwedge Y^\Delta \not\leq y \text{ or } w^{\Delta\Delta} \wedge x^\Delta \wedge \bigwedge Y^\Delta \not\leq x)$$

Suppose $w^\Delta \wedge x^\Delta \wedge \bigwedge Y^\Delta \leq x$. Then $w^{\Delta\Delta} \wedge x^\Delta \wedge \bigwedge Y^\Delta \not\leq y$ for each $y \in Y$. We also have $w^{\Delta\Delta} \wedge x^\Delta \wedge \bigwedge Y^\Delta \not\leq x$, because otherwise it would follow by 16a) that $x^\Delta \wedge \bigwedge Y^\Delta = (w^\Delta \wedge x^\Delta \wedge \bigwedge Y^\Delta) \vee (w^{\Delta\Delta} \wedge x^\Delta \wedge \bigwedge Y^\Delta) \leq x$ and hence $\bigwedge Y^\Delta \leq x$ by Lemma 4 which contradicts $x \notin F_{\tilde{J}}$. If we replace Y by any finite subset $\tilde{Y}$ of $\tilde{J}$,

then the inequality and the two negated inequalities have still to be true; hence $w^{\Delta\Delta} \wedge \bigwedge Y^{\Delta} \not\in I$ for all finite subsets Y of $\tilde{J}$. Furthermore, $w^{\Delta\Delta} \wedge \bigwedge Y^{\Delta} \not\leq w^{\Delta}$ for each finite subset Y of $\tilde{J}$, because otherwise, by Lemma 4(1), we would have $\bigwedge Y^{\Delta} \leq w^{\Delta}$ which contradicts $x^{\Delta} \wedge \bigwedge Y^{\Delta} = w^{\Delta} \wedge x^{\Delta} \wedge \bigwedge Y^{\Delta} \leq x$. Thus, $(\tilde{J} \cup \{w^{\Delta}\}) \cap F_{\tilde{J} \cup \{w^{\Delta}\}} = \emptyset$ and $x \not\in F_{\tilde{J} \cup \{w^{\Delta}\}}$. Now, the maximality of $\tilde{J}$ yields $w^{\Delta} \in \tilde{J}$ and hence $w^{\Delta\Delta} \in F_{\tilde{J}}$. Analogously, $w^{\Delta\Delta} \wedge x^{\Delta} \wedge \bigwedge Y^{\Delta} \leq x$ leads to $w^{\Delta} \in F_{\tilde{J}}$. Finally, we have to consider the case: $w^{\Delta} \wedge x^{\Delta} \wedge \bigwedge Y^{\Delta} \not\leq x$ and $w^{\Delta\Delta} \wedge x^{\Delta} \wedge \bigwedge Y^{\Delta} \not\leq x$ for all finite subsets Y of $\tilde{J}$. Suppose $w^{\Delta} \wedge x^{\Delta} \wedge \bigwedge Y^{\Delta} \leq y_1$ and $w^{\Delta\Delta} \wedge x^{\Delta} \wedge \bigwedge Y^{\Delta} \leq y_2$ for some finite subset Y of $\tilde{J}$ and $y_1, y_2 \in Y \cap I$; then we would get $x^{\Delta} \wedge \bigwedge Y^{\Delta} = (w^{\Delta} \wedge x^{\Delta} \wedge \bigwedge Y^{\Delta}) \vee (w^{\Delta\Delta} \wedge x^{\Delta} \wedge \bigwedge Y^{\Delta}) \leq y_1 \vee y_2 \in I$ which contradicts $F_{\tilde{J}} \cap (\{x\} \cup I) = \emptyset$. Thus, without loss of generality, we can assume that $w^{\Delta\Delta} \wedge x^{\Delta} \wedge \bigwedge Y^{\Delta} \not\in I$ for all finite subsets Y of $\tilde{J}$. Consequently, $F_{\{w^{\Delta}\} \cup \tilde{J}} \cap (\{x\} \cup I) = \emptyset$. The maximality of $\tilde{J}$ yields again $w^{\Delta} \in \tilde{J}$ and hence $w^{\Delta\Delta} \in F_{\tilde{J}}$. Since $w \in L \setminus F_{\tilde{J}}$ implies $w^{\Delta\Delta} \in L \setminus F_{\tilde{J}}$, we finally obtain that $w^{\Delta} \in F_{\tilde{J}}$ for all $w \in L \setminus F_{\tilde{J}}$. □

Lemma 6 *The derivation in* $\mathbb{K}(\underline{L})$ *yields* $\mathfrak{F}'_x = \mathfrak{I}_x$ *and* $(\mathfrak{F}_{pr}(\underline{L}) \setminus \mathfrak{F}_x)' = \mathfrak{I}_{x^{\Delta}}$ *for all* $x \in L$.

Proof: Let $I \in \mathfrak{I}_x$. Then $x \in F \cap I$ for all $F \in \mathfrak{F}_x$; hence $I \in \mathfrak{F}'_x$. Conversely, let $I \in \mathfrak{I}_{pr}(\underline{L}) \setminus \mathfrak{I}_x$. Lemma 5 guarantees the existence of a filter F disjoint from I and not containing x^{Δ}, for which $w^{\Delta} \in F$ for all $w \in L \setminus F$; consequently, $x \in F$. By Zorn's Lemma, there exists a maximal filter $\tilde{F}$ disjoint from I and not containing x^{Δ}, but containing F. Obviously, $\tilde{F} \in \mathfrak{F}_x$; hence $I \not\in \mathfrak{F}'_x$. This proves $\mathfrak{F}'_x = \mathfrak{I}_x$. Now, let $I \in \mathfrak{I}_{x^{\Delta}}$. Since any primary filter F with $x \not\in F$ contains x^{Δ} by definition, we obtain $I \in (\mathfrak{F}_{pr}(\underline{L}) \setminus \mathfrak{F}_x)'$. Conversely, let $I \not\in \mathfrak{I}_{x^{\Delta}}$. Then, by Lemma 5, there is a filter F with $F \cap (\{x\} \cup I) = \emptyset$ and $w^{\Delta} \in F$ for all $w \in L \setminus F$. Therefore, a maximal filter disjoint from I and not containing x, but containing F, which exists by Zorn's Lemma, must be primary; hence $I \not\in (\mathfrak{F}_{pr}(\underline{L}) \setminus \mathfrak{F}_x)'$. This proves the second claim. □

Theorem 4 *For a dicomplemented lattice* $\underline{L}$*, the map* $x \mapsto (\mathfrak{F}_x, \mathfrak{I}_x)$ *is an injective homomorphism* ι *from* $\underline{L}$ *into* $\underline{\mathfrak{A}}(\mathbb{K}(\underline{L}))$.

Proof: By Lemma 6 and its dual, $(\mathfrak{F}_x, \mathfrak{J}_x)$ is a formal concept for each $x \in L$. For $x \not\leq y$ in $\underline{L}$, by Lemma 5 and Zorn's Lemma, there exist always an $F \in \mathfrak{F}_x$ with $y \not\in F$; hence $\mathfrak{F}_x \neq \mathfrak{F}_y$ and so $(\mathfrak{F}_x, \mathfrak{I}_x) \neq (\mathfrak{F}_y, \mathfrak{I}_y)$. Therefore $x \mapsto (\mathfrak{F}_x, \mathfrak{I}_x)$ describes an injective map from $\underline{L}$ into $\underline{\mathfrak{A}}(\mathbb{K}(\underline{L}))$. It is even a homomorphism because we can show that $\mathfrak{F}_{x \wedge y} = \mathfrak{F}_x \cap \mathfrak{F}_y$, $\mathfrak{I}_{x \vee y} = \mathfrak{I}_x \cap \mathfrak{I}_y$, $\mathfrak{F}_{x^{\Delta}} = (\mathfrak{F}_{pr}(\underline{L}) \setminus \mathfrak{F}_x)''$, $\mathfrak{I}_{x^{\nabla}} = (\mathfrak{I}_{pr}(\underline{L}) \setminus \mathfrak{I}_x)''$, $(\mathfrak{F}_0, \mathfrak{J}_0) = (\emptyset, \mathfrak{I}_{pr}(\underline{L}))$, and $(\mathfrak{F}_1, \mathfrak{J}_1) = (\mathfrak{F}_{pr}(\underline{L}), \emptyset)$. These equalities result from the following equivalences, equalities, and their duals: $F \in \mathfrak{F}_{x \wedge y} \Leftrightarrow x \wedge y \in F \Leftrightarrow x, y \in F \Leftrightarrow F \in \mathfrak{F}_x \cap \mathfrak{F}_y$, $\mathfrak{F}_{x^{\Delta}} = (\mathfrak{I}_{x^{\Delta}})' = (\mathfrak{F}_{pr}(\underline{L}) \setminus \mathfrak{F}_x)''$, and $\mathfrak{F}_0 = \emptyset$. □

Corollary 3 *The quasi-equational axioms of dicomplemented lattices generate the quasi-equational theory of the dicomplemented lattices.*

For a finite dicomplemented lattice $\underline{L}$, the elements of $\mathfrak{F}_{pr}(\underline{L})$ are just the principal filters generated by a $\vee$-primary element and the elements of $\mathfrak{I}_{pr}(\underline{L})$ are just the principal ideals generated by a $\wedge$-primary element; at this, an element $a \neq 0$ of $\underline{L}$ is called $\vee$-*primary* if a covers at most two elements and if $x \not\geq a$ always implies $x^{\Delta} \geq a$, and an element $a \neq 1$ of $\underline{L}$ is called $\wedge$-*primary* if a is covered by at most two elements and if $x \not\leq a$ always implies $x^{\Delta} \leq a$. Every $\vee$-irreducible element is $\vee$- *primary* by 16a), and every $\wedge$-irreducible element is $\wedge$-*primary* by 16b). The set of all $\vee$-primary elements of $\underline{L}$ is denoted by $J_{pr}(\underline{L})$, and the set of all $\wedge$-primary elements of $\underline{L}$ is denoted by $M_{pr}(\underline{L})$.

Corollary 4 *Let $\underline{L}$ be a finite dicomplemented lattice. Then $x \mapsto (\{a \in J_{pr}(\underline{L}) \mid a \leq x\}, \{c \in M_{pr}(\underline{L}) \mid x \leq c\})$ describes an isomorphism from $\underline{L}$ onto the concept algebra $\underline{\mathfrak{A}}(J_{pr}(\underline{L}), M_{pr}(\underline{L}), \leq)$.*

5 Further Research

This paper can only be considered as the beginning of developing Boolean Concept Logic. After identifying double Boolean algebras and dicomplemented lattices as basic algebraic structures for Boolean Concept Logic, the word problems for those two classes of algebras should be attacked which, in particular, sets the task of determining the free double Boolean algebras and the free dicomplemented lattices. In this scope, the question arises whether there are useful normal forms for algebraic terms concerning double Boolean algebras and dicomplemented lattices. Solutions to those problems and tasks will lead to further questions about the algorithmic treatment concerning the solutions. For the logical analysis of data contexts, a developed structure theory of algebras of protoconcepts and concept algebras would be desirable. In particular, constructions, decompositions, and interesting properties should be investigated for those algebras. A main direction of research will be determined by the needs concerning a successful development of Contextual Logic; in particular, the development of Contextual Judgment Logic based on concept graphs (see [Wi97a], [Wi98], [Pr98], [PW99], [MSW99]) would benefit from substantial results in Boolean Concept Logic. A measure for successful research will always be given by the usefulness of its results for conceptual knowledge representation and processing.

Acknowledgement

This paper has substantially benefited from the financial support given by the Australian Research Council (IREX-Programme) which allowed the author to work for two months at the School of Information Technology of the Griffith University (Gold Coast Campus).

References

BM74. W. Beierwaltes, A. Menne: Gegensatz. In: J. Ritter†, K. Gründer (eds.): *Historisches Wörterbuch der Philosophie*. Bd.3. Schwabe & Co., Basel 1974, 105–119.

Bo54. G. Boole: *An investigation of the laws of thought, on which are founded the mathematical theories of logic and probabilities*. Macmillan 1854. Reprinted by Dover Publ., New York 1958.

DP92. B. A. Davey, H. Priestley: *Introduction to lattices and order*. Cambridge University Press, Cambridge 1990.

GW96. B. Ganter, R. Wille: *Formal Concept Analysis: Mathematical Foundations*. Springer, Heidelberg 1999; German version: Springer, Heidelberg 1996.

GW99. B. Ganter and R. Wille: Contextual attribute logic. In: W. Tepfenhart, W. Cyre (eds.): *Conceptual Structures: Standards and Practices*. LNAI **1640**. Springer, Heidelberg 1999, 377–388.

HLSW99. C. Herrman, P. Luksch, M. Skorsky, R. Wille: Algebras of semiconcepts and double Boolean algebras. Contributions to General Algebra **13** (to appear)

KL67. W. Kamlah, P. Lorenzen: *Logische Propädeutik. Vorschule des vernünftigen Redens*. B.I.-HTB 227, Mannheim 1967.

Ka88. I. Kant: *Logic*. Dover, New York 1988.

LW91. P. Luksch, R. Wille: A mathematical model for conceptual knowledge systems. In: H.-H. Bock, P. Ihm (eds.): *Classification, data analysis, and knowledge organization*. Springer, Heidelberg 1991, 156–162.

Me84. A. Menne: Negation. In: J. Ritter†, K. Gründer (eds.): *Historisches Wörterbuch der Philosophie*. Bd.6. Schwabe & Co., Basel 1984, 666–670.

MSW99. G. Mineau, G. Stumme, R. Wille: Conceptual structures represented by conceptual graphs and formal concept analysis. In: W. Tepfenhart, W. Cyre (eds.): *Conceptual Structures: Standards and Practices*. LNAI **1640**. Springer, Heidelberg 1999, 423–441.

Pr98. S. Prediger: *Kontextuelle Urteilslogik mit Begriffsgraphen. Ein Beitrag zur Restrukturierung der mathematischen Logik*. Dissertation, TU Darmstadt. Shaker Verlag, Aachen 1998.

PW99. S. Prediger, R. Wille: The lattice of concept graphs of a relationally scaled context. In: W. Tepfenhart, W. Cyre (eds.): *Conceptual Structures: Standards and Practices*. LNAI **1640**. Springer, Heidelberg 1999, 401–414.

Wi92. R. Wille: Concept lattices and conceptual knowledge systems. *Computers & Mathematics with Applications* **23** (1992), 493–515.

Wi96. R. Wille: Restructuring mathematical logic: an approach based on Peirce's pragmatism. In: A. Ursini, P. Agliano (eds.): *Logic and Algebra*. Marcel Dekker, New York 1996, 267–281.

Wi97a. R. Wille: Conceptual Graphs and Formal Concept Analysis. In: D. Lukose, H. Delugach, M. Keeler, L. Searle, J. F. Sowa (eds.): *Conceptual structures: fulfilling Peirce's dream*. LNAI **1257**. Springer, Heidelberg 1997, 290–303.

Wi97b. R. Wille: Conceptual landscapes of knowledge: a pragmatic paradigm for knowledge processing. In: W. Gaul, H. Locarek-Junge (eds.): *Classification in the Information Age*. Springer, Heidelberg 1999, 344–356; already printed in: G. Mineau, A. Fall (eds.): *Proc. 2nd Intl. KRUSE*. Simon Fraser University, Vancouver 1997, 2–13.

Wi98. R. Wille: Triadic Concept Graphs. In: M.-L. Mugnier, M. Chein (eds.): *Conceptual Structures: Theory, Tools and Applications*. LNAI **1453**, Springer, Heidelberg 1998, 194–208.

Lattices of Triadic Concept Graphs

Bernd Groh[1] and Rudolf Wille[2]

[1] School of Information Technology, Griffith University, Gold Coast Campus,
PMB 50 Gold Coast Mail Centre, QLD 9726, Australia
b.groh@gu.edu.au
[2] Fachbereich Mathematik, Technische Universität Darmstadt,
Schloßgartenstr. 7, D–64289 Darmstadt
wille@mathematik.tu-darmstadt.de

Abstract. Triadic concept graphs have been introduced as a mathematization of conceptual graphs with subdivision. In this paper it is shown that triadic concept graphs of a triadic power context family always form a complete lattice with respect to the generalization order. For stating this result, a clarification of the notion of generalization is needed. It turns out that the generalization order may be differently defined, depending on the assumed background knowledge, respectively.

Contents

1. Triadic Concept Graphs
2. The Generalization Order
3. Lattices of Triadic Concept Graphs

1 Triadic Concept Graphs

The aim of this paper is to show that the triadic concept graphs of a triadic power concept family (see [Wi98]) always form a complete lattice with respect to the generalization order (cf. [PW99]). For this we have to clarify the notion of generalization for triadic concept graphs which, of course, presupposes a basic understanding of triadic power context families and their conceptual structures. Basic definitions and results of Dyadic Concept Analysis may be found in [GW96] and of Triadic Concept Analysis in [LW95],[Bi98],[WZ99].

Before recalling the definition of a triadic power context family and its triadic concept graphs, we start with an example from music concerned with the "logic" of the common musical notation. Figure 1 shows the notation of a simple melody modulating from g major via d major to $b\flat$ major. The added boxes and circles yield a conceptual graph with a subdivision indicating the domains of the three keys. The notes without accidentals are references, the accidentals $\natural$, $\sharp$, and $\flat$ combined with a key are concept names ($\natural$ may be added in all concept boxes without an accidental), and the intervals $\pm 2nd$ (minor second), $\pm 2ND$ (major second), $\pm 3rd$ (minor third), $\pm 3RD$ (major third), and $\pm 4th$ (fourth) are the

B. Ganter and G.W. Mineau (Eds.): ICCS 2000, LNAI 1867, pp. 332–341, 2000.
© Springer-Verlag Berlin Heidelberg 2000

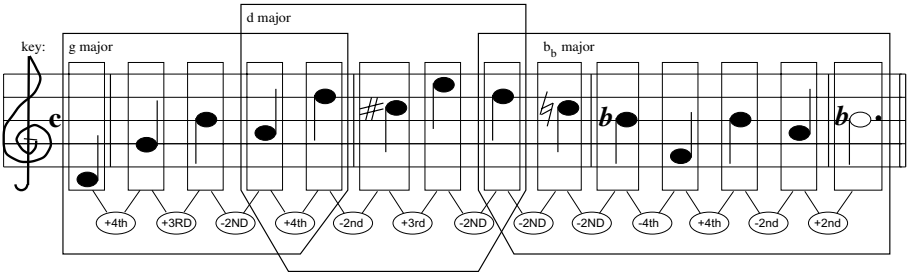


Fig. 1. The notation of a modulating melody represented as a conceptual graph

connecting binary relations. The relationships between notes, accidentals, keys, and intervals can be coded in formal contexts as follows (for simplifying the coding, we consider notes and intervals modulo octave):

The notes d, e, f, g, a, b, c are represented by the numbers $0, 1, 2, 3, 4, 5, 6$, the accidentals $\flat, \natural, \sharp$ by $-1, 0, 1$, and the major keys $c_\flat, g_\flat, d_\flat, a_\flat, e_\flat, b_\flat, f, c, g, d, a, e, b, f_\sharp, c_\sharp$ by $-7, \ldots, 0, \ldots, 7$, respectively. We define a triadic context $\mathbb{K}_0 := (G_0, M_0, B, Y_0)$ by

$$G_0 := \{0, 1, 2, 3, 4, 5, 6\},\ M_0 := \{-1, 0, 1\},\ B := \{-7, \ldots, 0, \ldots, 7\},\ \text{and}$$

$$(i, j, k) \in Y_0 :\Leftrightarrow \begin{cases} k \in B \setminus \{-7, 7\},\ j = 0, \text{ and} \\ \quad i = k - 4n (mod\, 7) \text{ with } n \in \{-5, \ldots, 0, 1\}, \\ k \in B \setminus \{0\},\ j = k : |k|^{-1}, \text{ and} \\ \quad i = j \cdot (2 + 4n) (mod\, 7) \text{ with } n \in \{0, \ldots, |k - 1|\}. \end{cases}$$

$(i, j, k) \in Y$ is read: the note i has the accidental y in the key k. A triadic diagram representing the concept trilattice of $\mathbb{K}_0$ is shown in Figure 2 where each black spot in the triangular net together with the linked spots to the right, the left, and above represent a triadic concept with its extent of notes, intent of accidentals, and modus of keys, respectively (triadic diagrams are explicitely described in [LW95],[Bi97]).

Now, we form a second triadic context for including musical intervals into our considerations. The (upward) intervals are represented by pairs of numbers as follows:

$1st$	$2nd$	$2ND$	$3rd$	$3RD$	$4th$	$4th_\sharp$	$5th_\flat$	$5th$	$6th$	$6TH$	$7th$	$7TH$
$(0,0)$	$(1,1)$	$(1,2)$	$(2,3)$	$(2,4)$	$(3,5)$	$(3,6)$	$(4,6)$	$(4,7)$	$(5,8)$	$(5,9)$	$(6,10)$	$(6,11)$

The set of these pairs of numbers shall be denoted by M_2. For $j \in G_0$, let $l_j := 2 \cdot j$ if $0 \leq j \leq 1$, $l_j := 2 \cdot j - 1$ if $2 \leq j \leq 5$, and $l_6 := 10$; clearly, $(0, l_j)$ represents just the interval from the note 0 to the note j in the key 0 (i.e. c major). The second triadic context is now defined by

$$\mathbb{K}_2 := (G_0 \times G_0, M_2, B, Y_2) \text{ with}$$
$$Y_2 := \{((i, j), (j - i (mod\, 7), l_{j-4k (mod\, 7)} - l_{i-4k (mod\, 7)} (mod\, 12)), k) \mid i, j \in G_0 \text{ and } k \in B\}.$$

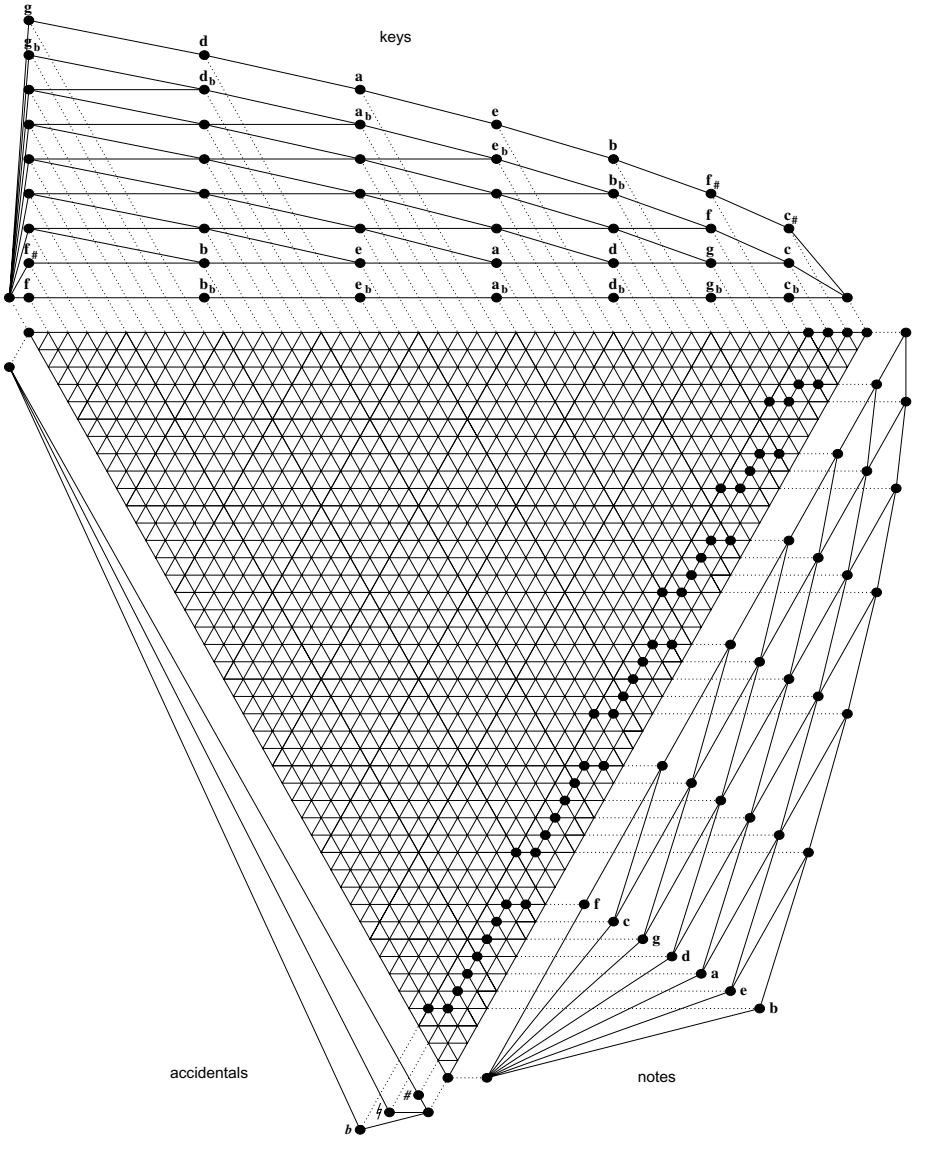


Fig. 2. A triadic diagram of the concept trilattice of the musical notation context

The formal contexts $\mathbb{K}_0$ and $\mathbb{K}_2$ form a triadic power context family which codes the basic relationships of the notation of tonal music. It allows, for instance, to represent the conceptual graph of Figure 1 as a mathematical object, i.e. a triadic concept graph. To make this understandable in general, we discuss in this paper basic relationships in triadic power context families and between their triadic concept graphs.

A *triadic power context family* of triadic contexts is a family $\vec{\mathbb{K}} \coloneqq (\mathbb{K}_0, \ldots, \mathbb{K}_n)$ $(n \geq 2)$ of triadic contexts $\mathbb{K}_k := (G_k, M_k, B, Y_k)$ $(k = 0, \ldots, n)$ such that $G_k \subseteq (G_0)^k$. If $G_k = \emptyset$ for some k, we may obmit $\mathbb{K}_k$ from the context listing. The concept trilattices $\underline{\mathfrak{T}}(\mathbb{K}_k)$ yield the fundamental relationships in $\vec{\mathbb{K}}$ between objects, attributes, modalities, extents, intents, modi, and triadic concepts as, for instance, represented in Figure 2 for the above defined music context $\mathbb{K}_0$. The formal relationships on the contextual and conceptual level extend to higher complexity on the judgment level by the formation of the triadic concept graphs of $\vec{\mathbb{K}}$.

For the definition of triadic concept graphs (without coreference links), first, an *abstract concept graph with subdivision* is specified as a structure $\mathfrak{G} := (V, E, \nu, C, \kappa, \sigma)$ for which

1. V and E are sets and ν is a mapping of E to $\bigcup_{k=1}^{n} V^k$ $(2 \leq n \in \mathbb{N})$ so that (V, E, ν) can be considered as a directed multi-hypergraph with vertices from V and edges from E (we define $|e| = k :\Leftrightarrow \nu(e) = (v_1, \ldots, v_k)$),
2. C is a set and κ is a mapping of $V \cup E$ to C such that $\kappa(e_1) = \kappa(e_2)$ always implies $|e_1| = |e_2|$ (the elements of C may be understood as abstract concepts),
3. σ is a partial mapping of the vertex set V to the power set of $V \times C$ such that, for the partial mappings σ_1 and σ_2 with $\sigma_1(w) := \{v \in V \mid (v, c) \in \sigma(w)$ for some $c \in C\}$ and $\sigma_2(w) := \{c \in C \mid (v, c) \in \sigma(w)$ for some $v \in V\}$ for $w \in dom(\sigma)$, we have $\sigma_1(V) \neq \emptyset \neq \sigma_2(V)$ and $v \notin (\sigma_1)^m(v)$ for all $v \in V$ and $m \in \mathbb{N}$ (for $c \in \sigma_2(w)$, the vertex set $A(c) := \{v \in V \mid (v, c) \in \sigma(w)\}$ is called the *area* of c within the vertex w). Additionally, we assume that there is a *universal vertex* $w^\top \in V \cap C$ with $\sigma(w^\top) = \{(v, w^\top) \mid v \in V \setminus \sigma_1(w)$ for all $w \in dom(\sigma) \setminus \{w^\top\}\}$.

The abstract concept graph with subdivision belonging to the conceptual graph in Figure 1 has as vertices the small rectangles and the large boxes (including the area of the whole diagram), as edges the links joining two rectangles via a circle, and as abstract concepts $(\natural, g\ major)$, $(\natural, \{g\ major, d\ major\})$, $(\natural, d\ major)$, $(\natural, \{d\ major, b_\flat\ major\})$, $(\natural, b_\flat\ major)$, $(\sharp, d\ major)$, $(\flat, b_\flat\ major)$, $g\ major$, $d\ major$, $b_\flat\ major$, and *key* (as the universal "vertex-concept"); furthermore, σ_1 maps a large box to the set of all rectangles it contains and σ_2 maps a large box to the abstract concept named in it. Hence, the retangles in a large box form the area of the corresponding abstract concept within that box.

In this example of an abstract concept graph we already have connections to a triadic power context family. What that means, mathematically, shall be specified next: Let $\vec{\mathbb{K}} := (\mathbb{K}_0, \ldots, \mathbb{K}_n)$ be a triadic power context family with $\mathbb{K}_k := (G_k, M_k, B, Y_k)$ $(0 \leq k \leq n)$ and let $C_{\vec{\mathbb{K}}} := \bigcup_{k=0}^{n} \underline{\mathfrak{T}}(\mathbb{K}_k)$; then an abstract concept graph $\mathfrak{G} := (V, E, \nu, C, \kappa, \sigma)$ with subdivision is called a *triadic concept graph over the triadic power context family* $\vec{\mathbb{K}}$ if

4. $C = C_{\vec{\mathbb{K}}}$,
5. $\kappa(V) \subseteq \underline{\mathfrak{T}}(\mathbb{K}_0)$,
6. $\kappa(e) \in \underline{\mathfrak{T}}(\mathbb{K}_k)$ for all $e \in E$ with $|e| = k$,

7. $\sigma_2(w) \subseteq G_0$ for all $w \in dom(\sigma)$,
8. $\mathfrak{c}_1 \in Ext(\mathfrak{c}_2), \mathfrak{c}_2 \in Ext(\mathfrak{c}_3), \ldots, \mathfrak{c}_{m-1} \in Ext(\mathfrak{c}_m)$ for $\mathfrak{c}_1, \mathfrak{c}_2, \ldots, \mathfrak{c}_m \in \sigma_2(V)$ imply $\mathfrak{c}_1 \neq \mathfrak{c}_m$.

A *realization* of such triadic concept graph $\mathfrak{G}$ over $\vec{\mathbb{K}}$ is defined to be a map ρ of V to the power set of G_0 which satisfies, for all $v \in V$, $w \in dom(\sigma)$, $\mathfrak{c} \in C_{\vec{\mathbb{K}}}$, and $e \in E$ with $\nu(e) = (v_1, \ldots, v_k)$, allowing the abreviations $\rho(g) := \rho(v_1) \times \cdots \times \rho(v_k)$ and $(e, \mathfrak{c}) \in \sigma(w) :\Leftrightarrow (v_1, \mathfrak{c}), \ldots, (v_k, \mathfrak{c}) \in \sigma(w)$, the conditions

9. $\emptyset \neq \rho(v) \subseteq (Int(\kappa(v)) \times Mod(\mathfrak{c}))^{(1)}$ if $(v, \mathfrak{c}) \in \sigma(w)$,
10. $\rho(e) \subseteq (Int(\kappa(e)) \times Mod(\mathfrak{c}))^{(1)}$ if $(e, \mathfrak{c}) \in \sigma(w)$,
11. $\sigma_2(w) \subseteq \rho(w)$.

The pair $\underline{\mathfrak{G}} := (\mathfrak{G}, \rho)$ is called a *realized triadic concept graph* of the triadic power context family $\vec{\mathbb{K}}$ or, shortly, a *triadic concept graph* of $\vec{\mathbb{K}}$ (cf. [Wi98]).

Now, let us prove that Figure 1 represents indeed a triadic concept graph of the triadic power context family $(\mathbb{K}_0, \mathbb{K}_2)$ of musical notation. Above we already discussed how the diagram in Figure 1 can be understood as a representation of an abstract concept graph which, by definition, satisfies the conditions 1, 2, and 3. It also satisfies 4, 5, and 6 under the assumption that κ maps each small rectangle to the triadic concept of $\mathbb{K}_0$ generated by the accidental of the rectangle together with the keys of the large boxes containing the rectangle, and each large box to the triadic concept of $\mathbb{K}_0$ uniquely determined by the key named in the box (cf. Figure 2), and each link to the triadic concept of $\mathbb{K}_2$ uniquely determined by an attribute, namely the number code of the interval named in the circle of the link. For confirming condition 7, we admit that each key is not only understood as a modality, but also as the triadic concept of $\mathbb{K}_0$ uniquely determined by the key and, following the principle of hypostatic abstraction, as a (general) object too; hence we assume that the keys understood as triadic concepts belong to the (extended) object set G_0 of $\mathbb{K}_0$, which yields the condition 7 and also 8. The realization map ρ has to assign to each small rectangle the number code of the note it contains, and to each large box the triadic concept of the key named in that box. Checking the conditions 9 to 11 is now straightforward.

2 The Generalization Order

In this section we discuss how an appropriate ordering may be defined on the set $\Gamma(\vec{\mathbb{K}})$ of all triadic concept graphs of a triadic power context family $\vec{\mathbb{K}} := (\mathbb{K}_0, \ldots, \mathbb{K}_n)$ consisting of the triadic contexts $\mathbb{K}_k := (G_k, M_k, B, Y_k)$ $(0 \leq k \leq n)$. For concepts the dominating ordering is the subconcept-superconcept-relation, also called the *generalization order*. Generalizing a concept means to select some attributes of the concept and to determine then a superconcept by them (cf. [Ac71]). This method cannot directly applied to triadic concepts, but if we proceed in selecting attributes while keeping the modalities of a triadic concept fixed, then we may obtain more general triadic concepts. This idea give rise to a generalization order on the set $\Gamma(\vec{\mathbb{K}})$ of triadic concept graphs which we capture by the following definitions:

For a triadic concept graph $\mathfrak{G} := (V, E, \nu, C_{\vec{\mathbb{K}}}, \kappa, \sigma, \rho)$ of the triadic power context family $\vec{\mathbb{K}}$, we define the *conceptual content* $C(\mathfrak{G}) := (C_0(\mathfrak{G}), \ldots, C_n(\mathfrak{G}))$ by

$$C_0(\mathfrak{G}) := \{(g, \kappa(v), \mathfrak{c}) \mid (v, \mathfrak{c}) \in \sigma(V) \text{ and } g \in \rho(v)\},$$
$$C_k(\mathfrak{G}) := \{(\vec{g}, \kappa(e), \mathfrak{c}) \mid (e, \mathfrak{c}) \in \sigma(w) \text{ with } w \in dom(\sigma) \text{ and } \vec{g} \in \rho(e)\}$$

Obviously, the realization ρ of $\mathfrak{G}$ and the conceptual content $C(\mathfrak{G})$ determine each other because $g \in (Int(\kappa(v)) \times Mod(\mathfrak{c}))^{(1)}$ is equivalent to $(g, \kappa(v), \mathfrak{c}) \in C_0(\mathfrak{G})$ and, for $|e| = k$, $\vec{g} \in (Int(\kappa(e)) \times Mod(\mathfrak{c}))^{(1)}$ is equivalent to $(\vec{g}, \kappa(e), \mathfrak{c}) \in C_k(\mathfrak{G})$.

Now, we introduce the quasi-order $\lesssim$ of *generalization* on $\Gamma(\vec{\mathbb{K}})$ as follows: For two triadic concept graphs $\mathfrak{G}_1 := (V_1, E_1, \nu_1, C_{\vec{\mathbb{K}}}, \kappa_1, \sigma_1, \rho_1)$ and $\mathfrak{G}_2 := (V_2, E_2, \nu_2, C_{\vec{\mathbb{K}}}, \kappa_2, \sigma_2, \rho_2)$, we define $\mathfrak{G}_1 \lesssim \mathfrak{G}_2$ if, for all $(g, \kappa_2(v), \mathfrak{c}) \in C_0(\mathfrak{G}_2)$, there exist $(g, v_s, \mathfrak{c}) \in C_0(\mathfrak{G}_1)$ $(s \in S)$ with

$$\nabla_{23}(\{\kappa_1(v_s) \mid s \in S\}, \mathfrak{c}) \lesssim_1 \nabla_{23}(\kappa_2(v), \mathfrak{c}),$$

and if, for all $((g_1, \ldots, g_k), \kappa_2(e), \mathfrak{c}) \in C_k(\mathfrak{G}_2)$, there exist $((g_1, \ldots, g_k), \kappa_1(e_t), \mathfrak{c}) \in C_0(\mathfrak{G}_1)$ $(t \in T)$ with

$$\nabla_{23}(\{\kappa_1(e_t) \mid t \in T\}, \mathfrak{c}) \lesssim_1 \nabla_{23}(\kappa_2(e), \mathfrak{c});$$

if $\mathfrak{G}_1 \lesssim \mathfrak{G}_2$ we say that $\mathfrak{G}_2$ is *more general* than $\mathfrak{G}_1$. The triadic concept graphs $\mathfrak{G}_1$ and $\mathfrak{G}_2$ of $\vec{\mathbb{K}}$ are said to be *conceptually equivalent* (in symbols: $\mathfrak{G}_1 \sim \mathfrak{G}_2$) if $\mathfrak{G}_1 \lesssim \mathfrak{G}_2$ and $\mathfrak{G}_2 \lesssim \mathfrak{G}_1$. The class of all triadic concept graphs of $\vec{\mathbb{K}}$ which are conceptually equivalent to a given triadic concept graph $\mathfrak{G}$ is denoted by $\widetilde{\mathfrak{G}}$. The set of all equivalence classes $\widetilde{\mathfrak{G}}$ of triadic concept graphs $\mathfrak{G}$ of $\vec{\mathbb{K}}$ together with the order $\leq$ induced by the quasi-order $\lesssim$ is an ordered set denoted by $\widetilde{\Gamma}(\vec{\mathbb{K}})$.

The definition of the generalization order for the triadic concept graphs of a triadic power context family $\vec{\mathbb{K}}$ uses as "background knowledge" only the trilattice structure of $\mathfrak{T}(\vec{\mathbb{K}})$ but not the contextual relationships of $\vec{\mathbb{K}}$. This is the reason that the operation ∇_{23} is only applied to pairs of the form $(\{\kappa_1(v_s) \mid s \in S\}, \mathfrak{c})$ with one element on the second place, because the necessary knowledge of $\mathfrak{c} \in G_0 \cap \mathfrak{T}(\mathbb{K}_0)$ may only be deduced from the image of σ_2, but not from $\vec{\mathbb{K}}$. The restriction of the "background knowledge" to the lattice structure of the concepts was also basic for the treatment of the dyadic case in [PW99]. Notice that the triadic approach also covers the dyadic case: If $B := \{b\}$ then the triadic concepts (A_1, A_2, A_3) of a triadic context (G, M, B, Y) are in ono-to-one correspondence to the dyadic concepts (A_1, A_2) of the dyadic context (G, M, I) with $gIm :\Leftrightarrow (g, m, b) \in Y$. By this correspondence, $\nabla_{23}(\{\kappa_1(v_s) \mid s \in S\}, \mathfrak{c}) \lesssim_1 \nabla_{23}(\kappa(v_2), \mathfrak{c})$ translates to $\bigwedge\{\kappa_1(v_s) \mid s \in S\} \leq \kappa_2(v)$ (an adapted version of this inequality should replace the missprinted inequality in [PW99] on page 407, line 3 from below).

There might be good reasons to assume as "background knowledge" the full contextual and conceptual structure of the triadic power context family $\vec{\mathbb{K}}$. This

may lead to a modified generalization order $\precsim$ in the following way: For a triadic concept graph $\mathfrak{G} := (V, E, \nu, C_{\vec{\mathbb{K}}}, \kappa, \sigma, \rho)$ of the triadic power context family $\vec{\mathbb{K}}$, we define the *contextual content* $Y(\mathfrak{G}) := (Y_0(\mathfrak{G}), \ldots, Y_n(\mathfrak{G}))$ by

$$Y_0(\mathfrak{G}) := \bigcup(\rho(v) \times Int(\kappa(v)) \times Mod(\mathfrak{c}) \mid (v, \mathfrak{c}) \in \sigma(w) \text{ with } w \in dom(\sigma)),$$
$$Y_k(\mathfrak{G}) := \bigcup(\rho(e) \times Int(\kappa(e)) \times Mod(\mathfrak{c}) \mid (e, \mathfrak{c}) \in \sigma(w) \text{ with } w \in dom(\sigma)).$$

Based on the notion of a contextual content, we propose a quasi-order $\precsim$ on $\Gamma(\vec{\mathbb{K}})$ by defining

$$\mathfrak{G}_1 \precsim \mathfrak{G}_2 :\Leftrightarrow Y_k(\mathfrak{G}_2) \subseteq Y_k(\mathfrak{G}_1) \text{ for } k = 0, \ldots, n.$$

$\mathfrak{G}_1$ and $\mathfrak{G}_2$ of $\vec{\mathbb{K}}$ are said to be *contextually equivalent* (in symbols: $\mathfrak{G}_1 \approx \mathfrak{G}_2$) if $\mathfrak{G}_1 \precsim \mathfrak{G}_2$ and $\mathfrak{G}_2 \precsim \mathfrak{G}_1$. The class of all triadic concept graphs of $\vec{\mathbb{K}}$ which are contextually equivalent to a given triadic concept graph $\mathfrak{G}$ is denoted by $\widehat{\mathfrak{G}}$. The set of all equivalence classes $\widehat{\mathfrak{G}}$ of triadic concept graphs $\mathfrak{G}$ of $\vec{\mathbb{K}}$ together with the order $\preceq$ induced by the quasi-order $\precsim$ is an ordered set denoted by $\widehat{\Gamma}(\vec{\mathbb{K}})$.

3 Lattices of Triadic Concept Graphs

For studying relationships between the triadic concept graphs of a triadic power concept family $\vec{\mathbb{K}}$, it is valuable to understand the structure of the ordered set $\widetilde{\Gamma}(\vec{\mathbb{K}})$. Supprisingly, $\widetilde{\Gamma}(\vec{\mathbb{K}})$ is always a complete lattice which is isomorphic to a complete subdirect product of specific sublattices of the dyadic concept lattices $\underline{\mathfrak{B}}((\mathbb{K}_k)_{12}^{Mod(\mathfrak{c})})$ ($k = 0, \ldots, n$ and $\mathfrak{c} \in G_0 \cap \mathfrak{T}(\mathbb{K}_0)$), extended by a new top element as Proposition 1 states (in general, we assume that $G_0 \cap \mathfrak{T}(\mathbb{K}_0) \neq \emptyset$).

Proposition 1 *Let $\vec{\mathbb{K}} := (\mathbb{K}_0, \ldots, \mathbb{K}_n)$ be a triadic power context family with $\mathbb{K}_k := (G_k, M_k, B, Y_k)$ for $k = 0, \ldots, n$; furthermore, for each $\vec{g} \in G_k$ and $\mathfrak{c} \in G_0 \cap \mathfrak{T}(\mathbb{K}_0)$, let $L_k^{(\vec{g},\mathfrak{c})}$ be the ordered set $(\mathfrak{I}_{\mathbb{K}_k}^{(\vec{g},\mathfrak{c})}, \lesssim_1)$ with*

$$\mathfrak{I}_{\mathbb{K}_k}^{(\vec{g},\mathfrak{c})} := \{\mathfrak{b}_{13}(A, Mod(\mathfrak{c})) \mid \vec{g} \in A \subseteq G_k\}$$

together with a new top-element $\top_k^{(\vec{g},\mathfrak{c})}$. Then $\widetilde{\Gamma}(\vec{\mathbb{K}})$ is isomorphic to the complete subdirect product of the complete lattices $L_k^{(\vec{g},\mathfrak{c})}$ ($k \in \{0, \ldots, n\}$, $\vec{g} \in G_k$, $\mathfrak{c} \in G_0 \cap \mathfrak{T}(\mathbb{K}_0)$) consisting of all elements $\vec{\mathfrak{a}} := (\mathfrak{a}_k^{(\vec{g},\mathfrak{c})})_{k=0,\ldots,n}^{g \in G_k}$ of the direct product satisfying the following condition:

$(\star)$ *If $\mathfrak{a}_k^{(\vec{g},\mathfrak{c})} \neq \top_k^{(\vec{g},\mathfrak{c})}$ and $\vec{g} = (\vec{g}_1, \ldots, g_k)$ then $\mathfrak{a}_0^{g_i} \neq \top_0^{g_i}$ for $i = 1, \ldots, k$.*

Proof: The map assigning the triadic concepts $(A_1, A_2, A_3) := \mathfrak{b}_{13}(A, Mod(\mathfrak{c}))$ of $(\mathfrak{I}_{\mathbb{K}_k}^{(\vec{g},\mathfrak{c})}, \lesssim_1)$ to the dyadic concepts (A_1, A_2) of $\underline{\mathfrak{B}}((\mathbb{K}_k)_{12}^{Mod(\mathfrak{c})})$ is an isomorhism between ordered sets; hence $(\mathfrak{I}_{\mathbb{K}_k}^{(\vec{g},\mathfrak{c})}, \lesssim_1)$ is a complete lattice and therefore $L_k^{(\vec{g},\mathfrak{c})}$ too.

Now, let $\mathfrak{G} := (V, E, \nu, C_{\vec{\mathbb{K}}}, \kappa, \sigma, \rho)$ be a triadic concept graph of $\vec{\mathbb{K}}$. For $\vec{g} \in G_k$ and $\mathfrak{c} \in G_0 \cap \mathfrak{T}(\mathbb{K}_0)$, let $\mathfrak{a}_k^{(\vec{g},\mathfrak{c})}(\mathfrak{G})$ be the element $\top_k^{(\vec{g},\mathfrak{c})}$ if there is no $u \in V \cup E$ with $(u, \mathfrak{c}) \in \sigma(V)$ and $\vec{g} \in \rho(u)$, and let otherwise

$$\mathfrak{a}_k^{(\vec{g},\mathfrak{c})}(\mathfrak{G}) := \nabla_{23}(\{\kappa(u) \mid u \in V \cup E, (u, \mathfrak{c}) \in \sigma(V), \vec{g} \in \rho(u)\}, \mathfrak{c}) \in \mathfrak{I}_{\mathbb{K}_k}^{(\vec{g},\mathfrak{c})}.$$

In the second case, we construct the "atomic" triadic concept graph consisting only of $w^\top$ and $\hat{u}$ with $\kappa(w^\top) := \mathfrak{b}_{13}(G_k, Mod(\mathfrak{c}))$, $\kappa(\hat{u}) := \mathfrak{a}_k^{(\vec{g},\mathfrak{c})}(\mathfrak{G})$, $\sigma(w^\top) := \{(\hat{u}, \mathfrak{c})\}$, $\rho(w^\top) := \{\mathfrak{c}\}$, $\rho(\hat{u}) := \{\vec{g}\}$ and, if $\vec{g} = (g_1, \ldots, g_k)$, also of $\hat{v}_1, \ldots, \hat{v}_k$ with $\nu(\hat{u}) := (\hat{v}_1, \ldots, \hat{v}_k)$, $\kappa(\hat{v}_i) := \mathfrak{b}_{13}(G_k, Mod(\mathfrak{c}))$, $\sigma(w^\top) := \{(\hat{v}_1, \mathfrak{c}), \ldots, (\hat{v}_k, \mathfrak{c})\}$, and $\rho(\hat{v}_i) := \{g_i\}$ for $i = 1, \ldots, k$. The disjoint union of all such atomic triadic concept graphs, having an object $\vec{g}$ as reference, is conceptually equivalent to the given concept graph $\mathfrak{G}$.

Furthermore, for each $(\vec{g}, \kappa(u), \mathfrak{c}) \in C_k(\mathfrak{G})$, we can construct an atomic triadic concept graph $\mathfrak{A}(\mathfrak{a}_k^{(\vec{g},\mathfrak{c})}(\mathfrak{G}))$ as follows: If $k = 0$ and $\vec{g} = g \in G_0$, we define

$$\mathfrak{A}(\mathfrak{a}_0^{(g,\mathfrak{c})}(\mathfrak{G})) := (\{w^\top, v\}, \emptyset, \emptyset, \kappa_0^{(g,\mathfrak{c})}, \sigma_0^{(g,\mathfrak{c})}, \rho_0^{(g,\mathfrak{c})})$$

($v \in V$) with $\kappa_0^{(g,\mathfrak{c})}(w^\top) := \mathfrak{b}_{13}(G_k, Mod(\mathfrak{c}))$, $\kappa_0^{(g,\mathfrak{c})}(v) := \mathfrak{a}_0^{(g,\mathfrak{c})}(\mathfrak{G})$, $\sigma_0^{(g,\mathfrak{c})}(w^\top) := \{(v, \mathfrak{c})\}$, $\rho(w^\top) := \{\mathfrak{c}\}$, and $\rho_0^{(g,\mathfrak{c})}(v) := \{g\}$; if $k \geq 1$ and $\vec{g} = (g_1, \ldots, g_k) \in G_k$, we define

$$\mathfrak{A}(\mathfrak{c}_k^{(\vec{g},\mathfrak{c})}(\mathfrak{G})) := (\{w^\top, v_1, \ldots, v_k\}, \{e\}, \nu_k^{(\vec{g},\mathfrak{c})}, \kappa_k^{(\vec{g},\mathfrak{c})}, \sigma_k^{(\vec{g},\mathfrak{c})}, \rho_k^{(\vec{g},\mathfrak{c})})$$

($v_1, \ldots, v_k \in V$ and $e \in E$) with $\nu_k^{(\vec{g},\mathfrak{c})}(e) := (v_1, \ldots, v_k)$, $\kappa_k^{(\vec{g},\mathfrak{c})}(e) := \mathfrak{a}_k^{(\vec{g},\mathfrak{c})}(\mathfrak{G})$, $\kappa(w^\top) := \mathfrak{b}_{13}(G_k, Mod(\mathfrak{c}))$, $\kappa_k^{(\vec{g},\mathfrak{c})}(v_i) := \mathfrak{b}_{13}(G_k, Mod(\mathfrak{c}))$, $\sigma_k^{(\vec{g},\mathfrak{c})}(w^\top) := \{(v_1, \mathfrak{c}), \ldots, (v_k, \mathfrak{c})\}$, $\rho(w^\top) := \{\mathfrak{c}\}$, and $\rho_k^{(\vec{g},\mathfrak{c})}(v_i) = \{g_i\}$ for $i = 1, \ldots, k$.

Now, the desired isomorhism can be defined as follows (cf. [PW99]):

$$\begin{aligned} \eta^{\vec{\mathbb{K}}}\colon\ \widetilde{\Gamma}(\vec{\mathbb{K}}) \ &\to\ \prod (L_k^{(\vec{g},\mathfrak{c})} \mid k \in \{0, \ldots, n\} \text{ and } \vec{g} \in G_k) \quad \text{with} \\ \eta^{\vec{\mathbb{K}}}(\widetilde{\mathfrak{G}}) \ &:=\ (\mathfrak{a}_k^{(\vec{g},\mathfrak{c})}(\mathfrak{G}) \mid k \in \{0, \ldots, n\} \text{ and } \vec{g} \in G_k) \end{aligned}$$

is a mapping, the image of which consists of all elements of the direct product satisfying condition ($\star$). It can be easily seen that this image is a complete subdirect product. For concept graphs $\mathfrak{G}_1$ and $\mathfrak{G}_2$, we have the equivalences

$$\begin{aligned} \mathfrak{G}_1 \lesssim \mathfrak{G}_2 \quad &\Leftrightarrow \quad \forall k \in \{0, \ldots, n\}\ \forall \vec{g} \in G_k\colon\ \mathfrak{a}_k^{(\vec{g},\mathfrak{c})}(\mathfrak{G}_1) \leq \mathfrak{a}_k^{(\vec{g},\mathfrak{c})}(\mathfrak{G}_2) \\ &\Leftrightarrow \quad \eta^{\vec{\mathbb{K}}}(\widetilde{\mathfrak{G}}_1) \leq \eta^{\vec{\mathbb{K}}}(\widetilde{\mathfrak{G}}_2). \end{aligned}$$

Hence $\eta^{\vec{\mathbb{K}}}$ is an injective homomorphism from $\widetilde{\Gamma}(\vec{\mathbb{K}})$ into $\prod(L_k^{(\vec{g},\mathfrak{c})} \mid 0 \leq k \leq n,$ $\vec{g} \in G_k,$ and $\mathfrak{c} \in G_0 \cap \mathfrak{T}(\mathbb{K}_0))$. Thus, the assertion of the proposition is proved. □

Proposition 1 yields, for the triadic concept graphs of a triadic power context family, a system of representatives for their equivalence classes which makes

the generalization order transparent. These representatives are described by the elements of all the direct products

$$\prod_{(k,\vec{g},\mathfrak{c})\in U} \mathfrak{I}_{\mathbb{K}_k}^{(\vec{g},\mathfrak{c})}$$

for which U is a finite subset of $\bigcup_{k=0,\ldots,n}\{k\}\times G_k\times(G_0\cap\mathfrak{T}(\mathbb{K}_0))$ satisfying the implication

$$(k,\vec{g},\mathfrak{c})\in U \text{ and } g=(g_1,\ldots,g_k)\Rightarrow(0,g_1,\mathfrak{c}),\ldots,(0,g_k,\mathfrak{c})\in U.$$

An element $\vec{\mathfrak{a}} := (\mathfrak{a}_k^{(\vec{g},\mathfrak{c})} \mid (k,\vec{g},\mathfrak{c})\in U)$ of the product represents the triadic concept graph which is the disjoint union of the atomic triadic concept graphs $\mathfrak{A}(\mathfrak{a}_k^{(\vec{g},\mathfrak{c})})$ with $(k,\vec{g},\mathfrak{c})\in U$. In the full direct product

$$\prod(L_k^{(\vec{g},\mathfrak{c})} \mid 0\le k\le n, \vec{g}\in G_k, \text{ and } \mathfrak{c}\in G_0\cap\mathfrak{T}(\mathbb{K}_0)),$$

this triadic concept graph is represented by the element which coincides with $\vec{\mathfrak{a}}$ on U and has outside U the corresponding top-elements as components (the top-element $\top_k^{(\vec{g},\mathfrak{c})}$ as a component indicates that $\vec{g}$ is not a reference in the represented graph).

Besides the already analyzed complete lattice $\widetilde{\Gamma}(\vec{\mathbb{K}})$, the ordered set $\widehat{\Gamma}(\vec{\mathbb{K}})$ of the classes of contextually equivalent triadic concept graphs also deserves an investigation to gain more insights into the relationships between triadic concept graphs of a triadic power context family $\vec{\mathbb{K}}$. We obtain that also this ordered set is a complete lattice:

Proposition 2 *Let $\vec{\mathbb{K}} := (\mathbb{K}_0,\ldots,\mathbb{K}_n)$ be a triadic power context family with $\mathbb{K}_k := (G_k, M_k, B, Y_k)$ for $k=0,\ldots,n$. Then $\widehat{\Gamma}(\vec{\mathbb{K}})$ is isomorhic to the complete lattice*

$$\prod_{k=0}^{n}\underline{\mathfrak{B}}(Y_k, G_k\times\mathfrak{T}(\mathbb{K}_k)\times(G_0\cap\mathfrak{T}(\mathbb{K}_0)), I_k)$$

with $(\vec{g},m,b)I_k(\vec{h},\mathfrak{b},\mathfrak{c}) :\Leftrightarrow \vec{g}\neq\vec{h}$ or $\vec{g}\notin(Int(\mathfrak{b})\times Mod(\mathfrak{c}))^{(1)}$ or $m\notin Int(\mathfrak{b})$ or $b\notin Mod(\mathfrak{c})$.

Proof: For a triadic concept graph $\mathfrak{G} := (V,E,\nu,C_{\vec{\mathbb{K}}},\kappa,\sigma,\rho)$ of $\vec{\mathbb{K}}$ we define $\iota(\mathfrak{G}) := (Y_k\setminus Y_k(\mathfrak{G}),(Y_k\setminus Y_k(\mathfrak{G}))^{I_k})$. Since $Y_k(\mathfrak{G}) = \bigcup(Y_k\setminus(\vec{h},\mathfrak{b},\mathfrak{c})^{I_k} \mid (u,\mathfrak{c})\in\sigma(w)$ with $w\in dom(\sigma),\mathfrak{b}=\kappa(u),$ and $\vec{h}\in G_k\setminus\rho(u))$, we have that $G_k\setminus Y_k(\mathfrak{G})$ is an extent and so $\iota(\mathfrak{G})$ a formal concept of the formal context $(Y_k, G_k\times\mathfrak{T}(\mathbb{K}_k)\times(G_0\cap\mathfrak{T}(\mathbb{K}_0)), I_k)$ for $k=0,\ldots,n$. Now, it is straightforward to prove that ι is even the desired isomorphism. □

The conceptual content and the contextual content are interesting notions for understanding which information might be given by triadic concept graphs. But, Figure 1 shows that there are even further interesting ideas of content, because the melodic character of the presented triadic concept graph has been grasped neither by its conceptual content nor by its contextual content.

Acknowledgement

This paper has substantially benefited from the financial support given by the Australian Research Council (IREX-Programme) which allowed the second author to work for two months at the School of Information Technology of the Griffith University (Gold Coast Campus).

References

Ac71. K. Acham: Abstraktion. Historisches Wörterbuch der Philosophie. Bd.1. Schwabe, Basel 1971, 59–63.

Bi97. K. Biedermann: How triadic diagrams represent conceptual structures. In: D. Lukose, H. Delugach, M. Keeler, L. Searle, J. F. Sowa (eds.): *Conceptual Structures: Fulfilling Peirce's Dream*. LNAI **1257**. Springer, Heidelberg 1997, 304–317.

Bi98. K. Biedermann: A foundation of the theory of trilattices. Dissertation, TU Darmstadt 1998. Shaker Verlag, Aachen 1998.

GW96. B. Ganter, R. Wille: *Formale Begriffsanalyse: Mathematische Grundlagen*. Springer, Heidelberg 1996; English translation: *Formal Concept Analysis: Mathematical Foundations*. Springer, Heidelberg 1999.

LW95. F. Lehmann, R. Wille: A triadic approach to formal concept analysis. In: G. Ellis, R. Levinson, W. Rich, J. F. Sowa (eds.): *Conceptual Structures: Applications, Implementation and Theory*. LNAI **954**. Springer, Heidelberg 1995, 32–43.

Pr98. S. Prediger: *Kontextuelle Urteilslogik mit Begriffsgraphen. Ein Beitrag zur Restrukturierung der mathematischen Logik*. Dissertation, TU Darmstadt. Shaker Verlag, Aachen 1998.

PW99. S. Prediger, R. Wille: The lattice of concept graphs of a relationally scaled context. In: W. Tepfenhart, W. Cyre (eds.): *Conceptual Structures: Standards and Practices*. LNAI **1640**. Springer, Heidelberg 1999, 401–414.

Wi97. R. Wille: Conceptual Graphs and Formal Concept Analysis. In: D. Lukose, H. Delugach, M. Keeler, L. Searle, J. F. Sowa (eds.): *Conceptual Structures: Fulfilling Peirce's Dream*. LNAI **1257**. Springer, Heidelberg 1997, 290–303.

Wi98. R. Wille: Triadic Concept Graphs. In: M.-L. Mugnier, M. Chein (eds.): *Conceptual Structures: Theory, Tools and Applications*. LNAI **1453**, Springer, Heidelberg 1998, 194–208.

WZ99. R. Wille, M. Zickwolff: Grundlagen einer Triadischen Begriffsanalyse. In: G. Stumme, R. Wille (eds.): *Begriffliche Wissensverarbeitung: Methoden und Anwendungen*. Springer, Heidelberg 2000, 125–150.

Formalizing Hypotheses with Concepts

Bernhard Ganter and Sergei O. Kuznetsov

[1] Technische Universität Dresden
ganter@math.tu-dresden.de
[2] Technische Universität Dresden
kuznetso@math.tu-dresden.de

Abstract. The frequently used strategy of forming hypotheses, based on observations, to generate predictions can be formalized in several ways. We study an elaborated approach that has some conceptual flavour and translate the basic ideas in the language of Formal Concept Analysis, blazing the trail for applications to other conceptual structures as well. We investigate the relation to pseudo intents, discuss algorithmic questions, and give an example.

Motivation

Imagine that on your way to work there is a junction where you frequently run into a traffic jam. You would like to know in advance whether you are likely to get in the hold up or not, because then you could take a detour to avoid a delay. But you known of no single obvious cause for the congestion. You know that certain constellations are very likely to cause such a problem, and you also know of situations where it is rather safe. Probably you will develop your personal hypotheses, based on your positive and negative experiences, to predict traffic jams at that junction.

A similarly structured problem, concerning pharmacological applications, is known as the Structure-Activity Relationship (SAR) Problem [14], where structural properties of chemical compounds (such as particular subgraphs of their molecular graphs) are used as predictors for certain biological activities (like mutagenicity, sedativity, or such).

The logical framework of such problems has been extensively studied. One of the broader approaches is called the *JSM-method* (after John Stuart Mill, an English philosopher of the 19th century, who was one of the first to systematically consider schemes of inductive reasoning). It was proposed by V. Finn [3] in 1983 and has been developed ever since by his group at the Moscow VINITI Institute. From the standpoint of Artificial Intelligence, that method can be considered as a formal system of inductive plausible reasoning based on examples and knowledge about the domain (see Finn [4]) or as a method of Machine Learning that employs examples and counterexamples of a goal attribute (compare [10]).

The original formalization of the JSM-method used by Finn was the language of first-order predicate calculus with two sorts of variables, six truth-value types

B. Ganter and G.W. Mineau (Eds.): ICCS 2000, LNAI 1867, pp. 342–356, 2000.
© Springer-Verlag Berlin Heidelberg 2000

and "quanitification over tuples with variable length" [1]. However, the underlying combinatorial structure of the JSM-method suggests analogies to another method of knowledge processing, that of the Formal Concept Analysis (FCA) [6]. Some attempts to connect these approaches were already made in [11], [12]. In this paper we propose a partial translation from the language of the JSM-theory into that of FCA.[1]

1 Hypotheses

We consider a finite set M of "structural attributes", a set G of objects (or observations) and a relation $I \subseteq G \times M$, such that $(g, m) \in I$ if and only if object g has the attribute m. Such a triple $\mathbb{K} = (G, M, I)$ is called a **formal context**. It is the basic data type of Formal Concept Analysis. Using the **derivation operators**, defined for $A \subseteq G$, $B \subseteq M$ by

$$A' := \{m \in M \mid gIm \text{ for all } g \in A\},$$
$$B' := \{g \in G \mid gIm \text{ for all } m \in B\},$$

we can define a **formal concept** (of the context $\mathbb{K}$) to be a pair (A, B) satisfying $A \subseteq G$, $B \subseteq M$, $A' = B$, and $B' = A$. A is called the **extent** and B is called the **intent** of the concept (A, B). These concepts, ordered by

$$(A_1, B_1) \geq (A_2, B_2) \iff A_1 \supseteq A_2$$

form a complete lattice, called **the concept lattice** of $\mathbb{K} := (G, M, I)$. Double application of the derivation operators, i.e., $(B')'$ or $(A')'$ are abbreviated as B'' or A'', repsectively. It can easily be shown that $''$ is a closure operator, i.e., it is monotone, idempotent, and extensive. Further on, this operator will be referred to as **closure** and sets $A \subseteq G$, $B \subseteq M$ such that $A'' = A$, $B'' = B$ as **closed**.

In what follows we shall need the notion of implication between attribute sets [6]. An **implication** $A \to B$ between a pair of subsets of the attribute set M **holds** for a given formal context $\mathbb{K} := (G, M, I)$ if every object that has all the attributes from A also has all attributes from B. This is equivalent to $A' \subseteq B'$.

In addition to the structural attributes of M we consider a **goal attribute** $w \notin M$. This divides the set G of all objects into three subsets: The set G_+ of those objects that are known to have the property w (these are the *positive examples*), the set G_- of those objects of which it is known that they do not have w (the *negative examples*) and the set G_τ of *undetermined examples*, i.e., of those objects, of which it is unknown if they have property w or not. This gives three subcontexts of $\mathbb{K} = (G, M, I)$:

$$\mathbb{K}_+ := (G_+, M, I_+), \mathbb{K}_- := (G_-, M, I_-), \text{ and } \mathbb{K}_\tau := (G_\tau, M, I_\tau),$$

where for $\varepsilon \in \{+, -, \tau\}$ we have $I_\epsilon := I \cap G_\varepsilon \times M$.

[1] In terms of the JSM-theory, we consider only "counterexample forbidding hypotheses" and the "atomistic case," where there is a single goal attribute

Intents are, as defined above, attribute combinations shared by some of the observed objects. In order to form hypotheses about structural causes of the goal attribute w, we are interested in sets of structural attributes that are shared by some positive, but by no negative examples. Thus, a **positive hypothesis**, or a (+)-hypothesis, for w is an intent of $\mathbb{K}_+$ that is not contained in the intent g' of any negative example $g \in G_-$. A **negative hypothesis**, or a (-)-hypothesis, is defined accordingly. If g is an object such that its intent[2]

$$g' := \{m \in M \mid (g, m) \in I\}$$

contains a positive or a negative hypothesis then we say, for short, that g contains that hypothesis.

We intend to use these hypotheses as predictors for the undetermined examples, predicting that an object $g \in G_\tau$ has the goal attribute if it contains a positive hypothesis and does not have w if it contains a negative one. These cases may be not exclusive. It may happen that an object contains both a positive and a negative hypothesis. An object that contains a positive, but no negative hypothesis will be *classified positively*. Negative classifications are defined similarly. If g' contains hypotheses of both kinds, or if g' contains no hypothesis at all, then the classification is contradictory or undetermined, respectively. The underlying principle can be formulated in the line of J.S.Mill as "common effects are brought about by common causes." We may restrict to *minimal* (w.r.t. inclusion $\subseteq$) hypotheses, positive as well as negative, since an object obviously contains a positive hypothesis if and only if it contains a minimal positive hypothesis, etc.

Of course there is, without further assumptions, little hope that these predictions will be correct. Realistically, we have no reliable information on the unknown set G_w of those objects in G which do have w, except for $G_+ \subseteq G_w$ and $G_- \cap G_w = \emptyset$. We have not even excluded the possibility that the goal does not depend at all on the structural attributes. It might happen that there are positive and negative examples with exactly the same intents. All we can guarantee is the following, trivial fact:

Proposition 1. *A positive example $g \in G_+$ contains a positive hypothesis if and only if g' is not contained in the intent of any negative example.*

Nevertheless it seems natural to operate with hypotheses, at least in situations where we expect that the goal attribute somehow depends on the structural attributes. An analogue of the following assumption on the (unknown) set G_w is called the **Spinoza-axiom** in [4]:

The goal attribute w is **properly implied** in (G, M, I) if there are sets $\mathcal{P}, \mathcal{N} \subseteq \mathcal{P}(M)$[3] such that

1. $g \in G_w \iff P \subseteq g'$ for some $P \in \mathcal{P}$,
2. $g \notin G_w \iff N \subseteq g'$ for some $N \in \mathcal{N}$.

[2] For brevity sake we write g' and g'' instead of $\{g\}'$ and $\{g\}''$, respectively.
[3] $\mathcal{P}(M)$ is the power set of M

Thus, the goal attribute is properly implied if there is a set $\mathcal{P}$ of *positive attribute combinations* that force the goal attribute w, and a set $\mathcal{N}$ of *negative attribute combinations* that exclude w, and if, moreover, these combinations would suffice to classify all objects in G.

However, this condition does not require that these positive or negative attribute combinations have been observed as such. Let us therefore assume that each P and each N is supported by some example, i.e., that

3. for each $P \in \mathcal{P}$ there is some $g \in G_+$ such that $P \subseteq g'$,
4. for each $N \in \mathcal{N}$ there is some $g \in G_-$ such that $N \subseteq g'$.

It seems that this condition is rather strong. It does however not suffice to make the hypotheses work, as the following example shows:

	a	b	c	d	e	f	$+, -, \tau$	w
g_1	×		×				$+$	yes
g_2		×	×				$+$	yes
g_3				×	×		$-$	no
g_4				×		×	$-$	no
g_5		×	×	×			τ	yes, but undetermined
g_6			×	×	×		τ	no, but undetermined

In this formal context, we have $\mathcal{P} := \{\{a, c\}, \{b, c\}\}$, $\mathcal{N} := \{\{d, e\}, \{d, f\}\}$. Each set in $\mathcal{P}$ is a positive hypothesis, each set in $\mathcal{N}$ is a negative hypothesis. But $\{c\}$ is also a positive hypothesis and $\{d\}$ is a negative one, which has the effect that each undetermined object contains both a positive and a negative hypothesis.

Note that in this example each $g \in G_w$ contains a positive hypothesis and every $g \notin G_w$ contains a negative one.

Another condition would be that whatever is not causal for the goal attribute will be observed to be non-causal. This may be formalized as follows:

5. If for a set $S \subseteq G_+$ there is no $P \in \mathcal{P}$ with $P \subseteq S'$, then there is some $h \in G_-$ such that $S' \subseteq h'$,
6. if for a set $T \subseteq G_-$ there is no $N \in \mathcal{N}$ with $N \subseteq T'$, then there is some $h \in G_+$ such that $T' \subseteq h'$.

Proposition 2. *If conditions 1), 2), 5), and 6) are satisfied, then every positive hypothesis contains some $P \in \mathcal{P}$ and every negative hypothesis contains some $N \in \mathcal{N}$.*

Then, in particular, any object containing a positive hypothesis must be in G_w and those containing a negative hypothesis must be in the complement of G_w. The converse is not necessarily true: it may happen that the observed hypotheses are too large and that, therefore, not every $g \in G_w$ contains a positive hypothesis in its intent. We may then try to improve our hypotheses by enlarging G_+ and G_- by those elements from G_τ that contain a positive or a negative hypothesis, respectively. Conditions 1), 2), 5), and 6) will remain satisfied, but the new configuration may lead to new hypotheses which are smaller then the ones before.

This process may be iterated. For such an iteration process suggested by Finn there is no guarantee that the conditions 1) to 6) are fulfilled. The dynamics of such an iteration process will not be studied here.[4]

2 Hypotheses and Implications

We have introduced (positive) hypotheses as attribute combinations that imply the goal attribute w, but with an additional "conceptual" condition: only intents of formal concepts are considered as hypotheses. We explain shortly why this is reasonable.

Suppose that C is some attribute combination occurring only in positive examples, so that we may expect that C implies w. But suppose that there are further attributes, say $d_1, \ldots, d_n$, that also "come with C" in the sense that each observed object having C also has the set of attributes $\{d_1, d_2, \ldots, d_n\}$. Then there are two typical strategies to generate a hypothesis:

> The *courageous* strategy would argue: "Since we have observed that w occurs whenever C does, we shall predict w when we observe C."
> The *cautious* strategy would say: "We have observed that in case of C we also have $d_1, \ldots, d_n$ and w, so we shall predict w if we meet the same conditions, i.e., $C \cup \{d_1, \ldots, d_n\}$. If you think that the set of attributes $\{d_1, \ldots, d_n\}$ is superfluous, give an example in which, given C, the set of conditions $\{d_1, \ldots, d_n\}$ does not hold, but w does."

The JSM-approach follows the cautious strategy. It is assumed that hypotheses are closed under implications to other structural attributes.

However, the subsets closed under implications are precisely the intents of $\mathbb{K}$. Therefore, it is sensible to allow only intents (of concepts of $\mathbb{K}$) as hypotheses.

There is a powerful tool for studying implications in formal contexts: the notion of a **pseudo intent**. The definition is recursive[5]. A **pseudo intent** of a formal context (G, M, I) is a set $P \subseteq M$ satisfying

1. $P \neq P''$, and
2. for every pseudo intent $Q \subseteq P$, $Q \neq P$, we have $Q'' \subseteq P$.

Pseudointents can be used to give an irredundant representation of the implicational theory of a formal context, see [6] for details. We show that they are closely related to the minimal hypotheses:

Let $\mathbb{K}$, $\mathbb{K}_+$ and $\mathbb{K}_-$ be as above and let

$$\mathbb{K}_\pm := (G_+ \cup G_-, M \cup \{w\}, I_+ \cup I_- \cup G_+ \times \{w\})$$

be the context of the positive and the negative examples, extended by the goal attribute w in the natural way. The derivation and closure operators of this

[4] For details, see [1], [3], [4].

[5] ... and a little confusing, because there seems to be no base case. But note that the empty set automatically fulfills the second condition, since it contains no proper subsets at all.

context will be denoted by superscripts $^{\pm}$ and $^{\pm\pm}$, respectively. In particular, for a given set $P \subseteq M$ of attributes we write $P^{\pm\pm}$ to denote the smallest intent of $\mathbb{K}_{\pm}$ that contains P. Derivation and closure operators for contexts $\mathbb{K}_+$ and $\mathbb{K}_-$ will be denoted by superscripts $^+$, $^-$ and $^{++}$, $^{--}$, respectively.

Proposition 3. *Let H be some minimal positive hypothesis for w (so in particular $w \notin H$). Then*

1. *if $P \subseteq H$ is a pseudo intent with $w \in P^{\pm\pm}$ then $P^{\pm\pm} = H \cup \{w\}$,*
2. *there is some pseudo intent P of $\mathbb{K}_{\pm}$ such that $P \subseteq H$ and $P^{\pm\pm} = H \cup \{w\}$.*
3. *If P is a pseudo intent such that $w \notin P$ but $w \in P^{\pm\pm}$, then $P^{\pm\pm} \setminus \{w\}$ is a positive hypothesis*[6]*.*

Proof. Suppose that H contains a pseudo intent P with $w \in P^{\pm\pm}$. Then, since H is an intent of $\mathbb{K}_+$, we get $P^{++} \subseteq H^{++} = H$, thus P^{++} is a hypothesis for w. The minimality of H implies $H = P^{++}$, which proves 1).

Since $H^{\pm\pm} = H \cup \{w\}$, there must be some pseudo intent $P \subseteq H$ with $w \in P^{\pm\pm}$. This gives 2).

The closure of P is $P^{\pm\pm}$ and thus $P^{\pm\pm} \setminus \{w\} = P^{++}$ is closed in $\mathbb{K}_+$ and is contained in no negative example, since otherwise $w \notin P^{\pm\pm}$. Therefore this set is a hypothesis.

In what follows, a pseudo intent P such that $w \notin P$ and $w \in P^{\pm\pm}$ will be called (w-)*generative*.

3 An Example

To illustrate the notion of a minimal hypothesis and the relation of hypotheses with pseudointent-based implications we consider an expert analysis of 17 winter wheel chains (that improve behavior of a car on winter roads). Information was given by the table below from the ADAC Magazine (1999, no. 11).

The values of the **system** attribute give the type of a chain system: SK - rope chain (Seilkette), SRK - steel ring chain (Stahlringkette), SMS - quick mounting chain (Schnellmontage-System).

The **mount** attribute takes the values F and F or R to denote that a chain of particular type can be mounted either only on the front wheels or both on the front and rear wheels. The values of **price** are given in DM, the values of **con** give the average expert asessment of the conveniency of a particular type of chain; the values of **snow** give average expert asessments of the maneuverability of a car, with a particular kind of chain, on snow; **ice** means the same for ice; the values of **dur** give average expert asessments of the durability of a particular kind of chain; the values of **grade** give average expert asessments of the general quality of a particular chain type. Smaller values of attributes **con**, **snow**, **ice**, **dur**, and **grade** correspond to better asessments of the corresponding chain properties.

[6] not necessarily a minimal one

type	system	mount	price	con	snow	ice	dur	grade
1	SK	F	206	1.9	1.4	1.8	2.7	1.8
2	SRK	F or R	520	2.1	0.8	3.8	2.3	1.9
3	SK	F	160	1.7	1.9	1.6	3.7	2.1
4	SK	F	213	1.7	2.0	2.4	3.4	2.1
5	SMS	F or R	598	1.6	2.4	2.7	2.8	2.2
6	SK	F	109	2.0	1.9	2.4	3.7	2.3
7	SRK	F or R	325	2.0	2.1	3.2	2.8	2.3
8	SMS	F or R	498	1.5	3.3	3.5	2.0	2.4
9	SRK	F or R	396	2.8	2.1	3.1	2.5	2.6
10	SRK	F or R	325	2.2	2.2	4.6	3.2	2.6
11	SRK	F or R	389	2.0	2.2	3.3	4.3	2.6
12	SRK	F	298	2.5	2.3	3.3	2.8	2.6
13	SK	F	149	1.9	2.5	4.0	3.8	2.6
14	SMS	F or R	684	1.7	3.3	4.4	2.2	2.6
15	SK	F	99	2.8	2.2	2.5	4.0	2.7
16	SK	F	140	2.6	2.3	3.3	3.4	2.7
17	SK	F	215	2.3	3.8	4.8	2.3	3.1

Here the values of the **type** attribute substitute tradenames.

As goal attributes we considered the **grade** (obtained as an average expert asessment of quality) and the **price**. Since the information was given in numerical values, we had to scale it before obtaining hypotheses and pseudointents. Scalings in both cases were similar, except for the goal attributes.

3.1 Grade

First, we made an assumption that the values of **grade** less or equal to 2.1 testify to the high quality of an item and the values of **grade** greater or equal to 2.6 testify to the low quality of an item. Thus, items 1-4 were treated as positive and items 9-17 were treated as negative examples, respectively. The items with numbers 5-8 were neglected as those with ambiguous medium-value grades. Thus, the positive and negative contexts w.r.t. the goal attribute **grade** are given as follows (the values of the **grade** attribute are given in brackets to indicate that this is the goal attribute and its actual values are insignificant within a context, either positive or negative):

Positive context

type	system	mount	price	con	snow	ice	dur	(grade)
1	SK	F	206	1.9	1.4	1.8	2.7	(1.8)
2	SRK	F or R	520	2.1	0.8	3.8	2.3	(1.9)
3	SK	F	160	1.7	1.9	1.6	3.7	(2.1)
4	SK	F	213	1.7	2.0	2.4	3.4	(2.1)

Negative context

type	**system**	**mount**	**price**	**con**	**snow**	**ice**	**dur**	(**grade**)
9	SRK	F or R	396	2.8	2.1	3.1	2.5	(2.6)
10	SRK	F or R	325	2.2	2.2	4.6	3.2	(2.6)
11	SRK	F or R	389	2.0	2.2	3.3	4.3	(2.6)
12	SRK	F	298	2.5	2.3	3.3	2.8	(2.6)
13	SK	F	149	1.9	2.5	4.0	3.8	(2.6)
14	SMS	F or R	684	1.7	3.3	4.4	2.2	(2.6)
15	SK	F	99	2.8	2.2	2.5	4.0	(2.7)
16	SK	F	140	2.6	2.3	3.3	3.4	(2.7)
17	SK	F	215	2.3	3.8	4.8	2.3	(3.1)

The present example is not yet fully compatible with our definitions from Section 1. To obtain a formal context, we apply a conceptual scaling (see [6] for details) replacing a given many-valued attribute by one-valued ones. We only list the scale attributes. The first two scales are nominal, the other are ordinal.

system	SK	SRK	SMS	
mount	F	F or R		
price	≤ 160	≤ 215	≤ 520	> 520
con	≤ 2.1	≤ 2.5	> 2.5	
snow	≤ 2.0	> 2.0		
ice	≤ 2.4	≤ 3.0	≤ 4.0	> 4.0
dur	≤ 3	≤ 3.7	> 3.7	

The table is read as follows. Original many-valued attributes are listed in the first column. Each many-valued attribute staying in the beginning of the row is replaced by several Boolean attributes that stay in other row positions. For example, the many-valued attribute **system** is replaced by attributes SK, SRK, and SMS, so that each object gets exactly one of them. The attribute **mount** is scaled similarly. This type of scaling is called *nominal* in [6]. The many-valued attribute **price** is replaced by four Boolean attributes ≤ 160, ≤ 215, ≤ 520, and > 520. In contrast to the nominal attributes **system** and **mount**, the objects that have the attribute ≤ 160, have also attributes ≤ 215 and ≤ 520 (in what follows, for brevity sake, we do not write this explicitly in the descriptions of object intents); the objects that have the attribute ≤ 215 have also the attribute ≤ 520. The other numerical attributes are scaled similarly. This type of scaling is called *ordinal* in [6].

The following positive and negative minimal hypotheses were obtained (they are unique, since the intersections of all positive and of all negative example intents are nonempty):

- the minimal positive hypothesis: {**con** ≤ 2.1, **snow** ≤ 2.0, **ice** ≤ 4, **dur** ≤ 3.7};
- the minimal negative hypothesis: {**snow** > 2.0}.

The generative (w.r.t. the **grade** attribute) pseudointents of $K_{\pm}$ are {**snow** ≤ 2.0} and {**con** ≤ 2.1, **ice** ≤ 4, **dur** ≤ 3.7}.

The first one {**snow** ≤ 2.0} is obviously a pseudointent of $\mathbb{K}_{\pm}$, since it is a nonclosed one-element subset of M. The second one is a pseudointent of $\mathbb{K}_{\pm}$, since it is not closed w.r.t. $\pm$ and all its subsets are closed.

If we introduce the "antigoal attribute" as a new attribute (in this case it corresponds to the "low quality of an item"), then the corresponding generative pseudointent will be {**snow** > 2.0}.

One can see that the positive minimal hypothesis is the closure (in the positive context with examples 1-4 and all attributes except for the general grade) of the both generative pseudointents (compare with Proposition 3). The implications

$$\{\textbf{snow} \leq 2.0\} \ \rightarrow \ \{\text{good quality}\},$$
$$\{\textbf{con} \ \leq 2.1, \ \textbf{ice} \ \leq 4, \ \textbf{dur} \ \leq 3.7\} \ \rightarrow \ \{\text{good quality}\},$$

which correspond to the generative pseudointents, coincide here with the minimal conditions for good quality. They can be used, e.g., by a producer who wants to attain the best sales at the lowest cost. For example, it suffices for a chain to make a car behave good on snow to make customer consider it as a good one. The implication

$$\{\textbf{con} \leq 2.1, \ \textbf{snow} \leq 2.0, \ \textbf{ice} \leq 4, \ \textbf{dur} \leq 3.7\} \ \rightarrow \ \{\text{good quality}\},$$

which corresponds to the minimal positive hypothesis, informs one about the whole bunch of attributes relative to the notion "good chain." This can be interpreted as a viewpoint of a customer who wants to know what is really a good chain. A good chain should be convenient, behave excellently on snow and at least satisfactorily on ice, and its life time should not be very small.

Both viewpoints are justified at their own and the consideration of both provides one with multifacetous understanding of the situation under study.

Note that for the above consideration of causes of good quality it is also reasonable to consider nonminimal hypotheses. For example, the (+)-hypothesis

$$\{\text{SK, F}, \ \textbf{price} \leq 215, \ \textbf{con} \leq 2.1, \ \textbf{snow} \leq 2.0, \ \textbf{ice} \leq 4.0, \ \textbf{dur} \leq 3.7\}$$

describes a class of relatively cheap chains, which have the same system and same mounting possibilities with good behavior on snow and satisfactory behavior on ice, that have good asessment of quality. One can also indicate (-)-hypotheses

$$\{\text{SRK, F or R}, \ \textbf{price} \leq 520, \ \textbf{snow} > 2.0\};$$
$$\{\text{SK, F}, \ \textbf{price} \leq 215, \ \textbf{snow} > 2.0\},$$

which describe different classes of chains with only low quality. These classes use different systems, have different mounting possibilities and their prices range within different intervals.

Nonminimal hypotheses can give information about taxonomy of positive and negative examples, which may be useful for the understanding of real causes of

the goal attribute or its absence. If a customer identifies the quality of a good with some construction and utilization specificity, then these hypotheses can allow him to form simple consumer heuristics like the following one: "the quality of SRK chains with front or rear mounting is not high, but the quality of SK chains with front mounting can be different depending on other parameters."

3.2 Price

In the case of this goal attribute we took the items with costs less or equal to DM 215 to be cheap (positive examples, items 1, 3, 4, 6, 13, 15, 16, 17) and the items with costs greater or equal to DM 430 to be expensive (negative examples, items 2, 5, 8, 14). The items with numbers 7, 9, 10, 11, 12 were neglected as those with ambiguous medium-value prices. The **grade** attribute is scaled to obtain the following three ordinal attributes **grade** ≤ 2.1, **grade** ≤ 2.5, **grade** > 2.5. Other attributes were scaled in the same way as in the case of the **grade** goal attribute.

The following positive and negative minimal hypotheses were obtained (they are unique, since the intersections of all positive and of all negative example intents are nonempty):

- the minimal positive hypothesis: {SK, F}
- the minimal negative hypothesis: {F or R, **con** ≤ 2.1, **dur** ≤ 3.0}.

There are two generative (w.r.t. the "cheap" antigoal attribute) pseudointents: {SK} and {F}.

If we introduce the "antigoal attribute" as a new attribute (in this case it corresponds to the "expensiveness"), then the corresponding three generative pseudointents are {F or R}, {**con** ≤ 2.1}, and {**dur** ≤ 3.0}.

As in the case with the **grade** goal attribute, each pseudointent generative w.r.t. the goal attribute **price** gives small sufficient conditions for the occurrence of the goal attribute, but minimal hypotheses give more detailed description of what does a "cheap chain" mean and what does an "expensive chain" mean.

The consideration of the hypotheses and pseudointent-based implications shows that the quality of chains (asessed by experts) is fairly independent of their price.

Among nonminimal hypotheses that may be of interest here we can indicate the negative hypothesis

$$\{\text{SMS, F or R, } \mathbf{con} \leq 2.1, \mathbf{snow} \geq 2.0, \mathbf{dur} \leq 3.0\},$$

which describes a class of expensive chains with a certain system, certain mounting possibilities, convenient, but with bad behavior on snow.

4 Algorithmic Problems

As it was shown in [5], the set of all formal concepts of a formal context can be generated in time $O(|G|^2 \cdot |M| \cdot |\underline{\mathfrak{B}}(K)|)$, where $|\underline{\mathfrak{B}}(K)|$ is the size of the concept

lattice of the context K, by an algorithm with polynomial delay. Recall that an algorithm for listing a family of combinatorial structures is said to have *delay d* [9] if and only if it satisfies the following conditions whenever it is run with any input of length p:

1. It executes at most $d(p)$ computation steps before either outputting the first structure or halting.

2. After any output it executes at most $d(p)$ machine instructions before either outputting the next structure or halting. An algorithm whose delay is bounded from above by a polynomial in the length of the input is called a *polynomial delay* algorithm[9].

In [7] the algorithm **Next Concept** from [5] was extended for the construction of the concept lattice for a given context (also with polynomial time delay). Algorithms for generating hypotheses can be found in [13]. However, to generate the set of all hypotheses, one can also adapt the **Next Concept** algorithm, running it in the bottom-up order (from least extents to least intents) in the following way. To obtain all hypotheses one should repeatedly call the following procedure, where $\mathcal{H}_+$ denotes the current set of positive hypotheses.

Next Hypothesis

0. A: = a hypothesis from $\mathcal{H}_+$
1. FOUND: = false;
2. g: = LAST_IN(G_+);
3. while not (FOUND or PREVIOUS_IN$(g, G_+) = g$) do
4. begin
5. if $g \notin A$ then
6. begin
7. $A := A \cup \{g\}$;
8. FOUND:= $\min(A^{++} \backslash A) > g$ and HYP(A^+);
9. end;
10. $A := A \backslash \{g\}$;
11. g:=PREVIOUS_IN(g, G_+);
12. end;
13. next_hyp:=FOUND;
14. $\mathcal{H}_+$: = $\mathcal{H}_+ \cup \{A^+\}$;

The function HYP(X) tests whether the positive intent X is a hypothesis, i.e., the condition $\forall f \in G_- \; X \not\subseteq f^-$.

For $X \subseteq G_+$ the function $\min(X)$ returns the smallest (w.r.t. the ordering in G_+) element of X.

The function PREVIOUS_IN(g, G_+) returns g if g is the first element of G_+, and returns the greatest element of G_+ smaller than g, otherwise.

The function LAST_IN(G_+) returns the greatest element of G_+ (i.e., the element with the greatest number).

When the algorithm that computes the set of all hypotheses by calling **Next Hypothesis** generates a new positive intent, it needs to test whether this intent

was not generated before and whether it is not contained in a negative example (line 8). Execution of line 8 takes $(|G_+| \cdot |M| + |G_-| \cdot |M|)$ time. If the positive intent is not contained in any negative example, then the process of concept generation goes further, otherwise, the intent is not a hypothesis and the algorithm backtracks. Thus, the test is executed for no more than $O(|G_+| \cdot |\mathcal{H}_+|)$ positive concepts ($\mathcal{H}_+$ is the set of all positive hypotheses) and the resulting time complexity is $O(|G_+| \cdot |G_-| \cdot |M| \cdot |\mathcal{H}_+|)$. The algorithm for finding all hypotheses has delay $O((|G_+|^2 + |G_-|) \cdot |M|)$, i.e., is a polynomial delay algorithm.

To find all minimal hypotheses, one needs to replace the line 8 with the following line 8* (and make some minor changes in the procedure that calls **Next Hypothesis**):

8*. FOUND:= $\min(A^{++} \backslash A) > g$ and $\forall f \in G_-(A^+ \not\subseteq f^-$ and MINHYP$(A^+))$;

The function MINHYP(X) tests whether the hypothesis X is a minimal hypothesis: first, all sets $(X \cap g^+)$, $g \in G_+ \backslash X$ maximal by inclusion are generated, then the condition $(X \cap g^+) \not\subseteq g_-^-$ is tested for each generated set and each $g_- \in G_-$.

The MINHYP test requires additional $O(|G_+|^2|M| \cdot |G_-|)$ operations for at most $|\mathcal{H}_+|$ hypothesis intents, so the resulting batch algorithm that constructs the set of all minimal hypothesis has $O(|G_+|^2|M| \cdot |G_-| \cdot |\mathcal{H}_+|)$ time complexity. The algorithm that computes all minimal hypotheses by calling **Next Hypothesis** with line 8* is not polynomial-delay. It is not clear whether it satisfies a weaker notion of efficiency, namely has cumulative polynomial delay. Recall that an algorithm listing a set of objects is said to have a *cumulative delay* d [8] if it is the case that at any point of time in any execution of the algorithm with any input of length p the total number of instructions that have been executed is at most $d(p)$ plus the product of $d(p)$ and the number of structures that have been output so far. The cumulative polynomial delay means that $d(p)$ is a polynomial of p.

Below, we give an incremental algorithm **Next Example** that modifies the list $\mathcal{H}_+^m$ of all minimal positive hypotheses if the example context K is updated with a new object g_n.

• If g_n has the goal attribute w, then all implications $H \to w$ of the old context remain true in the new context, but it may happen that some of $H \in \mathcal{H}_+^m$ are no longer minimal hypotheses. These must be replaced by $H \cap g_n^+$, which is clearly a minimal hypothesis.

• If g_n does not have w, then all implications $H \to w$ of the old context with $H \subseteq g_n^-$ become false. If $H \in \mathcal{H}_+^m$ and $H \subseteq g_n^-$, then H is no longer a positive hypothesis (it is called a *falsified* hypothesis), and it is deleted from $\mathcal{H}_+^m$. There may be new minimal hypotheses, which must be of the form $(H \cup \{m\})^{++}$, where $m \notin g_n^-$. These new minimal hypotheses are "most general specializations" of the old ones relative to the new negative example. We systematically generate the sets $(H \cup \{m\})^{++}$ and check if they are minimal hypotheses.

Therefore, our procedure is this:
1. For every $H \in \mathcal{H}^m_+$, $H \subseteq g^-_n$ we delete H from $\mathcal{H}^m_+$.
2. For every $H \in \mathcal{H}^m_+$, $H \subseteq g^-_n$ and every $m \in M \setminus g^-_n$ we construct $F := (H \cup \{m\})^{++}$. The set $\mathcal{F}$ (of tentatively new minimal hypotheses) is updated with F.
3. Each element of $\mathcal{F}$ that is a superset of another element of $\mathcal{F}$ is deleted from $\mathcal{F}$.
4. The family $\mathcal{H}^m_+$ is updated with sets from $\mathcal{F}$. Finally, each element of $\mathcal{H}^m_+$ that is a supersets of $\mathcal{H}^m_+$ is deleted from $\mathcal{H}^m_+$.

Below we present a more formal pseudocode description of the algorithm. Here for two families of sets $\mathcal{X}$ and $\mathcal{Y}$ the function MERGE$(\mathcal{X}, \mathcal{Y})$ takes the union of $\mathcal{X}$ and $\mathcal{Y}$, discards every element set of it that is a superset of another set from the union (only one set is left for a pair of equal sets), and returns the resulting family of sets. Obviously, MERGE$(\mathcal{X}, \mathcal{X})$ selects all sets from $\mathcal{X}$ minimal with respect to the set-theoretic inclusion $\subseteq$.

The function NEXT_IN$(\mathcal{H}^m_+, X,$ Cond) returns "true" if there is an element of $\mathcal{H}^m_+$ that is lectically greater than X and satisfies condition Cond (in this case X takes the value of this element) and returns "false" otherwise. The function FIRST_IN$(\mathcal{H}^m_+, X,$ Cond) returns "true" if there is an element of $\mathcal{H}^m_+$ that satisfies condition Cond (in this case X takes the value of the lectically smallest such element) and returns "false" otherwise.

Next Example

```
0   F := ∅,
1   if w ∈ g_n^± then
2     if FIRST_IN(H^m_+, H, ⊈ g_n^+) then
3       repeat
4         if for all h ∉ w^±   g_n^+ ∩ H ⊈ h^+ then
5           H^m_+ := H^m_+\{H} ∪ {H ∩ g_n^+}
6       until not NEXT_IN(H^m_+, H, ⊆ g_n^+)
7     else
8   if FIRST_IN(H^m_+, H, ⊆ g_n^-) then
9    repeat
10            H^m_+ := H^m_+\{H}
11            for all m ∉ g_n^- do F := F ∪ {(H ∪ {m})^{++}}
12    until not NEXT_IN(H^m_+, H, ⊆ g_n^-)
13  if F ≠ ∅ then
14     begin
15          F := MERGE(F, F)
16          H^m_+ := MERGE(H^m_+, F)
17     end
```

Complexity. If g_n is a new positive example (i.e., $w \in g^{\pm}_n$), then the algorithm terminates in time $O(|\mathcal{H}^m_+| \cdot |G_-| \cdot |M|)$, since it intersects each old minimal hypothesis with g^+_n and tests the inclusion of the result in the intent of each negative example (lines 1-6).

If g_n is a new negative example (i.e., $w \notin g_n^{\pm}$), then in the worst case it will take much more time to terminate. The lines 9-12 are executed in the worst case for each element of $\mathcal{H}_+^m$ and for each $m \in g_n^-$ and, therefore, require $O(|\mathcal{H}_+^m| \cdot |M| \cdot |G_+| \cdot |M|)$ time, since line 11, the most time-consuming operation of the cycle 9-12, is done in $O(|G_+| \cdot |M|)$ time,

Since $\mathcal{F}$ has at most $|\mathcal{H}_+^m| \cdot |M|$ elements, the lines 15 and 16 can be executed in $O(|\mathcal{H}_+^m|^2|M|^2)$ time each. Note that line 15 is not necessary, it can reduce the execution time in practice, but does not affect the upper bound of the worst-case complexity. Thus, the complexity of executing lines 14-17 is also $O(|\mathcal{H}_+^m|^2|M|^2)$ and the total time complexity of the algorithm is $O(|\mathcal{H}_+^m|^2|M|^2 + |\mathcal{H}_+^m| \cdot |M|^2|G|)$.

Note that the worst-time complexity of the algorithm is quadratic in the size of the set of all minimal hypotheses and not of the set of all hypotheses. Therefore, when the number of all minimal hypotheses is much less than the number of all hypotheses, the incremental algorithm can operate faster than the batch algorithm, whose time complexity is linear in the number of all hypotheses.

5 Conclusion

We considered the JSM-method of generating hypotheses from positive and negative examples in terms of Formal Concept Analysis. We presented conditions that ensure correct behavior of hypotheses relative to examples. We showed how hypotheses and minimal hypotheses are related to pseudointent-based implications. We proposed a batch algorithm for computing all hypotheses and/or all minimal hypotheses in time linear in the number of all hypotheses. We also proposed an incremental algorithm for computing all minimal hypotheses, which runs in time quadratic in the number of minimal hypotheses. The example concerning consumer properties of wheel chains illustrated the relationship between hypotheses and implications.

Acknowledgments

The second author was supported by the Alexander von Humboldt Foundation.

References

1. O. M. Anshakov, V. K. Finn, and D. P. Skvortsov, On axiomatization of many-valued logics associated with the formalization of plausible reasonings, *Studia Logica*, **25**, No. 4, 23-47 (1989).
2. V. Duquenne and J.-L. Guigues, Familles minimales d'implications informatives résultant d'un tableau de données binaires, *Math. Sci. Hum.*, **24**, No. 95, 5-18 (1986).
3. V. K. Finn, On Machine-Oriented Formalization of Plausible Reasoning in the Style of F. Backon–J. S. Mill, *Semiotika i Informatika*, No. 20, 35-101, 1983 [in Russian].
4. V. K. Finn, Plausible Reasoning in Systems of JSM Type, *Itogi Nauki i Tekhniki, Seriya Informatika*, **15**, 54-101, 1991 [in Russian].

5. B. Ganter, Algorithmen zur Formalen Begriffsanalyse, in: *Beiträge zur Begriffsanalyse*, (B. Ganter, R. Wille, and K. E. Wolff, eds.), B. I. Wissenschaftsverlag, Mannheim, 241-254, 1987.
6. B. Ganter and R. Wille, Formal Concept Analysis. Mathematical Foundations, Berlin, Springer, 1999.
7. B. Ganter and S. Kuznetsov, Stepwise Construction of the Dedekind-McNeille Completion, Proc. 6th Int. Conference on Conceptual Structures, Montpellier (France), 1998, Lecture Notes in Artificial Intelligence, no. 1453, 295-302.
8. L.A. Goldberg, Efficient Algorithms for Listing Combinatorial Structures, Cambridge University Press, 1993.
9. D.S. Johnson, M.Yannakakis, and C.H. Papadimitriou, On Generating all Maximal Independent Sets, *Information Processing Letters*, **27**, 1988, 119-123.
10. S. O. Kuznetsov, JSM-method as a Machine Learning System, *Itogi Nauki i Tekhniki, ser. Informatika*, no. 15, 17-54, 1991 [in Russian].
11. S.O. Kuznetsov, Mathematical Aspects of Concept Analysis, *Journal of Mathematical Science, Series Contemporary Mathematics and Its Applications*, **18**, 1654-1698, 1996.
12. S.O. Kuznetsov and V.K. Finn, On a model of learning and classification based on similarity operation, *Obozrenie Prikladnoi i Promyshlennoi Matematiki* **3**, no. 1, 66-90, 1996 [in Russian].
13. S.O. Kuznetsov, Learning of Simple Conceptual Graphs from Positive and Negative Examples, in Proc. *Principles of Data Mining and Knowledge Discovery, Third European Conference, PKDD'99*, J. Zytkow, J. Rauch, Eds., Lecture Notes in Artificial Intelligence, **1704**, pp. 384-392, 1999.
14. H. van de Waterbeemd, Ed., *Structure-Property Correlations in Drug Research*, Landes, Austin, 1996.

Generalized Formal Concept Analysis

Laurent Chaudron and Nicolas Maille

Onera Cert, 2 avenue Edouard Belin
BP 4025 - 31055 Toulouse Cedex 04 - FRANCE
Phone: +33 5 62 25 26 55, Fax: +33 5 62 25 25 64
{chaudron, maille}@cert.fr

Abstract. Formal Concept Analysis is an algebraic model based on a Galois connection that is used for symbolic knowledge exploration from an elementary form of the representation of data ("formal context"). The aim of this paper is to design the theoretical models required for the extension of Formal Concept Analysis to any kind of lattice-structured set of properties ("generalized formal concept"), and especially the case of predicates. Beyond their theoretical interest, the models aim at better solving real applied problems thanks to an improvement of the knowledge representation skills.

1 Introduction

1.1 Formal Analyses

As far as knowledge information is concerned, the induction of clustered groups of entities from a context described through their features is a pivotal topic. Generally, if numerical valuations (belief measures, preferences...) can be defined on the considered data, numerical or mixed methods can be directly used: classical classification tools, fuzzy methods and, closer to the symbolic community, rough sets [20] or cartesian space model [13]... In some cases, requirements or accessibility constraints imply to rest only on symbolic attributes. Thus, more fundamental models and techniques are required; Formal Concept Analysis - also called "Galois Lattices" - [10] is a suitable candidate for such a purpose. Even if FCA theory relies on an elementary form of the representation of data (the "cross table"), the use of more complex data types, such as "many-valued contexts", is also possible. The data have then to be reduced to the basic type by a method of interpretation. A large amount of applications require to characterize the context with more detailed attributes (e.g. properties with arguments: $Speed(vehicle_1, 12), Aircraft(BO727, landing)$). Defining an adequate method of interpretation which clarify all the links between such attributes is not obvious. The purpose of this paper is to give a new framework which allows the concept lattice to be calculated without reducing the data to the basic type when objects are described by properties within a lattice-structured set, such as sets of literals from a first order logical language.

The relations between FCA and first-order logic have been studied, especially in [22]. In this work the model-logic approach is adopted and the entities are considered as constant symbols that appear as arguments of predicates representing

B. Ganter and G.W. Mineau (Eds.): ICCS 2000, LNAI 1867, pp. 357–370, 2000.
© Springer-Verlag Berlin Heidelberg 2000

the properties. This consistent formal construction does not aim at directly representing knowledge of *a priori* given entities with their attributes. Our approach may be considered as closer to the original purpose of FCA as it is more focused on the formal prerequisites of a consistent model of predicate-logic like FCA and its programming conditions. This leads to search for first-order operators corresponding to the fundamental ones in FCA: set union, set intersection, and the Galois connection (in fact, nothing else but these three operators is needed to generate the FCA theory). In a first step, an extension of FCA to any kind of lattice-structured set of properties is built: the *Generalized Formal Concept Analysis* (say: *G-FCA*). Then a lattice structure is given to sets of literals by the *Cube* model and a predicate based model is designed and implemented in Prolog: the *Cubical Formal Concept Analysis (C-FCA)*. C-FCA is a formal extension of FCA (every context of FCA is a context of G-FCA without any transformation), while G-FCA is a general frame for both. Nevertheless, it is clear that every concept lattice generated by G-FCA (or C-FCA) is isomorphic to some concept lattice of FCA. Reducing the data of the G-FCA (or C-FCA) to the basic type in order to build this isomorphic concept lattice with FCA theory is always possible, but difficult.

Foreword: In the context of the paper the proofs are omitted, all of them are completely detailed in [8] and [16].

1.2 Application

Throughout this paper, a very simple example of aeronautical incidents analysis is used. The information about aeronautical problems comes from many reports produced by the crews of the implied airplanes. Among the tremendous amount of these heterogeneous incident reports, the main objective of the aeronautical community is to find out means to discover precursors of accidents. This supposes to have tools to represent knowledge and to discover links between features of incidents and accidents. In our example (which was extracted from large data of the Nasa incidents database [17]), the data are represented in a synthetic way by table 1.

Table 1. Description of aeronautical accidents

Accident	Persons	Aircraft		Scenario
		Plane	Flight Phase	
Acc1	crew(BO727) ATC	BO727 DC9	Take Off Take Off	Airmissground
Acc2	ATC crew(MD82)	BO727 MD82	Landing Landing	Airmissground
Acc3	ATC	B0727 DC9	Landing Take Off	Airmissground
Acc4		MD82	Landing	Windshear

For example, the first line captures **(i)** that in the accident denoted "*Acc1*" the crew of the Boeing 727 and the air traffic control (ATC) have both a part of responsibility, **(ii)** that both aircraft (Boeing 727 and DC9) were landing and **(iii)** that the global scenario of the incident was a so-called "airmissground".

The aim of data analysis is to cluster the accidents and to point out the specificity of each set. The better the connections between accidents are described, the better the typical dangerous configurations are highlighted.

2 A Generalization of Formal Concept Analysis

Formal Concept Analysis (say: FCA) is a set-theoretical model for concepts that captures the philosophical understanding of a concept as a context-based unit of thought consisting in two parts: the *extent*, which contains all the entities (the objects, the examples...) belonging to the concept, and the *intent*, which is the collection of all the attributes (the characteristics, the properties...) shared by the entities [1]. Based upon Galois connections, FCA was described in [2] and in the 80's Rudolf Wille designed a dedicated theory and launched a research program at the University of Darmstadt. An introduction can be found in [9] and FCA theory is now described in the reference book of Ganter and Wille [10].

In FCA, the basic notion that models the knowledge about a specific domain is the formal *context* –described as a binary relation between two sets– from which concepts and conceptual double hierarchies can be formally derived so as to form the mathematical structure of a lattice[1] with respect to a subconcept-superconcept relation. FCA is used for self-emergent classification of objects, detection of hidden implications between objects, construction of concept sequences, object recognition, aggregation of data and information, knowledge representation and analysis. FCA is also frequently used as a preprocessing tool for classification [6].

Classical definitions of FCA are not recalled, but the method is exemplified with a naïve interpretation of table 1.

2.1 Example of Fundamental Formal Concept Analysis

Let the data of the table 1 be reduced to the table 2 which define a relation R between the set of objects O={*Acc1,Acc2,Acc3,Acc4*} and the set of attributes P = {*crew-BO727, crew-MD82, ATC, BO727-take-off, BO727-landing, DC9-take-off, MD82-landing, Airmissground, Windshear*}. $C_1 = (O, P, R)$ is a formal context.

The Galois connection - denoted $'$ - joins a set of objects with the set of attributes common to objects and conversely it joins a set of attributes with the

[1] given two internal operators $\sqcap$ (infimum) and $\sqcup$ (supremum) on a set E, $(E, \sqcap, \sqcup)$ is a lattice, iff$_{def}$: $\sqcap$ and $\sqcup$ are idempotent, associative, commutative and they verify the absorption laws $x \sqcap (x \sqcup y) = x$ and $x \sqcup (x \sqcap y) = x$. A lattice is always an ordered set: the relation $\leq$ defined on E as: $(x \leq y) \leftrightarrow_{def} (x \sqcap y) = x$ is an order relation for which $\sqcap$ and $\sqcup$ represent the greatest lower bound and the least upper bound [3].

Table 2. A naïve interpretation of the table 1

R	*crew-BO727*	*crew-MD82*	*ATC*	*BO727-take-off*	*BO727-landing*	*DC9-take-off*	*MD82-landing*	*Airmiss-ground*	*Wind-shear*
Acc1	×		×	×		×		×	
Acc2		×	×		×		×	×	
Acc3			×		×	×		×	
Acc4							×		×

set of objects which have all those attributes. We have: {*Acc1, Acc2*}' = {*ATC, Airmissground*} and {*ATC, Airmissground*}' = {*Acc1, Acc2, Acc3*}.

A concept of this context is a pair (A, B) with A a set of objects, B a set of attributes, $A' = B$ and $B' = A$.

As {*Acc1, Acc2, Acc3*}' = {*ATC, Airmissground*}, the pair *(*{*Acc1, Acc2, Acc3*}*,* {*ATC, Airmissground*}*)* is a concept of C_1.

In fact, 10 concepts are induced by C_1:
1: *(*{*Acc1,Acc2,Acc3,Acc4*},{}*)*
2: *(*{*Acc1,Acc2,Acc3*},{*ATC,Airmissground*}*)*
3: *(*{*Acc2,Acc4*},{*MD82-landing*}*)*
4: *(*{*Acc2,Acc3*},{*ATC,Airmissground,BO727-landing*}*)*
5: *(*{*Acc1,Acc3*},{*ATC,Airmissground,DC9-take-off*}*)*
6: *(*{*Acc4*},{*MD82-landing,Windshear*}*)*
7: *(*{*Acc2*},{*ATC,Airmissground,BO727-landing,Crew-MD82,MD82-landing*}*)*
8: *(*{*Acc3*},{*ATC,Airmissground,DC9-take-off,BO727-landing*}*)*
9: *(*{*Acc1*},{*ATC,Airmissground,DC9-take-off,BO727-take-off,Crew-BO727*}*)*
10: *(*{}*,P)*

These concepts have a lattice structure as illustrated by figure 1 (the Hasse-diagram of the lattice). Traditionally, on such a diagram, an element labeled by an object *Obj* represents the concept with the smallest set containing *Obj*, and an element labeled by a property *Prop* represents the concept with the smallest set containing it. Hence, a given concept ⓝ inherits all the properties which are linked above it in the diagram and ⓝ is constituted of all the objects which are linked below it.

FCA allows us to find the common attributes of sets of accidents. Nevertheless, we can see on the diagram that accidents 3 and 4 are not related even if, in the data of section 2, both accidents involved a landing plane. Thus, the characterization of accidents by the table 2 does not lead to describe precisely enough all the connections. The description of table 1 is closer to first order logic. Unfortunately, such a richer representation cannot be submitted to the simple set-theory laws as: {*Aircraft(BO727,take-off)*}$\cap${*Aircraft(DC9,take-off)*}$= \emptyset$ although they clearly have common pieces of knowledge.

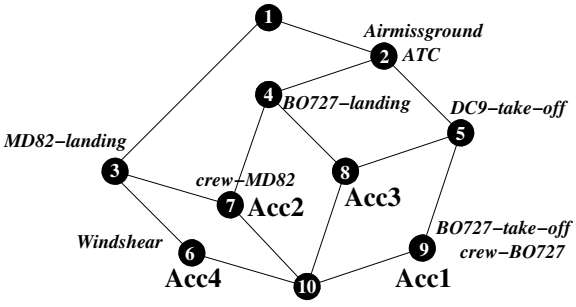


Fig. 1. The Concept Lattice L_1

On one hand, we can try to make clear all the connections contained in the representation of table 1. Thus, the attribute *Aircraft(BO727,take-off)* could be represented by four elementary properties: *aircraft, BO727, take-off, BO727-take-off*. Nevertheless, considering the description of *Acc*1 of table 1, a lot of elementary properties will have to be derived, such as:

the two aircraft are taking off;
the two aircraft are in the same flight phase;
the crew of an airplane which is taking off is implied in the accident;
the crew of an airplane which is in the same flight phase than a DC9 is implied in the accident...

Therefore it will be difficult to describe all the actual elementary properties which are directly captured by the first-order representation.

On the other hand, we can keep the first-order representation of table 1 and search for first-order corresponding operators to the fundamental ones in FCA: set union, set intersection and the Galois connection. Thus, two models must be designed: 1: a generic extension of FCA to all kind of lattice (this is the purpose of the next section), 2: a lattice structure on conjunctions of predicate literals that copies the set-theory operators of the propositional calculus (this is the Cube model described in section 3.1).

2.2 Generalized Formal Concept Analysis

In this section, we build an extension of the fundamental definitions of formal concept analysis theory to the more general case in which the properties of the objects in O belongs to any kind of lattice. Therefore, we define the new context and all the related tools.

Definition 1
A *generalized context* $\mathbb{K}$ is a triple $(O, (\mathfrak{L}, \leq, \sqcap, \sqcup), \zeta)$ where O is a finite set, $(\mathfrak{L}, \leq, \sqcap, \sqcup)$ is a lattice and ζ is a mapping from O to $\mathfrak{L}$.
$'$ and $^\circ$ are two mappings defined by:

$$\begin{array}{rcl} ' : \mathcal{P}(O) & \longrightarrow & \mathfrak{L} \\ X & \longmapsto & X' = \sqcap_{x \in X} \zeta(x) \end{array} \quad \text{and} \quad \begin{array}{rcl} \circ : \mathfrak{L} & \longrightarrow & \mathcal{P}(O) \\ Y & \longmapsto & Y^\circ = \{x \in O \mid Y \leq \zeta(x)\} \end{array}$$

We proved in [16] that this pair of maps is a Galois connection; they satisfy the classical properties:

$$X \subseteq X_1 \Rightarrow X' \geq X_1' \qquad X \subseteq X'^\circ$$
$$Y \leq Y_1 \Rightarrow Y_1{}^\circ \subseteq Y^\circ \qquad Y \leq Y^{\circ\prime}$$

Furthermore it is easy to prove the following proposition:

Proposition 1
Let $\mathbb{K} = (O, (\mathfrak{L}, \leq, \sqcap, \sqcup), \zeta)$ be a generalized context, X a subset of O and Y a subset of $\mathfrak{L}$.

$$X' = X'^{\circ\prime} \qquad Y^\circ = Y^{\circ\prime\circ}$$
$$(\bigcup_{j \in J} X_j)' = \bigsqcap_{j \in J} X_j' \qquad (\bigsqcup_{j \in J} Y_j)^\circ = \bigcap_{j \in J} Y_j^\circ$$

We are now equipped enough to define a concept of a generalized context as a couple of objects and properties stable for the $'$ and $\circ$ operators.

Definitions 2
A *generalized concept* from the generalized context $\mathbb{K} = (O, (\mathfrak{L}, \leq, \sqcap, \sqcup), \zeta)$ is a couple (X, Y) such that $X \subseteq O$, $Y \in \mathfrak{L}$, $X' = Y$, $Y^\circ = X$.
The set of all the concepts of the context $\mathbb{K}$ is denoted by $\mathfrak{TC}(\mathbb{K})$.
The order relation is defined as a mapping $\ll$:

$$\begin{array}{rcl} \ll : \mathfrak{TC}(\mathbb{K}) \times \mathfrak{TC}(\mathbb{K}) & \longrightarrow & \{\ \mathsf{True}, \mathsf{False}\} \\ ((X_1, Y_1), (X_2, Y_2)) & \longmapsto & \mathsf{True} \text{ iff } X_1 \subseteq X_2 \end{array}$$

Proposition 2 $\ll$ is an order relation on $\mathfrak{TC}(\mathbb{K})$

Definition 3 The infimum $\wedge$ and the supremum $\vee$ are respectively defined by:

$$(X_1, Y_1) \wedge (X_2, Y_2) = (X_1 \cap X_2, (Y_1 \sqcup Y_2)^{\circ\prime})$$
$$(X_1, Y_1) \vee (X_2, Y_2) = ((X_1 \cup X_2)'^\circ, Y_1 \sqcap Y_2)$$

And the fundamental theorem is:

Theorem 3 $(\mathfrak{TC}(\mathbb{K}), \ll, \wedge, \vee)$ is a complete lattice

Examples
• FCA is a particular case of G-FCA. Indeed, by replacing the triple of operators $(\ll, \wedge, \vee)$ by $(\subseteq, \cap, \cup)$ the definitions and results are those of the fundamental FCA.
• Let $\mathcal{L}^1$ be a classical first-order logic language, and $\mathcal{F}^1$ the set of all well formed formulas. The "common sense implication[2]" $\hookleftarrow$ defines a preorder on

[2] implication $\rightarrow$ by itself is not a relation, it is a functor. Common sense implication - say "$\hookrightarrow$" - between formulas must be strictly defined as: $(\varphi \hookrightarrow \psi)$ iff$_{def}$ $\vdash (\varphi \rightarrow \psi)$.

$\mathcal{F}^1$ and considering its correlated equivalence relation $\sim$, the quotient set $\mathcal{F}^1_{\sim}$ is an ordered set. Moreover $(\mathcal{F}^1_{\sim}, \hookleftarrow, \wedge, \vee)$ is a lattice. Stating $\mathfrak{L} = \mathcal{F}^1_{\sim}$, thanks to the generalized concept analysis, we define a complete *first-order formal concept analysis* in which each object of a finite set O is characterized by a finite set of well formed formulas (i.e. a knowledge base). This structure may get theoretical interest for Knowledge Base Clustering and more generally for Symbolic Fusion.

But in the case of real applications based on perception of information, the representations of entities relies on more simple formula and first order formal concept analysis must be specialized so as to consider balanced complex terms to capture the attributes. This is the aim of the Cube model which deals with simplified formula as conjunctions of literals.

3 Cubical Formal Analysis

3.1 The Cube Model

The cube model is dedicated to formalize conjunctions of properties and in this study, it will allow first order contexts to be defined. The cube model is based on a classical first order language (*Const, Var, Funct, Pred*) whose set of terms *Term* is the functional closure of *Const* $\cup$ *Var* by *Funct*. Such an approach has been used for knowledge representation in intelligence systems and cooperative systems [7]. The elementary properties are represented by literals and the elements of their power set $\mathcal{C}$ are called *logical cubes*[3], they are interpreted as the conjunction of the literals. Cubes play a dual role besides the classical clauses, and by default their variables are existentially quantified.

In expert or deductive systems, knowledge squares with general rules that are captured by clauses: $c = \{\neg Aircraft(x, Landing), Pers(Crew(x))\}$ represents the information *"landing aircraft have crew"*; the associated logical formula is $\forall x$ *(¬Aircraft(x,Landing)* $\vee$*Pers(Crew(x)))*. When we plan to describe as a context the state of an observed situation, cubes are more adequate and $c =$ *{¬Aircraft(x,Landing), Pers(Crew(x))}* means: *"the crew of an airplane which is not in the landing phase is implied in the accident"*. The logical interpretation is $\exists x\ (\neg Aircraft(x, Landing) \wedge Pers(Crew(x)))$[4].

Thus, this implication $\hookrightarrow$ is a relation between individual well formed formulas, which is a preorder.

[3] "cube" is the name that was used for the first time by A. Thayse [21].

[4] Remark: the attributes *¬Aircraft(x,Landing)* and *¬Aircraft(y,Landing)* have the same meaning but the cubes *{¬Aircraft(x,Landing), Pers(Crew(x))}* and *{¬Aircraft(y,Landing), Pers(Crew(x))}* have not. Hence, describing an object by the attribute *¬Aircraft(x,Landing)* (which is equivalent to *¬Aircraft(y,Landing)*) and by the attribute *Pers(Crew(x))* has not the same meaning than describing the object by the cube *{¬Aircraft(x,Landing), Pers(Crew(x))}*. That is why a mapping ζ from O to $\mathfrak{L}$ is used in the definition of a generalized context rather than a relation (a relation cannot capture links between variables).

As far as knowledge representation of a context is concerned, we would like the order relation induced on $\mathcal{C}$ to capture the intuitive notion of "information enrichment". But such an "enrichment" can be obtained via different means: quantity of information, precision of terms, logical dependency. For instance, $\{Aircraft(BO727, Landing), Pers(Crew(x))\}$ is more informative than $\{Pers(Crew(x))\}$ for the number of literals is higher; and $\{Pers(Crew(DC9))\}$ is more informative than $\{Pers(Crew(x))\}$ (with x variable) for a sake of precision. Unfortunately, the combination of both intuitive criteria may lead to a more complex definition of the order relation as {*Aircraft(BO727,Landing), Pers(Crew(x))*} cannot be directly compared to {*Pers(Crew(DC9))*}. The previous cases highlight the need for sound definitions to the intuitive notions of union and intersection of two finite information sets in accordance to the following requirements: the infimum has to capture the common features (while giving more information than the empty set frequently generated by the unification rule). The supremum has to cope with the complementary criteria: quantity/precision of the information (while giving a more synthetic result than the set union). Such purposes are achieved thanks to an algebraic approach.

Definition 4 $(\forall c_i \in \mathcal{C})$ c_1 *subsumes* c_2, and we write $c_1 \leq_c c_2$, when $c_1\sigma \subseteq c_2$ for some substitution σ.
c_1 is said to be *equivalent* to c_2, $c_1 \equiv_c c_2$ when $c_1 \leq_c c_2$ and $c_2 \leq_c c_1$.

As $\leq_c$ is a preorder, $\equiv_c$ is an equivalence relation.

Remark: symbol $\subseteq$ is equivalent to $\subset$ and represents the set inclusion. $\subsetneq$ means "included but not equal" and must not be confused with $\not\subseteq$ (not included).

Definition 5 A cube c is *reducible* if there exists a substitution θ such that $c\theta \subsetneq c$.

Proposition 4 An irreducible reduction of a cube c always exists and is unique up to variable renaming. It is denoted as *reduc(c)* and the set of reduced cubes is denoted as $\mathcal{C}^r$. We also have $reduc(c) \equiv_c c$.

Remark: if a cube c_0 does not contain any variable (i.e. $c_0 \in \mathcal{C}_0$) then $reduc(c_0) = c_0$.

Example It is clear that *reduc({a(x),a(1)})={a(1)}*, but *reduc({a(1,x),a(y,2)}) = {a(1,x),a(y,2)}*.

The order relation $\leq_c$ formalizes information enrichment. Two cubes are equivalent iff they capture the same piece of knowledge. As any information can be represented by a reducible cube, we will define the infimum and supremum operators on $\mathcal{C}^r$. We adopt the approach of [12] and [14] which allows a lattice on the terms algebra to be defined properly thanks to the *anti-unification* operator.

Definition 6 Let Φ be any bijection from $Term \times Term$ to Var.

$antiunif_\Phi(f(s_1, ..., s_m), f(t_1, ..., t_m)) =_{def}$
$f(antiunif_\Phi(s_1, t_1), ..., antiunif_\Phi(t_m, s_m))$
for every function or constant symbol f,
$antiunif_\Phi(s, t) =_{def} \Phi(s, t)$ otherwise.

This definition is extended to anti-unification on literals and finally on cubes as a mapping: $antiunif_\Phi : \mathcal{C} \times \mathcal{C} \to \mathcal{C}$ which is associative and commutative.

Example: $p(x, g(y, b))$ is the anti-unified literal of $p(a, g(a, b))$ and $p(1, g(b, b))$.

In fact, anti-unification allows the infimum to generalize the terms so as to properly enrich the set intersection on the cubes. The result of the anti-unification of two cubes c_1 and c_2 is defined as the union of the anti-unification of every couple (l_1, l_2) based on the same predicate name and such that l_1 belongs to c_1 and l_2 belongs to c_2.

Definition 7 Let c_1 and c_2 belong to $\mathcal{C}^r$. The infimum and supremum operators $\cup_c$ and $\cap_c$ are defined on $\mathcal{C}^r$ as follows:

$c_1 \cup_c c_2 =_{def} reduc(c_1 \cup c_2)$;
$c_1 \cap_c c_2 =_{def} reduc[antiunif_\Phi(c_1, c_2)]$.

Theorem 5 $(\mathcal{C}^r, \leq_c, \cup_c, \cap_c)$ is a lattice.

Remark: by a simple duality argument, a lattice structure on cubes can be derived from the lattice structure defined on clauses (modulo $\equiv_c$) by Plotkin (see [18] page 163). This lattice on $\mathcal{C}_{/\equiv_c}$ includes the present lattice on $\mathcal{C}^r$ (up to variable renaming) but the definitions of infimum and supremum are presented here in a more mathematical way allowing an easier implementation to be made, as the extension of Huet's recursive anti-unification algorithm to the cubes lends itself well to a CLP implementation. Thus the cube approach proposes a more easily computable definition of the lattice than the initial definition of Plotkin. Moreover this definition is restricted to a class of cubes that fits properly to capture the knowledge of our applications; it has also been implemented in Prolog.

Let us denote $\mathcal{C}_0$ the set of all the finite subsets of positive literal of the propositional calculus. The corollary 6 shows that the cube lattice is a formal extension of the propositional set-theory based lattice.

Corollary 6 $(\mathcal{C}_0, \subseteq, \cup, \cap)$ is a sublattice of $(\mathcal{C}^r, \leq_c, \cup_c, \cap_c)$.

3.2 Cubical Formal Analysis

We are now ready to define a new formal concept analysis model in which the attributes of the objects are captured by a set of literals from a first order logical language instead of literals from a propositional language. Thus an object of the context is characterized by a logical cube.

Definitions 8 A cubical context is a pair (O, ξ) where O is finite set of objects, and ξ is a mapping from O to $\mathcal{C}^r$.

This is a particular case of G-FCA in which the notation of ζ is replaced by ξ. Here, each object o in O has one and only one image $p = \xi(o)$ in $\mathcal{C}^r$ which represents the set of (predicate) properties of o.

Definitions 9 The dual operators $'$ and $^\circ$ between O and $\xi(O)$ are defined by:

$$A' =_{def} \cap_{\mathbf{c}\ o_i \in A}\ \xi(o_i) \text{ and } B^\circ =_{def} \{o_i \in O \mid B \leq_c \xi(o_i)\}$$

Thanks to G-FCA, we know that the pair of operators $'$ and $^\circ$ is a Galois connection between O and $\xi(O) \subset \mathcal{C}^r$.

Definitions 10 A *cubical concept* is a pair (A, B) $A \subset O,\ B \in \mathcal{C}^r$ such that:

$$A' = B \text{ (up to variable renaming) and } B^\circ = A$$

The set of all cubical concepts defined by the context (O, ξ) is denoted as L^c.

Proposition 7 For cubical concepts (A_1, B_1) and (A_2, B_2) the relation defined by: $(A_1, B_1) \sqsubseteq (A_2, B_2) \Leftrightarrow A_1 \subseteq A_2 (\Leftrightarrow B_2 \leq_c B_1)$ is an order relation on L^c.

Definitions 11 The *supremum:* $\sqcup$ and *infimum:* $\sqcap$ are respectively defined on L^c as follows:

$$(A_1, B_1) \sqcup (A_2, B_2) =_{def} ((A_1 \cup A_2)'^\circ, B_1 \cap_c B_2)$$
$$(A_1, B_1) \sqcap (A_2, B_2) =_{def} (A_1 \cap A_2, (B_1 \cup_c B_2)^{\circ'})$$

Theorem 8 $(L^c, \sqsubseteq, \sqcup, \sqcap)$ is a lattice.

Example Back to our accident analysis application, table 1 is more precisely captured by the cubical context C_2:

ξ(Acc1) = {Pers(Crew(BO727)), Pers(ATC), Aircraft(BO727,TkOff), Aircraft(DC9,TkOff), Scen(Airmissground)}

ξ(Acc2) = {Pers(ATC),Pers(Crew(MD82)), Aircraft(BO727,Landing), Aircraft(MD82,Landing),Scen(Airmissground)}

ξ(Acc3) = {Pers(ATC),Aircraft(BO727,Landing), Aircraft(DC9,TkOff), Scen(Airmissground)}

ξ(Acc4) ={Aircraft(MD82,Landing), Scen(Windshear)}

For instance, we have:

{Acc1,Acc2}′ = inf(ξ(Acc1),ξ(Acc2)) = {Pers(ATC),Pers(Crew(x)), Aircraft(BO727,y), Aircraft(x,y), Scen(Airmissground)}

and :

{Pers(ATC),Pers(Crew(x)), Aircraft(BO727,y), Aircraft(x,y), Scen(Airmissground)}° = {Acc1,Acc2}

Clearly, as {*Acc1,Acc2*} is the intent of a concept, the two accidents are correlated and it is interesting to notice that the intent reveals that in both cases a B727 was involved with another aircraft x in a collision scenario. Moreover they were together in the same traffic phase y (thus, may be, contributing to the overshoot of the ATC's workload) and the crew of the second aircraft $Crew(x)$ was also involved. This kind of mixed links was hidden through the propositional representation and revealed by the cubical formal analysis. Such a knowledge mining through incidents/accidents reports is of a major interest for aeronautical safety programs.

The 12 cubical concepts and the diagram of the concept lattice L_2^c are:

1: ({Acc1, Acc2, Acc3, Acc4},{Scen(x), Aircraft(y,z)})
2: ({Acc1, Acc2, Acc3},{Pers(ATC), Aircraft(BO727,z), Scen(Airmissground)})
3: ({Acc2, Acc4},{Scen(x), Aircraft(MD82,Landing)})
4: ({Acc2, Acc3},{Scen(Airmissground), Aircraft(BO727,Landing), Pers(ATC)})
5: ({Acc1, Acc3},{Scen(Airmissground), Aircraft(DC9,TkOff), Aircraft(BO727,z), Pers(ATC)})
6: ({Acc4},{Aircraft(MD82,Landing), Scen(Windshear)})
7: ({Acc3},{Pers(ATC), Aircraft(BO727,Landing), Aircraft(DC9,TkOff), Scen(Airmissground)})
8: ({Acc2},{Pers(ATC) ,Pers(Crew(MD82)), Aircraft(BO727,Landing), Aircraft(MD82,Landing), Scen(Airmissground)})
9: ({Acc1},{Pers(Crew(BO727)), Pers(ATC), Aircraft(BO727,TkOff), Aircraft(DC9,TkOff), Scen(Airmissground)})
10: ({},{All-properties})
11: ({Acc2,Acc3,Acc4},{Scen(x), Aircraft(y,Landing)})
12: ({Acc1,Acc2},{Pers(ATC), Pers(Crew(x)), Aircraft(x,z), Aircraft(BO727,z), Scen(Airmissground)})

From the previous constructions (Corollary 6) it is easy to prove:

Corollary 9 C-FCA is an extension of FCA.
In other words: any fundamental concept lattice is a cubical concept lattice.

For instance, it is easy to notice that the cubical concept lattice L_2^c gets L_1 (see fig. 1) as a sublattice in which concepts 11 and 12 emerged. Indeed, if a first order context (O, ξ) contains no variable, it can be considered without any transformation as a fundamental context and its concept lattice, say L_0, can be computed. Due to the presence of functional terms and thanks to the anti-unification, the computation of its cubical concept lattice, L_0^c may induce the presence of variables, thus $L_0 \subseteq L_0^c$. Nevertheless an adequate reduction of the data of the first order context can always lead to an isomorphic lattice with fundamental FCA (but the corresponding context would not be a direct representation of the real data).

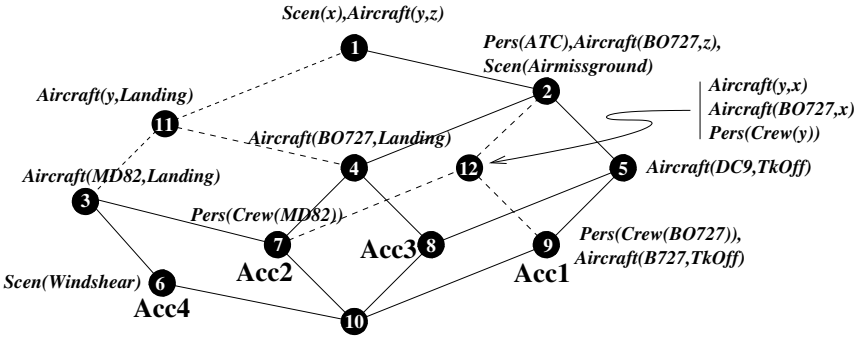


Fig. 2. The Cubical Concept Lattice L_2^c

3.3 Implementations and Exploitations

Prolog programs were designed so as to implement both the Cube model and the the Galois connection. Such programs can be considered as functions which associate to each context C its Concept Lattice L: $C \longrightarrow L$. The determination algorithm relies on Theorem 3: the concept lattice L is determined by the finite sequence T_n of sets of concepts:

$T_0 = \{(\emptyset, P)\}$, $T_1 = \{(o'^{\circ}, o') |\ o \in O\}$
$T_{n+1} = \{c_i \vee c_j |\ c_{i,j} \in T_n\}$
$T_{n_0} = \{(O, \emptyset)\}$, $L = \bigcup_{n \in [0, n_0]} T_n$

In each case, the context and the concept lattice can be considered as a global Prolog knowledge base $C \cup L$ on which knowledge exploration experiments are performed[5]. The knowledge base $C \cup L$ is used so as to look for contextual dependencies between either objects or attributes. The induction of the context-based rules is given by the generic frame of [11] widely developed in *ConImp* by Burmeister [5]. For the propositional case, the techniques are based on the property: $(\forall A \in P), \models (A \rightarrow (A'' - A))$.

A context rule generator was implemented so as to compute the set of proposition rules deducible form the context. But unfortunately, in the first-order case, the mechanisms described in the previous reference articles cannot be simply adapted for the following reason:

- all the set based definitions and mappings that are classically defined must be completely revised based on the cube model so as to make use of predicates;
- moreover, the subtraction operator used in the formula $A \rightarrow (A'' - A)$ gets no equivalent when predicates are concerned.

[5] it must be noticed that the computing time is not a critical problem in our projects as the situations we are analyzing are characterized by a high complexity and a very low evolution time. The lattice must be completely determined and it is completely calculated.

Hence, a specific work is currently under on cube subtraction. At this time the basic result we use is: $(\forall A \in P), \models A \rightarrow A^{\circ'}$.

The logical links between the features are captured by first order rules which are translated into production rules; this extension is currently intensively experimented. These programming results will not be detailed here, but one can notice that the cubical models allow new inner logical constraints to be taken into account thanks to the variables. Comparisons between the C-FCA and proximate models such as logical scaling [19] are under study. Indeed, the favorite domain of FCA and C-FCA is the symbolic knowledge, but thanks to the CLP experiments, many improvements are expected for the analysis of large numerical databases as far as an new extension of the cube model is now completely defined: the Constrained-Cube model - in which the arguments of cubes are submitted to order constraints - will allow to take into account a larger scale of applications thanks to a CC-FCA (Constrained Cubical Formal Concept Analysis) which is the next step of our study [16].

4 Conclusion

Thanks to the design of a Generalized Formal Concept Analysis, the integration of the Cube model - a lattice structure on conjunctions of predicate literals - in the fundamental FCA was defined as a *Cubical FCA*. Its theoretical interests rely also on several applications in which predicates and arguments are required for knowledge exploration. A current study focuses on first order context based rules generation which related to the classical approaches of the ILP[6] community. The new applications are pilot activity modeling (in which context dependent incidents are searched for [15]), and the correlations between the context and the interactions between artificial systems and persons (e.g. noise/annoyance analysis) [4].

References

[1] A. Arnaud and P. Nicole. *La logique ou l'art de penser.* Gallimard, 1992, Paris, original publication 1662. in French.
[2] M. Barbut and B. Monjardet. *Ordre et classification : Algèbre et combinatoire.* Collection Hachette Université. Hachette, Paris, 1970. in French.
[3] G. Birkhoff. *Lattice Theory.* ACM, 1940.
[4] M. Boyer and L. Chaudron. Airport noise/annoyance analysis. In *ICAS Congress.* Intl. Coouncil for the Aeronautical Sciences, 27 August-1 September 2000.
[5] Peter Burmeister. *Programm zur Formalen Begriffsanalyse einwertiger Kontexte.* TH Darmstadt, 1987.
[6] C. Carpineto and G. Romano. A lattice conceptual clustering system and its application to browsing retrievial. *Machine Learning*, 24:95–122, 1996.
[7] L. Chaudron, C. Cossart, N. Maille, and C Tessier. A purely symbolic model for dynamic scene interpretation. *International Journal on Artificial Intelligence Tools*, 6(4):635–664, 1997.

[6] Inductive Logic Programming

[8] L. Chaudron and N. Maille. Le modéle des cubes : représentation algébrique des conjonctions de propriétés. Technical Report 1/7601.43 DCSD-T, Onera, 1999.
[9] B.A Davey and H.A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, 1990.
[10] Bernhard Ganter and Rudolf Wille. *Formal Concept Analysis, Mathematical foundations*. Springer–Verlag, 1999. Isbn 3-540-62771-5.
[11] J-L Guigues and V. Duquenne. Familles minimales d'implications informatives résultant d'un tableau de données binaires. *Math. Sci. Hum.*, 95:5–18, 1986.
[12] G.P. Huet. *Résolution d'équations dans des langages d'ordre 1,2,...,ω*. PhD thesis, Université de Paris VII, 1976. Thèse d'État (in French).
[13] M. Ichino and H. Yaguchi. *Data Science, Classification and related Methods*, chapter Symbolic Pattern Classifiers Based on the Cartesian System Model, pages 358–369. Springer-Verlag, 1998.
[14] J-L. Lassez, M.J. Maher, and K. Marriott. *Foundations of Deductive Databases and Logic Programming*, chapter Unification Revisited. J. Minker, 1987.
[15] P. Le Blaye, F. Dehais, and L. Wioland. Human activity modeling for flight analysis. In *Proceedings of HCP'99, Intl. Conf. on Human Centered Processes*, Brest, France, 22-24 September 1999. Euro.
[16] Nicolas Maille. *Un modèle logico-algébrique pour la fusion symbolique et l'analyse formelle*. PhD thesis, Supaéro, 22 novembre 1999.
[17] Hum. Fact. Lab. Nasa. `http://human-factors.arc.nasa.gov/IHS/ASRS/`, 2000.
[18] G. D. Plotkin. *Machine Intelligence*, volume 5, chapter 8. A note on inductive generalization, pages 153–163. Edinburgh University Press, Edinburgh, 1970. B. Meltzer and D. Michie ed.
[19] Susanne Prediger. Logical scaling in formal concept analysis. In D. Lukose, H. Delugach, M. Keeler, L. Searle, and J. F. Sowa, editors, *Conceptual structures: Fulfilling Peirce's dream*, number 1257 in Lecture Notes in Artificial Intelligence, Berlin–Heidelberg–New York, 1997. Springer–Verlag.
[20] A. Skowron and L. Polkowski. *Rough Sets in Knowledge Discovery*. Physica-Verlag, Heidelberg, 1998.
[21] A Thayse and col. *Approche logique de l'Intelligence Artificielle*. Dunod, Paris, 1989. in French.
[22] M. Zickwolf. *Rule exploration: First Order Logic in Formal Concept Analysis*. PhD thesis, THD Darmstadt Univ., 1991. Short English version, nr1580, 1993.

A Logical Generalization of Formal Concept Analysis

Sébastien Ferré and Olivier Ridoux

IRISA, Campus Universitaire de Beaulieu, 35042 RENNES cedex,
{ferre,ridoux}@irisa.fr

Abstract. We propose a generalization of Formal Concept Analysis (FCA) in which sets of attributes are replaced by expressions of an almost arbitrary logic. We prove that all FCA can be reconstructed on this basis. We show that from any logic that is used in place of sets of attributes can be derived a contextualized logic that takes into account the formal context and that is isomorphic to the concept lattice. We then justify the generalization of FCA compared with existing extensions and in the perspective of its application to information systems.

1 Introduction

The origin of this work is the search for flexible organisations for managing, updating, querying, or navigating in data. In this context, several roles are played by possibly different people: designer, administrator, and end-user. Hierarchical organisations are not flexible, and updating, querying and navigation are difficult to conciliate (see for instance the view update problem in data-bases). The literature shows that *Formal Concept Analysis* (*FCA*) is a good candidate for supporting querying and navigation. However, we feel it is not flexible enough as far as the description of data is concerned, and the literature on FCA insists more on analysing a given context than on managing contexts. In this article, we present an extension to FCA that allows for flexible descriptions, and we sketch an organisation that handles updating, querying, or navigation in data at the same time.

Given a *formal context* $(\mathcal{O}, \mathcal{A}, I)$, where $\mathcal{O}$ is a set of *objects*, $\mathcal{A}$ is a set of *attributes*, I is a relation between objects and attributes (i.e., a subset of $\mathcal{O} \times \mathcal{A}$), *Formal Concept Analysis* (FCA [Wil82,GW99a], Chapter 11 in [DP90a]) defines *concepts* as maximal sets of objects that share the same attributes. More formally, a concept is a pair (O, A) where O is a subset of $\mathcal{O}$, A is a subset of $\mathcal{A}$, such that the following relation holds:

$\sigma(O) = A$ and $\tau(A) = O$
where $\sigma : 2^{\mathcal{O}} \to 2^{\mathcal{A}}$ $\quad \sigma(O) := \{a \in \mathcal{A} \mid \forall o \in O : (o, a) \in I\}$
and $\tau : 2^{\mathcal{A}} \to 2^{\mathcal{O}}$ $\quad \tau(A) := \{o \in \mathcal{O} \mid \forall a \in A : (o, a) \in I\}$

The application σ returns for every set of objects the set of attributes that are shared by all these objects. The application τ returns for every set of attributes the set of objects that owns all these attributes. The O part of a concept is called

B. Ganter and G.W. Mineau (Eds.): ICCS 2000, LNAI 1867, pp. 371–385, 2000.
© Springer-Verlag Berlin Heidelberg 2000

its *extent*, and the A part is called its *intent*. The fundamental theorem of FCA is that the set of all concepts that can be built on a given formal context forms a complete lattice when it is ordered by set-inclusion of concept extensions. In fact, the pair $\sigma - \tau$ forms a Galois connection.

FCA has received attention for its application in many domains such as in software engineering [Sne98,Lin95,KS94]. The interest of FCA as a navigation tool in general has also been recognized [GMA93,Lin95,VW95].

The various application domains bring the need for more sophisticated formal contexts than the mere presence/absence of attributes. For instance, many application domains use numerical values (e.g., lengths, prices, ages), and the need to express negation and disjunction is often felt. In a much more specialized scope, it is imaginable to use the type of software components instead of attributes. Several enrichments to the attribute structure have been proposed: e.g., many valued attributes [GW99a], and first-order terms [CM98]. However, not a single extended FCA framework covers all the concrete domains, and can pretend covering all the concrete domains to come. So, we propose to construct a more general framework for concept analysis, Logical Concept Analysis (LCA), in which the logic of attributes becomes a parameter. This will allow for instantiating the general framework by merely filling in a dedicated logic.

In the rest of this article, we will refer to the original form of concept analysis by FCA or *standard* CA, while we will refer to the concept analysis as it is developed in this article by LCA or *generalized* CA. The term CA will be used to talk about both forms at once. Section 2 goes from standard CA (FCA) to its generalized form, i.e., LCA. Section 3 defines how a contextualized logic can be derived from a logical context. Section 4 studies the contribution of LCA compared with existing extensions of FCA. Section 5 explains how LCA can provide an interesting organization framework for information systems in which objects are described by logical formulas.

In this article, results are given without proofs. They can be found in [FR99].

2 From FCA to LCA

2.1 General Presentation

We start on reformulating standard CA so that it can be generalized more naturally. The first step of this reformulation consists in replacing context $(\mathcal{O}, \mathcal{A}, I)$ by $(\mathcal{O}, 2^{\mathcal{A}}, i)$, where $2^{\mathcal{A}}$ is the power-set of $\mathcal{A}$ and i is a mapping from $\mathcal{O}$ to $2^{\mathcal{A}}$ defined by $i(o) := \{a \in \mathcal{A} \mid (o, a) \in I\}$. No information is lost, nor added, and both representations are equivalent: $(o, a) \in I \iff i(o) \supseteq \{a\}$.

Then, if applications σ and τ are reformulated with mapping i rather than relation I, the following equalities are easily obtained:

$$(1) \quad \sigma(O) = \bigcap_{o \in O} i(o) \qquad\qquad (2) \quad \tau(A) = \{o \in \mathcal{O} \mid i(o) \supseteq A\}$$

The rest of FCA theory is kept unchanged. Now, by carefully studying proofs in FCA, and considering the connection between algebraic structures and lattices [DP90a], we observe that a necessary and sufficient condition to FCA is

that $\langle 2^{\mathcal{A}}; \supseteq \rangle$ be a lattice whose *supremum* (least upper bound) is $\cap$, and *infimum* (greatest lower bound) is $\cup$. Then, $2^{\mathcal{A}}$ can be considered as a logic where $\supseteq$ is the deduction relation, $\cap$ is disjunction, and $\cup$ is conjunction.

From this interpretation of FCA, it becomes natural to generalize it by replacing the power-set of the set of attributes $2^{\mathcal{A}}$ by an arbitrary set of formulas $\mathcal{L}$, to which are associated a deduction relation $\dot{\models}$, a disjunctive operation $\dot{\vee}$, and a conjunctive operation $\dot{\wedge}$ (dots on symbols are only aimed at differentiating them from those of the meta-language used in this document). So as to keep FCA results in its generalized form (LCA), it is necessary and sufficient that $\langle \mathcal{L}; \dot{\models} \rangle$ be a lattice whose supremum and infimum are respectively $\dot{\vee}$ and $\dot{\wedge}$.

Example 1. An example of a logic usable in LCA is propositional logic, $\mathcal{P}$. On the syntactic side, the set of propositions $\mathcal{P}$ contains atomic propositions (taken in a set $\mathcal{A}$), formulas 0 and 1, and is closed under binary connectors $\wedge$ and $\vee$ and unary connector $\neg$. On a semantical side, interpretations are subsets of the set of atomic propositions $\mathcal{A}$. Logic $\langle \mathcal{P}; \vee, \wedge, \models \rangle$ satisfies LCA conditions, because its semantics $\langle 2^{2^{\mathcal{A}}}; \subseteq \rangle$ is a lattice whose supremum is $\cup$, and infimum is $\cap$.

2.2 Context and Galois Connection

In this section, we apply the idea introduced in the last section to give a formulation of CA that is generalized to an almost arbitrary logic: Logical Concept Analysis (LCA). Transposition from FCA to LCA consists in reformulating each occurrence of I with i, and replacing $\supseteq$, $\cap$, and $\cup$ respectively by $\dot{\models}$, $\dot{\vee}$, and $\dot{\wedge}$. Proofs are obtained from the chapter 11 in [DP90a], and by applying the above transposition.

Definition 1 (context) *A* (formal) context *is a triple* $(\mathcal{O}, \mathcal{L}, i)$ *where:*

- $\mathcal{O}$ *is a finite set of objects,*
- $\langle \mathcal{L}; \dot{\models} \rangle$ *is a (possibly infinite) lattice of formulas, whose supremum is* $\dot{\vee}$*, and whose infimum is* $\dot{\wedge}$*;* $\mathcal{L}$ *denotes a logic whose deduction relation is* $\dot{\models}$*, and whose disjunctive and conjunctive operations are respectively* $\dot{\vee}$ *and* $\dot{\wedge}$*,*
- i *is a mapping from* $\mathcal{O}$ *to* $\mathcal{L}$ *that associates to each object a formula that describes the object.*

If $f \dot{\models} g$ and $g \dot{\models} f$, f and g are called logically equivalent; we will consider them as different representations of the same equivalence class, and in fact we will consider that elements of $\mathcal{L}$ are the equivalence classes. Also, operations $\dot{\vee}$ and $\dot{\wedge}$ can be either connectors (as in propositional logic $\mathcal{P}$), either algebraic operations (as in the so-called cube logic [CM98]). We just assume, for practical reasons, that operations $\dot{\models}$, $\dot{\wedge}$, and $\dot{\vee}$ are computable. A word of caution is necessary. The logic $\mathcal{L}$ is completely independent from the set of objects. If the logic $\mathcal{L}$ is rich enough to have quantifiers, the individuals upon which formulas are quantified can never be the objects of the formal context. For instance, if $i(o) \doteq \forall x.p(x)$, this tells nothing about the fact that all objects have property p. It only says that

if $i(o')\dot{=}p(1)$ then o can be considered as being more generic than o'. A good interpretation of a description like $i(o)\dot{=}\forall x.p(x)$ is to read it as a polymorphic type declaration. According to the Curry-Howard isomorphism, types can be considered as formulas.

FCA derives from a context two applications σ and τ that form a Galois connection. Here, the Galois connection is between sets of objects, and formulas. Its definition is obtained by transposing equalities (1) and (2):

Definition 2 *Let $(\mathcal{O},\mathcal{L},i)$ be a context, $O\subseteq\mathcal{O}$, and $f\in\mathcal{L}$.*
$\sigma : 2^{\mathcal{O}}\to\mathcal{L},\quad \sigma(O) := \dot{\bigvee}_{o\in O} i(o) \qquad \tau : \mathcal{L}\to 2^{\mathcal{O}},\quad \tau(f) := \{o\in\mathcal{O} \mid i(o)\dot{\models} f\}$

Lemma 1 *Let $(\mathcal{O},\mathcal{L},i)$ be a context, $O,O_j\subseteq\mathcal{O}$, and $f,f_j\in\mathcal{L}$ for all $j\in J$, where J is a finite set of indices.*

(i) $O\subseteq\tau(\sigma(O))$	*(i')* $\sigma(\tau(f))\dot{\models} f$
(ii) $O_1\subseteq O_2 \Longrightarrow \sigma(O_1)\dot{\models}\sigma(O_2)$	*(ii')* $f_1\dot{\models}f_2 \Longrightarrow \tau(f_1)\subseteq\tau(f_2)$
(iii) $\sigma(O)\dot{=}\sigma(\tau(\sigma(O)))$	*(iii')* $\tau(f)=\tau(\sigma(\tau(f)))$
(iv) $\sigma(\bigcup_{j\in J}O_j)\dot{=}\dot{\bigvee}_{j\in J}\sigma(O_j)$	*(iv')* $\tau(\dot{\bigwedge}_{j\in J}f_j)=\bigcap_{j\in J}\tau(f_j)$
(v) $\forall o\in\mathcal{O} : \sigma(\tau(i(o)))\dot{=}i(o)$	

Results expressed in this lemma are a transposition from those of standard CA (see Lemma 11.4 in [DP90b]), except Lemma 1.(v) whose demonstration uses the actual definitions of σ and τ.

Example 2. An example of a formal context will illustrate the rest of our development on LCA. Context K_{ex} is deliberately small and simple as it is aimed at illustrating theoretic notions, and not at showing a realistic application of LCA. The logic used in this context is propositional logic $\mathcal{P}$ with a set of atomic propositions $\mathcal{A}=\{a,b,c\}$. We define context K_{ex} by $(\mathcal{O}_{ex},\mathcal{P},i_{ex})$, where $\mathcal{O}_{ex}=\{x,y,z\}$, and where $i_{ex}=\{x\mapsto a, y\mapsto b, z\mapsto c\wedge(a\vee b)\}$.

2.3 Concepts

In this section, we show that using our definitions of a context and Galois connection σ-τ, we retrieve all existing results about concepts. First, we recall the definition of concepts (see Chapter 11 in [DP90a]), just adapting notations.

Definition 3 (concept) *In a context $(\mathcal{O},\mathcal{L},i)$, a* concept *is a pair $c=(O,f)$ where $O\subseteq\mathcal{O}$, and $f\in\mathcal{L}$, such that $\sigma(O)\dot{=}f$ and $\tau(f)=O$. The set of objects O is the concept* extent *(written ext(c)), whereas formula f is its* intent *(written int(c)).*

The main difference with standard CA is that the intent is now a formula of the logic $\mathcal{L}$. The set of all concepts that can be built in a context $(\mathcal{O},\mathcal{L},i)$ is denoted by $\mathcal{C}(\mathcal{O},\mathcal{L},i)$, and is partially ordered by $\leq^c$ defined as follows.

Definition 4 (order $\leq^c$) *Let (O_1,f_1) and (O_2,f_2) be in $\mathcal{C}(\mathcal{O},\mathcal{L},i)$,*
$(O_1,f_1)\leq^c(O_2,f_2)\Longleftrightarrow O_1\subseteq O_2$

This order is compatible with order on intents.

Proposition 1 $(O_1, f_1) \leq^c (O_2, f_2) \Longleftrightarrow (f_1 \dot{\models} f_2)$

As in FCA, Definitions 3 and 4 lead to the following *fundamental theorem.*

Theorem 1 *Let $(\mathcal{O}, \mathcal{L}, i)$ be a context, and let J be a set of indices. The ordered set $\langle \mathcal{C}(\mathcal{O}, \mathcal{L}, i); \leq^c \rangle$ is a finite lattice, whose supremum (least upper bound) and infimum (greatest lower bound) are as follows:*

$\bigvee^c_{j \in J}(O_j, f_j) =^c (\tau(\sigma(\bigcup_{j \in J} O_j)), \dot{\bigvee}_{j \in J} f_j)$

$\bigwedge^c_{j \in J}(O_j, f_j) =^c (\bigcap_{j \in J} O_j, \sigma(\tau(\dot{\bigwedge}_{j \in J} f_j)))$

Example 3. Figure 1.(a) represents the Hasse diagram of the concept lattice of context K_{ex} (introduced in Example 2). Concepts are represented by a number and a box containing their extent on the left, and their intent on the right. The higher concepts are placed in the diagram the greater they are for order $\leq^c$. It can be observed that the concept lattice is not isomorphic to the power-set lattice of objects $\langle 2^{\mathcal{O}}; \subseteq \rangle$. Indeed, set $\{x, y\}$ is not the extent of any concept, because $\tau(\sigma(\{x, y\})) = \tau(a \vee b) = \{x, y, z\}$.

2.4 Labelling of Concept Lattices

Similarly to standard CA, it is possible to label concept lattices with objects and formulas.

In the perspective of our application, the designer will choose a logic $\mathcal{L}$, an administrator will manage a formal context $(\mathcal{O}, \mathcal{L}, i)$, and a end-user will navigate, query, and consult/create/delete designated objects by using arbitrary formulas as labels. The end-user knows $\mathcal{L}$, but he does not necessarily know the formal context. For instance, he might be discovering it through navigation. In the following, we will ignore the possibility of labelling concepts with objects. It is fully described in [FR99].

Definition 5 *Let $(\mathcal{O}, \mathcal{L}, i)$ be a context. One defines a mapping μ labelling concepts with formulas:*

$\mu : \mathcal{L} \to \mathcal{C}(\mathcal{O}, \mathcal{L}, i) \qquad \mu(f) := (\tau(f), \sigma(\tau(f)))$.

Images of mapping μ are indeed concepts from Definition 3 of concepts, and from properties of applications σ and τ (Lemma 1). The next lemma gives interesting properties of these labellings of concept lattices.

Lemma 2 *Let $(\mathcal{O}, \mathcal{L}, i)$ be a context, and $o \in \mathcal{O}$, $f \in \mathcal{L}$, $c \in \mathcal{C}(\mathcal{O}, \mathcal{L}, i)$.*

(1) $c \leq^c \mu(f) \Longleftrightarrow int(c) \dot{\models} f$ *(2) μ is surjective*

(3) $\mu(int(c)) =^c c$ *(4)* $int(\mu(f)) \dot{\models} f$.

Lemma 2.(1) shows that $\mu(f)$ is the greatest concept whose intent logically entails f; and Lemma 2.(2) establishes that every concept is labelled at least

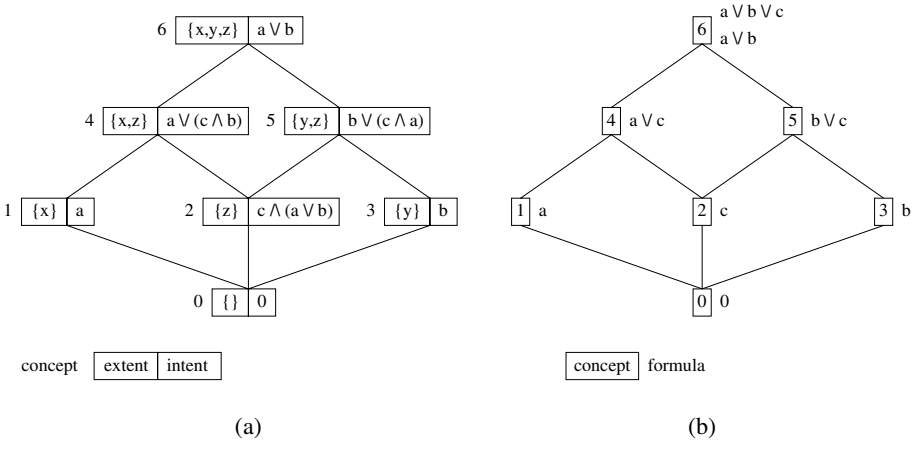


Fig. 1. The concept lattice of context K_{ex} (a) and its labelling (b).

by one formula. Regarding relation between concept intents and concept labels, Lemma 2.(3) shows that every concept is labelled by its intent; and Lemma 2.(4) adds that every formula labelling a concept is logically entailed by the concept intent. This means that concepts can be characterized by several formulas, the most precise one being its intent. This idea is developed in Section 3 and useful for the application sketched in Section 5.

Example 4. Figure 1.(b) represents the same concept lattice as Figure 1.(a), but it does not associate the same information to concepts. The number of each concept is reused in its box so as to identify it; formulas of the form $\bigvee A$ where $A \subseteq \mathcal{A}$ ($\bigvee \emptyset \equiv 0$) are placed on the right of their labelled concept. For instance, concept 1 is labelled by formula a (i.e., $\mu(a) =^c 1$). In Figure 1.(b) we have restricted labels to be formulas of the form $\bigvee A$, but it is only to have a finite number of labels that are not all in the formal context; recall that every formula in $\mathcal{P}$ labels something. It is important for the applications we have in mind (i.e., querying and navigation) not to restrict labels to be (sub)formulas of the logical context. This last point implies that some concepts are necessarily labelled by several formulas, because there is a finite number of concepts (this is indeed observed with concept 6).

3 Contextualized Logic

In a particular context, it is possible to order some properties, although they are not comparable by $\models$. For instance, if in some context every bird flies, then we can say that property "bird" contextually entails the property "fly", although we have not necessarily $bird \models fly$ in $\mathcal{L}$. We introduce a *contextualized deduction relation* as a generalization of implications between attributes, that can be used in standard CA for knowledge acquisition processes [GW99a,Sne98].

Definition 6 (contextualized deduction) *Let $K = (\mathcal{O}, \mathcal{L}, i)$ be a context, and $f, g \in \mathcal{L}$. One says that f* contextually entails *g in context K, which is noted $f \dot{\models}^K g$, if $\tau(f) \subseteq \tau(g)$, i.e., if every object that satisfies f also satisfies g.*

Relation $\dot{\models}^K$ is a preorder, and the associated equivalence relation is noted $\dot{\equiv}^K$.

Definition 7 (contextualized logic) *Let $K = (\mathcal{O}, \mathcal{L}, i)$ be a context. The term* contextualized logic *denotes the partially ordered set $\langle \mathcal{L}^K; \dot{\models}^K \rangle$ where*

$\mathcal{L}^K := \mathcal{L}/_{\dot{\equiv}^K}$

$\forall F, G \in \mathcal{L}^K : F \dot{\models}^K G \iff \exists f \in F, g \in G : f \dot{\models}^K g.$

Elements in $\mathcal{L}^K$ (equivalence classes of $\mathcal{L}$ modulo $\dot{\equiv}^K$) are called contextualized formulas*, and the class of a formula $f \in \mathcal{L}$ is denoted by $(f)^K$ (or more simply by f^K, if non-ambiguous).*

Lemma 3 *Let $K = (\mathcal{O}, \mathcal{L}, i)$ be a context, $f, g \in \mathcal{L}$, and $a, b \in \mathcal{C}(K)$.*

(1) $f \dot{\models} g \Longrightarrow f \dot{\models}^K g$ *(2)* $\forall o \in \mathcal{O} : i(o) \dot{\models} f \iff i(o) \dot{\models}^K f$

(3) $f \dot{\models}^K g \iff \mu(f) \leq^c \mu(g)$ *(4)* $a \leq^c b \iff int(a) \dot{\models}^K int(b)$

(5) $\sigma(\tau(f)) \dot{\equiv}^K f$

Lemma 3.(1) shows that contextualized deduction $\dot{\models}^K$ is only an enlargement of initial deduction $\dot{\models}$. Lemma 3.(2) shows that object properties are not altered by context: it does not add nor delete any property to objects. Lemmas 3.(3-4) show that μ and *int* are order-embeddings between $\langle \mathcal{L}; \dot{\models}^K \rangle$ and $\langle \mathcal{C}(K); \leq^c \rangle$. Lemma 3.(5) spots $\sigma(\tau(f))$ as the intent contextually equivalent to f.

A context plays the role of a theory extending the deduction relation and enabling new entailments. Contextualized logic can also be seen as a means for extracting knowledge from contexts. Two kinds of knowledge can thus be extracted: knowledge about context by deduction, and knowledge on the domain (from which context is extracted) by induction (e.g., generalizing from $bird \dot{\models}^K fly$).

Example 5. With morphism μ between contextualized deduction and order on concepts (Lemma 3.(3)), it is possible to use the labelled concept lattice (see Figure 1.(b)) to study contextualized logic in context K_{ex}. For instance, as concept 2 is smaller than concept 5 relation $c \models^{K_{ex}} b \vee c$ stands, which is already true in $\mathcal{P}$. More generally, it can be seen that all valid deductions in $\mathcal{P}$ are retrieved in contextualized deductions. Exam ination of the labelled concept lattice shows that the context adds new valid deductions between formulas: e.g., $c \models^{K_{ex}} a \vee b$, $a \vee b \vee c \equiv^{K_{ex}} a \vee b$.

We now consider connections between contextualized logic and concept lattice. The aim is to show that the concept lattice forms a new logic, derived from $\mathcal{L}$, and adapted to the context: the contextualized logic. For this, we use labelling mappings μ and *int* to establish a connection between formulas and concepts.

Theorem 2 *$\langle \mathcal{L}^K; \dot{\models}^K \rangle$ and $\langle \mathcal{C}(K); \leq^c \rangle$ are isomorphic, with μ^K as an isomorphism from contextualized formulas to concepts, and int^K as an isomorphism from concepts to contextualized formulas, defined by*

$\mu^K : \mathcal{L}^K \to \mathcal{C}(K) \qquad \mu^K(F) := \mu(f)$ *where* $f \in F$

$int^K : \mathcal{C}(K) \to \mathcal{L}^K \qquad int^K(c) := int(c)^K$

Algebraic structures $\langle \mathcal{C}(K); \vee^c, \wedge^c \rangle$ and $\langle \mathcal{L}^K; \dot{\vee}^K, \dot{\wedge}^K \rangle$ (where $\dot{\vee}^K, \dot{\wedge}^K$ denote supremum and infimum of $\langle \mathcal{L}^K; \dot{\models}^K \rangle$) are then isomorphic.

To summarize, there are three ways of considering a concept lattice, which is a sign of flexibility: (1) extents ($\tau(\sigma(2^{\mathcal{O}}))$) ordered by set inclusion ($\subseteq$), (2) intents ($\sigma(\tau(\mathcal{L}))$) ordered by logical deduction ($\dot{\models}$), (3) contextualized formulas ($\mathcal{L}^K$) ordered by contextualized deduction ($\dot{\models}^K$).

Finally, we consider connections between contextualized logic $\mathcal{L}^K$ and initial logic $\mathcal{L}$. The following relations stand between operations of both logic.

Theorem 3 *Let $f, g \in \mathcal{L}$,*

$f^K \dot{\wedge}^K g^K \dot{=}^K (f \dot{\wedge} g)^K$ and $f^K \dot{\vee}^K g^K \dot{=}^K \sigma(\tau(f) \cup \tau(g))^K$

It should be noticed that the mapping $(.)^K$ is a morphism from formulas to contextualized formulas for the conjunctive operation, but not for the disjunctive one. From this it follows that the concept lattice is not isomorphic to the initial logic $\mathcal{L}$. Therefore, some properties of this logic can be lost in concept lattice. For instance, this is the case of the distributive property in context K_{ex} (see Example 2). Indeed, propositional logic $\mathcal{P}$ is distributive, while the following counter-example can be found in the concept lattice of context K_{ex} (see Figure 1): $2 \wedge^c (1 \vee^c 3) =^c 2$, and $(2 \wedge^c 1) \vee^c (2 \wedge^c 3) =^c 0$.

A concept lattice inherits of all properties of $\mathcal{L}$ (modulo $=^c$) only if relation $f^K \dot{\vee}^K g^K \dot{=}^K (f \dot{\vee} g)^K$ holds. For this, it is sufficient that $\tau(f \dot{\vee} g) = \tau(f) \cup \tau(g)$ (because this implies $\sigma(\tau(f) \cup \tau(g)) \dot{=}^K \sigma(\tau(f \dot{\vee} g)) \dot{=}^K f \dot{\vee} g$). In more concrete terms, that amounts to laying down the following condition on formulas describing objects:

$$(3) \quad \forall o \in \mathcal{O} : i(o) \dot{\models} f \dot{\vee} g \iff i(o) \dot{\models} f \vee i(o) \dot{\models} g$$

This condition amounts to considering there is no form of disjunction in object descriptions. Back to context K_{ex}, it is observed that the description of object z satisfies proposition $a \vee b$ but satisfies neither a nor b, which falsifies condition (3). This explains why the distributive property is lost in the concept lattice.

4 Related Works

The first related work to consider is FCA itself since any given finite lattice is isomorphic to a concept lattice (see p. 27 in [GW99a]), and a formal context can be reconstructed from any concept lattice. This amounts to compiling $\mathcal{L}$ into a FCA formal context. But this compilation is not feasible because $\mathcal{L}$ is generally

an infinite and incomplete lattice. However, $\mathcal{O}$ is finite in all applications we have in mind (managing concrete data collections). So, a finite number of $\mathcal{L}$ formulas is actually used in any logical context. These formulas ordered by $\dot{\models}$ form a diagram which could be completed in a finite lattice, that could be compiled into a FCA formal context. This works for defining formal concepts, but it does not work for labelling since any formula of $\mathcal{L}$ can be a label. It does not work either for $\mathcal{L}^K$. It is also not a good idea to compile the finite number of actually used formulas because the context is bound to change as often as, say, the state of a file system changes. So, we believe that LCA can be considered as an extension to FCA because it permits to handle arbitrary descriptions when we need them.

The need for using as a formal context a more refined structure than the object-attribute relation has already been remarked by previous authors. Two directions have been followed for extending FCA. The first one is to replace attributes with more complex data. For instance, Chaudron and Maille replace attributes by first-order terms with free variables [CM98]. The underlying logic is that of unification and anti-unification [Plo70]. Kuznetsov replaces attributes by graphs that fit well his application in automated learning, and he mentions that such a construction can be done for all sorts of formal contexts [Kuz99]. In fact LCA gives a framework for this direction, which comprises labelling and contextualized logic. In his thesis [Mai99], Maille defines independently an extension of FCA that is similar to LCA. His Section 6.3 is parallel to our Sections 2.2 and 2.3. In fact, both works are inspired by Davey and Priestley's book [DP90a]. Maille does not describe labelling and contextualized logic, but he describes the concrete handling of a logic with constraints.

The second direction is to transform a more refined context into a classical formal context. The first paragraph of this section shows an example of this direction. This idea has been applied to an object-attribute-value relation [GW99b,Pre97]. The motivation is that multi-valued-contexts (in fact object-attribute-value relations) are widely used in the real world (e.g., in data-bases). There exists two methods for extracting a classical formal context from a multi-valued formal context.

One method uses *conceptual scales*. They are formal contexts whose objects are values and where attributes express some properties about values (e.g., 2 has the property ≤ 3). The concepts that follow from such formal contexts define a hierarchy among scale attributes. Then a mono-valued context can be derived by replacing values by scale attributes according to this hierarchy. Our generalized context $(\mathcal{O}, \mathcal{L}, i)$ can be seen as a multi-valued context where i is the only attribute and $\mathcal{L}$ is its domain of values. Then, one has to find a scale for retrieving the ordering $\dot{\models}$ on $\mathcal{L}$. But this is not feasible in general, as already explained in the first part of this section.

The other method is called *logical scaling*. It amounts to expressing attribute-value relations in a formalized language (e.g., SQL) using a finite (preferably small) collection of unary predicates. Names are given to all the predicates. Then a classical formal context can be constituted by replacing values associated to every object by the names of all the predicates that are satisfied by the values. As in our framework, logical scaling can be applied to an arbitrary logic. The big

difference is that we manage all formulas of the logic, which is in general infinite. It is of particular importance for applications we have in mind (see Section 5).

Some authors have also proposed to derive a logic from a standard formal context. E.g., Wille and Ganter propose to consider the formal context as defining prefered interpretations for the attributes considered as atomic propositions [GW99b]. A propositional formula is true (*a compound attribute is all-extensional* in Wille and Ganter's work) in the derived logic if it is true in all the prefered interpretations. This derived logic is in fact the contextualized logic of LCA when $\mathcal{L}$ is the logic of propositions $\mathcal{P}$, and every object description is a *complete conjunctive clause*.

5 Application to Logical Information Systems

We plan to use LCA in the design of flexible information systems. In these systems, LCA is not used only for structuring *a posteriori* a formal context, but rather for structuring *a priori* incoming data. These systems try to merge a querying facility with a navigation facility.

5.1 Querying and Navigating

Navigating usually means following links from places to places for reaching an objective. Links can be directories, URLs, file system links, etc. A fundamental concept in navigation is the *path*. A path is an ordered list of links that must be followed to find a place or an object. If the information system has a tree structure every object is accessible through only one path, and the game of navigating is to find this path. If the structure is a graph there may be several paths, and the game is to find one. Navigation requires erudition to know useful paths, judgement for recognizing that a path may lead to what is looked for, and some luck. Maintaining the information system structure, and keeping it navigable (i.e., ensuring objects are in the proper place), is difficult. However, it seems like the natural thing to do in information systems who have a structure *a priori* (e.g., given by a classification).

Querying usually means to elaborate a query that selects sufficiently few objects so that the relevant objects can be recognized among the answers. It supposes that every object have been indexed in some way. Indexes (and accordingly queries) can be sets of key-words, full-text words, etc. Just like navigation, querying requires erudition, judgement, and luck. Maintaining the information system structure is easier since the structure is essentially flat. Conversely, non-flat structures are not well-represented.

A concept lattice offers both navigation and querying but maintains the coherence automatically. Navigation amounts to following down-links in the concept lattice; concepts are considered as places, and objects that label a concept are considered to be *there* [GMA93,Lin95,VW95]. Querying amounts to conjunct the intent of the current path with some query, and to find which concept this conjunction labels. So doing, querying and navigation can be mixed in any order,

and the coherence between both is maintained by the coherence of intents and extents. We have applied this idea to the design of a conceptual file system/shell.

Compared with previous works, the main originalities of our proposal is to adopt the familiar interface of many Unix shells (i.e., commands *ls*, *cd*, *rm*, *mv* are transposed from a hierarchical setting to a conceptual setting), and to handle updates (e.g., via commands *rm* and *mv*). Typical applications are the management of personal directories, folders, or software development environments. We also envisage layman applications like catalogs or cookbooks (see [FR99] for an experiment with a Vietnamese cookbook). An important feature is also that concept lattices are used to give the semantics of the commands, but not in the implementation. Instead, it is the contextualized logic of Section 3 that is used.

A short presentation of the conceptual shell is given in the next section, but more details can be found in [FR00].

5.2 A Conceptual Shell

A conceptual shell can be described informally by comparison with a classical shell as follows: *files* become *objects*, *paths* become *logical formulas*, *directories* become *concepts* or *contextualized formulas*, the *root* becomes the concept $\top^c$ or formula $\dot{\top}$, and the *working directory* becomes a *working concept*.

The shell commands *cd* and *ls* are transposed in the conceptual shell for querying and navigating. Command *cd* merely maintains a *working query*, wq, labelling a *working concept* similar to a *working directory*, and command *ls* actually does the querying/navigation as follows. A command *ls* q returns a list of objects whose description is contextually equivalent to $q\dot{\wedge}wq$ (i.e., the objects labelling the concept $\mu(q\dot{\wedge}wq)$), plus a list of derived queries that possibly characterize strictly smaller but non empty concepts. The principle is that if a derived query q' is returned, the following holds:

$$(4) \quad \bot^c <^c \mu(q'\dot{\wedge}q\dot{\wedge}wq) <^c \mu(q\dot{\wedge}wq)$$

The fact that answers to queries may contain other queries that are relevant (i.e., satisfy the above inequation) is interesting in itself, and could be used in a natural language man-machine-interface: e.g., in a bookshop, *"Q: Do you have the Jungle Book? A: Yes we have, and also an illustrated version in the children department"*. Here, *Jungle Book* is the query, *Yes we have* means that there is an object with a contextually equivalent description (e.g., *Rudyard Kipling's best-known book*), and *illustrated version in the children department* means that there is also an object with a contextually strictly stronger description. According to the context, both *illustrated* and *children* can be derived queries. If the bookshop had only the illustrated version, the answer could have been *"Yes, in the children department"*.

In fact, derived queries act as sub-directories of the working directory. Just like the standard *ls* command, the conceptual logic *ls* command has an option *-r* which tells it to search "recursively" a directory and its sub-directories. However, in its details the conceptual query *ls -r q* is not recursive at all since it simply

returns the list of objects whose description satisfies $q \dot{\wedge} wq$ (i.e., the extent of the concept $\mu(q \dot{\wedge} wq)$).

Implementing a logical information system such as described above seems to require building the concept lattice, and being able to compute $<^c$ and μ. In fact, since the contextualized logic is isomorphic to the concept lattice by μ, operation $<^c$ on concepts can be replaced by operation $\dot{\models}^K$ on formulas (μ becomes unnecessary), whose computation can be reduced to the computation of τ (cf. Definition 6). Because of Definition 2, it is theref ore sufficient to memorize a Hasse subdiagram of the ordering of formulas ($\dot{\models}$), containing at least all the descriptions of objects ($i(\mathcal{O})$) to place them in the diagram, and possibly others like past queries.

This gives an approximation of the concept lattice whose size is in the order of the number of objects instead of an exponential of this number. The concept lattice can be seen as representing answers to all possible queries, while the Hasse diagram is a cache of answers to past queries. Our working hypothesis is that the end-user will navigate progressively, and will repetitively use the same idioms. One advantage of this structure is that it is not sensitive to the actual contents of the formal context; in particular it is not sensitive to context changes (i.e., *rm* and *mv*), but only to navigation steps.

All operations of the conceptual shell can be implemented using this data structure and the following primitive functions.

Extent of a query The extent of a query q is noted $\tau(q)$ and returns all objects whose description satisfies q. Operation τ is the same as in the Galois connection of LCA (cf. Definition 2). It is computed as the union of all objects accessible in the Hasse diagram starting from the node labelled q. It is similar to *ls -r q* in a hierarchy.

Objects of a query The objects of a query q, noted $t(q)$, are objects whose description is contextually equivalent to q (i.e., has the same extent): $t(q) = \{o \in \mathcal{O} \mid \tau(i(o)) = \tau(q)\}$. It is computed as the union of first objects accessible in the Hasse diagram starting from the node labelled q. It is similar to *ls q* in a hierarchy.

Derived queries $Inc(q) := \lceil \{x \in X \mid \emptyset \subsetneq \tau(x \dot{\wedge} q) \subsetneq \tau(q)\} \rceil$, where $\lceil E \rceil$ denotes the set of greatest elements of E according to the order $\dot{\models}$. Set X is chosen freely, but one shows that if every description of objects in $\tau(q)$ can be expressed as a conjunction of formulas in X, then for every object in $\tau(q)$ there is an x in $Inc(q)$ such that the object is in $\tau(x \dot{\wedge} q)$. This property ensures that every object is accessible through navigation. At any given moment in a navigation, the set of formulas of the diagram has this property.

None of these operations actually computes $\dot{\models}$ because it is entirely cached in the Hasse diagram. It is in command *cd q* that the position of $q \dot{\wedge} wq$ is searched in the diagram, or inserted if necessary. It is only there that $\dot{\models}$ is used.

6 Conclusion

We have shown how FCA can be reconstructed when the formal context is not restricted to be an object-attribute relation. We show how to use an almost arbitrary logic instead. A by-product of this reconstruction is the derivation of a contextualized logic that adds to the logic of the formal context deductions that are only valid in the formal context. The contextualized logic plays the same role as the concept lattice, and corresponds to attribute implication [GW99b].

We propose to exploit contextualized logic for navigating in a conceptual shell. The main interest of concept analysis in this application is in tightly combining querying and navigation: contextualized formulas are at the same time queries and places where it is possible to read and write. Note that in this application, it is important that no information is lost in converting an extended formal context into a standard one. On one hand, the operations of the conceptual shell make an intensive use of labelling functions and of the contextualized logic. So, it is important that concept analysis handles extended contexts directly. On the other hand, these operations never use the concept lattice as such. Its role is played by the contextualized logic. This makes it easy to update a formal context.

The logic used in the extended formal context is almost arbitrary, but new constraints apply on it when placed in the perspective of implementing a logical information system. Indeed, the primitive operations (e.g., deduction, conjunction, computation of the extent of a formula) must be tractable. For instance, we plan to apply LCA to the domain of software engineering. In this case, description logics (DL [DLNS96,DLNN97]), which are expressive though tractable logics can be used for version and configuration management [Zel98]; and the logic of type isomorphisms [Di 95] can be used for navigating in software component libraries.

At a more theoretical level, two directions for future works are to study the possibility of having relations in the extended formal context, and to study the possibility given by *views* (e.g., like in data-bases) to hide details. This would be extremely useful with extended formal contexts that could be overloaded by details that do not concern every user.

References

CM98. L. Chaudron and N. Maille. 1st order logic formal concept analysis: from logic programming to theory. *Computer and Information Science*, 13(3), 1998.

Di 95. R. Di Cosmo. Isomorphisms of Types: *from λ-calculus to information retrieval and language design*. Progress in theoretical computer science. Birkhä user, 1995.

DLNN97. F. M. Donini, M. Lenzerini, D. Nardi, and W. Nutt. The complexity of concept languages. *Information and Computation*, 134(1):1-58, 10 April 1997.

DLNS96. F.-M. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. Reasoning in description logics. In G. Brewka, editor, *Principles of Knowledge Representation*, pages 191-236. CSLI Publications, Stanford (CA), USA, 1996.

DP90a. B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, 1990.

DP90b. B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*, chapter 11 – Formal Concept Analysis, pages 221-236. Cambridge University Press, 1990.

FR99. S. Ferré and O. Ridoux. Une gé né ralisation logique de l'analyse de concepts logique. Technical Report RR-3820, Inria, Institut National de Recherche en Informatique et en Automatique, December 1999. An english version is available at http://www.irisa.fr/lande/ferre.

FR00. Sébastien Ferré and Olivier Ridoux. A file system based on concept analysis. In Yehoshua Sagiv, editor, *International Conference on Rules and Objects in Databases*, 2000. To appear.

GMA93. R. Godin, R. Missaoui, and A. April. Experimental comparison of navigation in a Galois lattice with conventional information retrieval methods. *International Journal of Man-Machine Studies*, 38(5):747-767, 1993.

GW99a. B. Ganter and R. Wille. Formal Concept Analysis – *Mathematical Foundations*. Springer, 1999.

GW99b. Bernhard Ganter and Rudolf Wille. Contextual attribute logic. *Lecture Notes in Computer Science*, 1640:377-388, July 1999.

KS94. M. Krone and G. Snelting. On the inference of configuration structures from source code. In *Proceedings of the 16th International Conference on Software Engineering*, pages 49-58. IEEE Computer Society Press, May 1994.

Kuz99. S. O. Kuznetsov. Learning of simple conceptual graphs from positive and negative examples. In Jan M. /.Z ytkow and Jan Rauch, editors, *Proceedings of the 3rd European Conference on Principles of Data Mining and Knowledge Discovery (PKDD-99)*, volume 1704 of LNAI, pages 384-391, Berlin, September 15-18 1999. Springer.

Lin95. C. Lindig. Concept-based component retrieval. In *IJCAI95 Workshop on Formal Approaches to the Reuse of Plans, Proofs, and Programs*, 1995.

Mai99. Nicolas Maille. *Modèle logico-algébrique pour la fusion symbolique et l'analyse formelle*. PhD thesis, Ecole Nationale Supérieure de l'Aéronautique et de l'Espace, November 1999.

Plo70. G. D. Plotkin. A note on inductive generalization. *Machine Intelligence*, Edinburgh Univ. Press, 5:153-163, 1970. Edinburgh Univ. Press, Edinburgh.

Pre97. Susanne Prediger. Logical scaling in formal concept analysis. *Lecture Notes in Computer Science*, 1257:332-341, August 1997.

Sne98. G. Snelting. Concept analysis – A new framework for program understanding. *ACM SIGPLAN Notices*, 33(7):1-10, July 1998.

VW95. F. Vogt and R. Wille. TOSCANA – a graphical tool for analyzing and exploring data. *Lecture Notes in Computer Science*, 894:226-??, 1995.

Wil82. Rudolf Wille. *Ordered Sets*, chapter Restructuring lattice theory: an approach based on hierarchies of concepts, pages 445-470. Reidel, Dordrecht Boston, 1982.

Zel98. A. Zeller. Versioning system models through description logic. *Lecture Notes in Computer Science*, 1439:127-132, 1998.

On the Treatment of Incomplete Knowledge in Formal Concept Analysis

Peter Burmeister and Richard Holzer

Department of Mathematics, AG 1, Darmstadt University of Technology,
Schloßgartenstr. 7, D-64289 Darmstadt, Germany
{burmeister,holzer}@mathematik.tu-darmstadt.de

Abstract. Some possible treatments of incomplete knowledge in conceptual data representation, data analysis and knowledge acquisition are presented. In particular, some ways of conceptual scalings as well as the role of the three-valued KLEENE-logic are briefly investigated. This logic is also one background in *attribute exploration*, a conceptual tool for knowledge acquisition. For this method a strategy is given to obtain as much of (attribute) implicational knowledge about a given "universe" as possible; and we show how to represent incomplete knowledge in order to be able to pin down the questions still to be answered in order to obtain complete knowledge in this situation.

Introduction

All our knowledge is incomplete. Therefore it is useful to have tools around to obtain as much of "certain" knowledge as possible in as many situations as possible. And in cases of incomplete knowledge it is desirable to be able to "measure" what is missing. In this note we want to present such tools for conceptual knowledge representation and conceptual knowledge acquisition.

In connection with the development of the program "ConImp" of basic Formal Concept Analysis,[1] which centers around attribute exploration — some special form of knowledge acquisition in connection with FCA[2] — quite soon the question arose, what one should do in the case that one cannot answer some of the questions of ConImp in this subprogram. First possible answers were tested in new versions of ConImp and finally published in [B91]. Recently, in connection with a doctoral thesis[3], much progress has been made in this direction. And the exploration algorithm of ConImp has been improved in such a way that now the expert can get maximal information about the valid attribute implications in the unknown context with respect to his knowledge.

In the first section we give some basic definitions of formal contexts, incomplete contexts and attribute implications. In the second section some propositions provide conditions to check whether an attribute implication is certainly

[1] In what follows Formal Concept Analysis will be abbreviated by FCA.
[2] See section 3 of this note for more details.
[3] See [H00].

B. Ganter and G.W. Mineau (Eds.): ICCS 2000, LNAI 1867, pp. 385–398, 2000.
© Springer-Verlag Berlin Heidelberg 2000

valid or possibly valid in a given incomplete context. The Kleene-Logic can be used to characterise the validity of attribute implications, if the premise and the conclusion are disjoint. In the third section we present an algorithm for attribute exploration with incomplete knowledge. At the end of the exploration the expert gets a list of implications which are certainly valid in the (unknown) universe, a list of counterexamples for implications that are certainly not valid in the universe, and a list of unknown implications. In this section an example is given to explain how the algorithm works.

1 Fundamental Concepts

1.1 Formal Contexts and the Meaning of Negative Entries

Before we go into more details let us recall some of the fundamental concepts of FCA. For more details see e.g. GANTER and WILLE [GW99]. The basic structure of FCA is that of a *formal context* (in what follows briefly called *context*) $\mathbb{K}$ i.e. of a triple $(\mathbb{K} =)\ (G, M, I)$, where G and M are sets, the elements of which are called *objects* and *attributes*, respectively, and where $I \subseteq G \times M$ is a binary relation between these sets, where $(g, m) \in I$ (or gIm) is read as:
"the object g has the attribute m" or "the attribute m applies to the object g".

Such formal contexts correspond to data tables with two possible values (e.g. "yes", usually represented by a cross "×" and "no", usually represented by a blanc entry, sometimes also by other symbols like "." for a better reading of the table),[4] where only one of them (e.g. "yes" or "×") is really relevant for the further evaluation. A cross in the line labeled by the name of the object g (i.e. in the *context row of* g) and in the column labeled by the name of the attribute m (i.e. in the *context column of* m) means that the object g has the attribute m, yet the meaning of a blanc or period in the line for g and the column of m is often not quite clear. Some of the possible meanings might be that[5]

(a) the object g *does not have* the attribute m (main case),
(b) it is irrelevant, whether or not the object g has the attribute m,
(c) at the time, when the table was built, it could not be decided, whether or not the object g has the attribute m.

In FCA all such kinds of "negative entries" are usually treated in the same way, and this treatment corresponds to case (a) from above. Yet except for the case when also the negation of m is contained in the set of attributes a "negative entry" does not contribute to the formation of the "concepts"[6]. For this reason such contexts are also called *one valued contexts*. In what follows we shall also refer to them as *complete contexts* in order to distinguish them from *incomplete contexts* as introduced below.

[4] In (possibly) incomplete contexts introduced below, we shall also use "+" for the positive and "−" for the negative entries. Tables 2, 4, 5, 8, and 10 therefore contain examples of such formal contexts.

[5] Cf. [B91] or [H00] for further possible meanings e.g. one connected with some probability measure.

[6] Cf. the end of the next subsection.

1.2 Incomplete Contexts

In this paper we shall mainly be concerned with case (c), which leads to the concept of an *incomplete context*, sometimes also called *three-valued context*, but this denotation is somewhat misleading (see the discussion below). By an *incomplete context* $\mathbb{K}_i := (G, M, \{+, ?, -\}, J)$ we understand a quadruple $(G, M, \{+, ?, -\}, J)$, where again G and M are the sets of objects and attributes under consideration, respectively, "+", "?" and "−" are the three possible entries of the corresponding table, and J is a ternary relation $J \subseteq G \times M \times \{+, ?, -\}$, which can also be considered as the graph of a mapping — also designated by J — $J : G \times M \to \{+, ?, -\}$ from the set $G \times M$ of all pairs of objects and attributes into the set $\{+, ?, -\}$ of possible values. The interpretation of the relation J is as follows:

$(g, m, +) \in J$**:** it is known that the object g has the attribute m,
$(g, m, -) \in J$**:** it is known that the object g does not have the attribute m,
$(g, m, ?) \in J$**:** it is unknown, whether or not the object g has the attribute m.

If an incomplete context $(G, M, \{+, ?, -\}, J)$ does not contain any question marks as entries, then we want to identify it with the *corresponding complete context* (G, M, I), where gIm, if and only if $(g, m, +) \in J$.

For sets $A \subseteq G$ of objects we define

$A^\square := \{m \in M \mid (g, m, +) \in J \text{ for all } g \in A\}$ to be the set of all attributes in M applying to all objects in A — the *certain intent* generated by A;
$A^\diamond := \{m \in M \mid ((g, m, +) \in J \text{ or } (g, m, ?) \in J) \text{ for all } g \in A\}$ to be the set of all attributes in M possibly applying to all objects in A — the *possible intent* generated by A; and dually:
$B^\square$**,** the *certain extent*, and $B^\diamond$, the *possible extent* generated by a subset B of M are defined in a dual way by exchanging above the roles of objects and attributes.

If the context $(G, M, \{+, ?, -\}, J)$ is "actually a complete one", i.e. if there are no $g \in G$ nor $m \in M$ such that $(g, m, ?) \in J$, then we have $A^\square = A^\diamond$ and $B^\square = B^\diamond$. For such complete contexts we define $A' := A^J := A^\square$ and $B' := B^J := B^\square$, and pairs (A, B) with $A = B' \subseteq G$ and $B = A' \subseteq M$ are called *(formal) concepts.*

Let $\mathbb{K}_i^j = (G, M, \{+, ?, -\}, J^j)$ $(j = 1, 2)$ be two incomplete contexts. We say that $\mathbb{K}_i^2$ is an *extension* of $\mathbb{K}_i^1$, if and only if for all $g \in G$ and for all $m \in M$

- $(g, m, +) \in J^1$ implies $(g, m, +) \in J^2$ and
- $(g, m, -) \in J^1$ implies $(g, m, -) \in J^2$.

In particular, $\mathbb{K}_i^2$ is called a *completion* of $\mathbb{K}_i^1$, if and only if $\mathbb{K}_i^2$ is an extension of $\mathbb{K}_i^1$ and it is "actually a complete context".

1.3 Attribute Implications

Besides contexts *attribute implications* are further important tools of conceptual data analysis. Let $\mathbb{K} = (G, M, I)$ be a complete context. For sets $A, B \subseteq M$ of

attributes one designates by $A \to B$ an *(attribute) implication* and one says that this implication *holds* (*is valid*) in $\mathbb{K}$, if and only if, for every object $g \in G$ to which all attributes from A apply, all attributes from B apply, too ($A \subseteq g'$ implies $B \subseteq g'$, for all $g \in G$).[7] One can interpret $A \to B$ as the propositional formula, $\bigwedge_{a \in A} a \Rightarrow \bigwedge_{b \in B} b$ where the elements of M are considered as propositional variables.[8] Such attribute implications contain a lot of information about the data, and the *concept lattice*[9] of a given complete context $\mathbb{K}$ is determined up to isomorphism by the knowledge of a set of attribute implications valid in $\mathbb{K}$ and "generating" the set of all attribute implications valid in $\mathbb{K}$ with respect to the ARMSTRONG *rules* (for arbitrary sets A, B, C and D of attributes):

$$\frac{}{A \to A}\ , \qquad \frac{A \to C}{A \cup B \to C}\ , \qquad \frac{A \to B \qquad B \cup C \to D}{A \cup C \to D}\ .$$

One should observe that we use implications in the material sense, and if A is a set of attributes which do not jointly apply to any object of $\mathbb{K}$, then A implies all other attributes, i.e. then $A \to M$ holds in $\mathbb{K}$.

In the case of an incomplete context $\mathbb{K}_i$ we assume the following

Semantical definition of validity and possibility ("our KRIPKE-semantics"): An attribute implication $A \to B$ (or any other propositional formula with the elements of M interpreted as propositional variables) *holds with certainty* (or *is certainly valid*) in $\mathbb{K}_i$, if and only if it holds in every completion of $\mathbb{K}_i$, and it is *satisfyable* (or *possibly valid*) in $\mathbb{K}_i$, if and only if it holds in at least one completion of $\mathbb{K}_i$.

1.4 Elimination of Question Marks by Implications

If one wants a set $\mathcal{I}$ of implications to be valid in some completions of an incomplete context $\mathbb{K}_i := (G, M, \{+, ?, -\}, J)$, then it may be possible to replace question marks in the context row of some $g \in G$ and in the context column of some $m \in M$ (i.e. for $(g, m, ?) \in J$)

- by a "+", if there is $A \to B$ in $\mathcal{I}$ such that $A \subseteq g^\square$ and $m \in B$;
- by a "−", if there is $A \to B$ in $\mathcal{I}$ such that $m \in A$, $A \setminus \{m\} \subseteq g^\square$ and $B \not\subseteq g^\diamond$.

As an example we consider the attribute implication {cubic} → {not prime} valid for the attributes of the incomplete context $\mathbb{K}_i$ shown in Table 1, which transforms $\mathbb{K}_i$ into the complete context $\mathbb{K}$ also shown in this table.[10]

[7] One also expresses this by saying "the intent of each object has to *respect* the implication $A \to B$".

[8] In the same way one could form other propositional (or first order) formulas and define their validity recursively in a similar way. Of particular interest are the *clauses* $\bigwedge_{a \in A} a \Rightarrow \bigvee_{b \in B} b$, the importance of which for FCA we cannot discuss here.

[9] I.e. the ordered set $\underline{\mathfrak{B}}(\mathbb{K}) := (\{(A, B) \mid (A, B) \text{ a concept of } \mathbb{K}\}, \leq)$, where, for concepts (A_1, B_1) and (A_2, B_2) of $\mathbb{K}$, one defines $(A_1, B_1) \leq (A_2, B_2)$ if and only if $A_1 \subseteq A_2$ (or, equivalently, $B_1 \supseteq B_2$). See [GW99] for more details.

[10] Observe that $2^{19} - 1$ is a (Mersennian) prime, and that {cubic} → {not prime} is valid for all natural numbers.

Table 1. Elimination of "?"s by the implication {cubic} → {not prime}

$\mathbb{K}_i$	cubic	not prime
19^9	+	?
$2^{19}-1$	?	−

↦

$\mathbb{K}$	cubic	not prime
19^9	+	+
$2^{19}-1$	−	−

2 Evaluation of Incomplete Contexts

2.1 Many-Valued Contexts

The way it is represented, an incomplete context just looks like an ordinary many-valued context $(G, M, (W_m)_{m\in M}, J)$ — then with $W_m = \{+, ?, -\}$ for all $m \in M$ —, where for each (many-valued) attribute $m \in M$ there exists a non-empty set W_m of possible attribute values; and in the corresponding table one has as entry in the row of g and in the column of m a value $J(g, m)$ from W_m.[11] In FCA many-valued contexts are usually converted by some *scaling* into a one-valued context, for which one then can use the usual methods of FCA — e.g. compute a basis of the set of attribute implications or draw the concept lattice — and in this way analyse the data. The idea of a conceptual scaling of a many-valued context consists in connecting with each many-valued attribute m a *scale context* $\mathbb{S}_m$ (for incomplete contexts this will usually be the same for all attributes). $\mathbb{S}_m$ has W_m — or a superset of it — as set of objects, and it has new attributes structuring W_m according to the ideas of the user. In the process of scaling each entry $J(g, m)$ is replaced — in the object row of g — by the object row of $J(g, m)$ in $\mathbb{S}_m$ (in the examples below we have regrouped the columns afterwards).[12] For incomplete contexts one could e.g. take a *nominal scaling* $\mathbb{S}_n$ or some of the other scales shown in Table 2.[13]

Table 2. Some conceivable scales to treat incomplete knowledge

$\mathbb{S}_n$	t	f	u
+	×	.	.
−	.	×	.
?	.	.	×

$\mathbb{S}_{pc}$	p	c
+	×	×
−	.	.
?	×	.

$\mathbb{S}_p$	p
+	×
−	.
?	×

$\mathbb{S}_c$	c
+	×
−	.
?	.

[11] Observe that this representation is similar to what happens in data bases. Observe, too, that in such many-valued tables incomplete knowledge is usually detected much earlier e.g. by a missing entry or by an additional value "unknown" (or a similar one).

[12] Again we refer to [GW99] for more details. Observe that the process of scaling is not unique and needs decisions of the user, as easily follows from our examples below.

[13] Here "t" stands for "true", "f" for "false", "u" for "unknown", "p" for "possible" and "c" for "certain".

The corresponding scalings of the incomplete context $\mathbb{K}_u$ given in Table 3 are shown in Tables 4, 5 and 6.[14]

Table 3. An incomplete context $\mathbb{K}_u$

$\mathbb{K}_u$	a	b	c
g	+	+	+
h	?	−	+
k	−	+	?

Table 4. The scaling of $\mathbb{K}_u$ by $\mathbb{S}_n$

$\mathbb{K}_{\mathbb{S}_n}$	a_t	b_t	c_t	a_f	b_f	c_f	a_u	b_u	c_u
g	×	×	×	.	.	.	.	.	.
h	.	.	×	.	×	.	×	.	.
k	.	×	.	×	.	.	.	.	×

Table 5. The scaling of $\mathbb{K}_u$ by $\mathbb{S}_{pc}$.

$\mathbb{K}_{\mathbb{S}_{pc}}$	a_p	b_p	c_p	a_c	b_c	c_c
g	×	×	×	×	×	×
h	×	.	×	.	.	×
k	.	×	×	.	×	.

Table 6. The scalings of $\mathbb{K}_u$ by $\mathbb{S}_p$ and $\mathbb{S}_c$, respectively

$\mathbb{K}_{\mathbb{S}_p}$	a_p	b_p	c_p
g	×	×	×
h	×	.	×
k	.	×	×

$\mathbb{K}_{\mathbb{S}_c}$	a_c	b_c	c_c
g	×	×	×
h	.	.	×
k	.	×	.

Although the scalings by $\mathbb{S}_n$ or $\mathbb{S}_{pc}$ allow to reconstruct the original incomplete context $\mathbb{K}_i$ — e.g. $(h, a, ?) \in J$ can be read from $(h, a_p) \in I_{pc}$ and $(h, a_c) \notin I_{pc}$ —, the implications certainly or possibly valid in $\mathbb{K}_i$ can only be read indirectly from $\mathbb{K}_{\mathbb{S}_n}$ or $\mathbb{K}_{\mathbb{S}_{pc}}$, respectively. E.g. $\mathbb{K}_{\mathbb{S}_{pc}}$ is the apposition[15] of $\mathbb{K}_{\mathbb{S}_p} =: (G, M, J_p)$ — which we call the *possibility context* — and $\mathbb{K}_{\mathbb{S}_c} =: (G, M, J_c)$ — which we call the *certainty context* of $\mathbb{K}_i = (G, M, \{+, ?, -\}, J)$ — and they allow to decide, whether a given attribute implication $A \to B$ is certainly valid or possibly valid in $\mathbb{K}_i$ (although this can also be decided directly in $\mathbb{K}_i$, as will be seen below):

[14] The notation $\mathbb{K}_{\mathbb{S}_*}$ shall indicate that this is a scaling of $\mathbb{K}_u$ w. r. t. the scale $\mathbb{S}_*$.

[15] The apposition of two contexts $\mathbb{K}_1 = (G, M, I)$ and $\mathbb{K}_2 = (H, N, J)$ with $G = H$ is the "disjoint union" of the two contexts (keeping the object set fixed): $\mathbb{K}_1|\mathbb{K}_2 := (G, M \dot{\cup} N, I \dot{\cup} J)$.

Proposition 1 (cv) *$A \to B$ is* certainly valid *in $\mathbb{K}_i$, if and only if, for every object $g \in G$, one has: if $A \subseteq g^{I_p}(= g^{\diamond})$, then $B \setminus A \subseteq g^{I_c}(= g^{\Box})$.*

(pv) *$A \to B$ is* possibly valid *in $\mathbb{K}_i$, if and only if, for every object $g \in G$, one has: if $A \subseteq g^{I_c}(= g^{\Box})$, then $B \subseteq g^{I_p}(= g^{\diamond})$.*

In [B91] we have considered in addition a *Boolean resolution* of an incomplete context $\mathbb{K}_i$.[16] This is not a scaling, yet its valid attribute implications are exactly the implications, which are certainly valid in $\mathbb{K}_i$.

2.2 The Role of the Three-Valued KLEENE-Logic

Having three values instead of the usual two values, and in particular their special meaning, suggests to use a three-valued logic. It has turned out, that none of the existing logics really fits "totally" to our semantical definition of validity and possibility. Yet one logic gets very close, as we shall see in Theorem 2, and that is the *three valued* KLEENE*-logic* for a propositional language with logical operators $\wedge$, $\vee$, $\to$ and $\neg$, the truth tables of which are given in Table 7.

Table 7. The truth tables of the three-valued KLEENE-logic

$\wedge$	$-$	$?$	$+$
$-$	$-$	$-$	$-$
$?$	$-$	$?$	$?$
$+$	$-$	$?$	$+$

$\vee$	$-$	$?$	$+$
$-$	$-$	$?$	$+$
$?$	$?$	$?$	$+$
$+$	$+$	$+$	$+$

$\to$	$-$	$?$	$+$
$-$	$+$	$+$	$+$
$?$	$?$	$?$	$+$
$+$	$-$	$?$	$+$

$\neg$	
$-$	$+$
$?$	$?$
$+$	$-$

That this logic does not always fit to our KRIPKE-semantics is seen from the fact that the implication $\{a\} \to \{a\}$ is valid in every completion of any incomplete context having a as attribute, yet one also easily realizes that it will get the truth value "?" whenever a gets this value. However, one has the following result, which is very helpful to check the validity and possibility of attribute implications with disjoint sets for premise and conclusion:

Theorem 2 *Let $\mathbb{K}_i = (G, M, \{+, ?, -\}, J)$ be an incomplete context, and let Φ be a propositional formula with propositional variables in M. Then the three-valued* KLEENE*-logic computes the truth value of Φ w. r. t. $\mathbb{K}_i$ correctly in the sense of our* KRIPKE*-semantics, whenever every propositional variable (i.e. attribute from M) occurs within Φ at most once.*

Corollary 3 *Let $A \to B$ be an attribute implication w. r. t. some incomplete context $\mathbb{K}_i$. Then the three-valued* KLEENE*-logic computes its truth value correctly, when it is applied to $A \to (B \setminus A)$ instead of $A \to B$.*

[16] However, in connection with attribute exploration the corresponding algorithm was not given correctly, since the already accepted implications have not been considered.

Since the implications $A \to B$ and $A \to (B \setminus A)$ always contain the same information, Corollary 3 shows a way, how attribute implications can be evaluated syntactically w. r. t. any incomplete context.

It has turned out that despite the modal formulation of our KRIPKE-semantics in connection with incomplete contexts the usual types of modal logics are not appropriate. They may be useful for describing the validity of a propositional formula for a *single object*, but if we consider the whole context then the above KRIPKE-semantics seems to fit better.

3 Attribute Exploration

3.1 The General Idea of Attribute Exploration

One of the main situations, where one often has to deal with incomplete knowledge, is the method of *attribute exploration* in FCA[17], which is a special kind of knowledge acquisition. Its main intention is to obtain complete knowledge about the valid attribute implications in a *universe*, i.e. in a formal context $\mathbb{K}_{\mathfrak{U}} = (G_{\mathfrak{U}}, M_{\mathfrak{U}}, I_{\mathfrak{U}})$, where the set $G_{\mathfrak{U}}$ of objects is usually too large to allow the context to be represented or even to be known in all details, or it may only be known implicitly: If some special question about it arises, then one hopes to be able to answer it or at least that a group of experts may agree on an answer. The set $M_{\mathfrak{U}}$ of attributes is a fixed *finite* choice of most interest among all the attributes conceivable for $G_{\mathfrak{U}}$. The idea of the procedure can be sketched as follows:

A computer program, which can be fed in advance with some "background knowledge", systematically computes in some prescribed order proposals of attribute implications which might be valid in the given universe according to the input given so far.[18] The user (e.g. an expert or a group of experts) then has to accept the proposal or to refute it by a counterexample, i.e. by an object from the universe. If the expert can answer each question completely, she has produced at the end a complete subcontext $\mathbb{K}^* = (G^*, M_{\mathfrak{U}}, I_{\mathfrak{U}} \cap (G_{\mathfrak{U}} \times M_{\mathfrak{U}}))$ of $\mathbb{K}_{\mathfrak{U}}$, where $G^* \subseteq G_{\mathfrak{U}}$ contains for each attribute implication not holding in $\mathbb{K}_{\mathfrak{U}}$ a counterexample; and she has produced a list of implications, from which every implication valid in the universe can be derived by the ARMSTRONG rules listed in section 1.3. In what follows we mainly want to discuss how to get optimal information — and how this looks like —, when one cannot answer all questions.

Although most attribute explorations carried through so far have mathematical universes, this is not necessary. For example one exploration known to us has been carried through about some properties of pieces of music[19]; another one on some properties of fairy tales, and one about animals.[20] We shall use

[17] Cf. also [GW99], pp. 85ff or [B91].
[18] They are possibly valid in the (possibly incomplete) context computed so far.
[19] See [W89].
[20] See [Wo93].

below for an example some properties of natural numbers, since we hope that here everybody will be able to check the details.

3.2 An Algorithm of Attribute Exploration

Let $\mathbb{K}_{\mathfrak{U}} = (G_{\mathfrak{U}}, M_{\mathfrak{U}}, I_{\mathfrak{U}})$ be an (unknown) universe. At the beginning of the exploration algorithm the user has to enter the attribute set $M_{\mathfrak{U}}$, and he may enter a set $\mathcal{H}$ of background implications and a *start context* $\mathbb{K}_1$ containing some objects of the universe. It is also possible that $\mathbb{K}_1$ contains questionmarks if the user does not always know, which attributes the objects have.

In the $j-th$ step, for $j = 1, 2, 3, \ldots$, of the algorithm the set P_j contains the implications that are so far accepted as valid. At the beginning this set is initialised with the empty set: $P_1 := \emptyset$. In the step j the program chooses an attribute implication $A \to B$ that might be valid in the universe $\mathbb{K}_{\mathfrak{U}}$. Here the set A is a minimal set (with respect to the order "$\subseteq$") respecting P_j, and $B := \{m \in M | A \to m \text{ is satisfyable in } \mathbb{K}_j\} \neq A$. If the implication $A \to B$ is derivable from $P_j \cup \mathcal{H}$ by application of the ARMSTRONG rules, then the implication is accepted automatically[21]: $P_{j+1} := P_j \cup \{A \to B\}$. Otherwise the program asks the user for the validity of this implication in the chosen universe. If the user gives the answer "yes" then this implication is added to the actual set of accepted implications: $P_{j+1} := P_j \cup \{A \to B\}$. If the user gives the answer "no" then she has to enter a counterexample $g \in G$ for this implication. The row with this counterexample and all its entries with respect to the attributes is added to the actual context (table) thus obtaining : $\mathbb{K}_{j+1} =: \mathbb{K}_j + g$. If the user gives the answer "unknown" then the program asks, for which attributes $b \in B$ of the conclusion the validity of the implication $A \to b$ is unknown. Let

$$Z_j := \{b \in B | A \to b \text{ is unknown } \}.$$

For each attribute $b \in Z_j$ a fictitious counterexample $g_{A,b}$ is added to the actual context: $\mathbb{K}_{j+1} := \mathbb{K}_j + \{g_{A,b} | b \in Z_j\}$, where the context row of $g_{A,b}$ is the smallest row (w. r. t. the information order $+ > ? < -$), which is a counterexample for the implication $A \to b$:

	A	b	rest
$g_{A,b}$	+++	−	???

The implication $A \to B \setminus Z_j$ is added to the actual set of implications if $B \setminus Z_j \neq A$. After each step of the exploration algorithm the program eliminates question marks in "real" (i.e. not fictitious) examples using the accepted implications and the background implications (see section 1.4), but one will loose information, if this is also done in connection with fictitious ones (this could be done only at the end of the algorithm). The algorithm ends in step n, if for all sets A which respect P_n one has $A = \{m \in M | A \to m \text{ is satisfyable in } \mathbb{K}_n\}$.

[21] If one deals with other background knowledge than just implications — e.g. pairs of attributes negating each other — other rules might be necessary, too — like "GANTER's exhaustion rule" (cf. e.g. [G00]).

After the exploration has been finished, often some superfluous fictitious objects must be removed from the context: If an unknown implication $A \to b$ follows from the accepted implications P_n, then the fictitious object $g_{A,b}$ must be removed from the context. If there is a normal (i.e. not fictitious) object g which is a counterexample of an unknown implication $A \to b$, then the fictitious object $g_{A,b}$ is superfluous and can be removed from the context, because g is then also a counterexample against every implication which is not valid for $g_{A,b}$.

Let $\mathbb{K}^*$ — with object set G^* — be the context resulting from the last context $\mathbb{K}_n$ after removing the superfluous objects. Assume that the the user knew the set, say G_w, of those fictitious objects in G^*, for which the corresponding implication is actually true in the universe $\mathbb{K}_\mathfrak{U}$.

Fact 1: *Let W be the subset $G^* \setminus G_w$ obtained from G^* by removing the fictitious objects in G_w. Then the restriction $\mathbb{K}^*|_W$ of the context $\mathbb{K}^*$ contains a complete system of counterexamples for the implications which are not valid in the universe. The implications* satisfiable *in $\mathbb{K}^*|_W$ are exactly the valid implications of the universe.*

Fact 2: *Let U be the union of the set P_n of accepted implications with the set of those unknown implications encoded by fictitious examples in G_w. Then U is a generating set of implications for the valid implications in the universe: The implications that are derivable from U (with the* Armstrong *rules) are exactly the valid implications of the universe.*

Fact 3: *If there exists a question mark in the context row of a normal (i.e. not fictitious) object g in $\mathbb{K}^*$, then this object can be removed without changing the set of the satisfiable implications.* Namely, for each implication $A \to B$, for which such a g is a counterexample, there exists another object which is also a counterexample. — But the other object might be a fictitious one, and then we would loose some information by removing g, because with the information contained in the context without the object g we possibly can no longer decide whether or not such an implication $A \to B$ is true in the universe $\mathbb{K}_\mathfrak{U}$. So in some cases it is better to keep such objects with question marks.

3.3 An Example of Attribute Exploration

As an example we take as universe $\mathbb{K}_\mathfrak{U}$ the context with the set $\mathbb{N} = \{1, 2, 3, \dots\}$ of all positive natural numbers as set of objects, and with the attributes

even, odd, prime, square, cubic and sum2square,

where a number $n \in \mathbb{N}$ has the attribute *sum2square* if and only if there exist two numbers $a, b \in \mathbb{N}$ with $n = a^2 + b^2$.

Before we begin with the exploration, we enter the following three implications as background knowledge.

$$\{even, odd\} \to \{prime, square, cubic, sum2square\},$$
$$\{prime, cubic\} \to \{even, odd, square, sum2square\},$$
$$\{prime, square\} \to \{even, odd, cubic, sum2square\}$$

Namely, they are valid in $\mathbb{N}$, since there does not exist any number which satisfies the premise of any one of these implications. Thus the set $\mathcal{H}$ consists of these implications. At the beginning of the exploration we have to choose a context,

which we would like to start with. In our example this will be the context that contains as objects the numbers less than ten:

Table 8. The (still complete) start context for the exploration

$\mathbb{K}_1$	even	odd	prime	square	cubic	sum2square
1	−	+	−	+	+	−
2	+	−	+	−	−	+
3	−	+	+	−	−	−
4	+	−	−	+	−	−
5	−	+	+	−	−	+
6	+	−	−	−	−	−
7	−	+	+	−	−	−
8	+	−	−	−	+	+
9	−	+	−	+	−	−

Now the exploration program asks for the validity of implications in the universe $\mathbb{K}_{\mathfrak{U}}$:
First question:[22] $\{cubic, sum2square\} \to \{even\}$?
This implication does not hold in $\mathbb{N}$ and we find a *counterexample:* $125 = 5^3 = 10^2 + 5^2$.
We add this new object to the context and enter the context row:

	even	odd	prime	square	cubic	sum2square
125	−	+	−	−	+	+

Second question: $\{square, sum2square\} \to \{even, odd, prime, cubic\}$?
Counterexample: 100 $(= 10^2 = 8^2 + 6^2)$
The exploration continues and after step 12 the algorithm ends. During the exploration the three background implications of the set $\mathcal{H}$ are accepted automatically and the 11th question of the program is answered with "yes":
$\{even, prime\} \to \{sum2square\}$?
The number 2 is the only even prime and $2 = 1^2 + 1^2$ is the sum of two squares, so this implication is valid.
Without an appropriate computer program it is difficult to find a cubic square that is a sum of two squares but not even, so the following two questions are answered with "unknown":
6th question: $\{square, cubic, sum2square\} \to \{even\}$
9th question: $\{odd, square, cubic, sum2square\} \to \{even, prime\}$
These unknown implications lead to three fictitious counterexamples:
$?g_1 := g_{\{square,cubic,sum2square\},even}$
$?g_2 := g_{\{odd,square,cubic,sum2square\},even}$

[22] Since every implication $A \to A$ is trivially true, we only list the implications in the form $A \to (B \setminus A)$.

$?g_3 := g_{\{odd,square,cubic,sum2square\},prime}$
The context at the end of the exploration is given in Table 9.

Table 9. The incomplete context with fictitious examples resulting from the attribute exploration

$\mathbb{K}_{13}$	even	odd	prime	square	cubic	sum2square
1	−	+	−	+	+	−
2	+	−	+	−	−	+
3	−	+	+	−	−	−
4	+	−	−	+	−	−
5	−	+	+	−	−	+
6	+	−	−	−	−	−
7	−	+	+	−	−	−
8	+	−	−	−	+	+
9	−	+	−	+	−	−
125	−	+	−	−	+	+
100	+	−	−	+	−	+
25	−	+	−	+	−	+
1000000	+	−	−	+	+	+
64	+	−	−	+	+	−
$?g_1$	−	?	?	+	+	+
$?g_2$	−	+	?	+	+	+
$?g_3$	?	+	−	+	+	+

We have four implications accepted as valid:

$P_{13} = \{$
$\{prime, cubic\} \to \{even, odd, square, sum2square\}$
$\{prime, square\} \to \{even, odd, cubic, sum2square\}$
$\{even, prime\} \to \{sum2square\}$
$\{even, odd\} \to \{prime, square, cubic, sum2square\}$
$\}$

In this special example we can find with a computer an odd cubic square that is a sum of two squares: $15625 = 125^2 = 25^3 = 35^2 + 120^2$. This number is a counterexample for all three unknown implications. For each fictitious object g the context row of 15625 is a completion of the context row of g. Therefore we can replace the fictitious objects by 15625 to get a complete context which contains a complete system of counterexamples against the implications which are not valid in $\mathbb{N}$ w.r.t. our 6 chosen attributes.[23]

[23] If we had used the additional attributes *sum2primes* (= "is the sum of two primes") and *sum2even* (= "is the sum of two even numbers"), then we could not have given a complete context at the end without answering the still unproven GOLDBACH-*conjecture* of number theory stating: "Every even number greater than 2 is the sum

Table 10. The complete context resulting from the attribute exploration

	even	odd	prime	square	cubic	sum2square
1	−	+	−	+	+	−
2	+	−	+	−	−	+
3	−	+	+	−	−	−
4	+	−	−	+	−	−
5	−	+	+	−	−	+
6	+	−	−	−	−	−
7	−	+	+	−	−	−
8	+	−	−	−	+	+
9	−	+	−	+	−	−
125	−	+	−	−	+	+
100	+	−	−	+	−	+
25	−	+	−	+	−	+
1000000	+	−	−	+	+	+
15625	−	+	−	+	+	+
64	+	−	−	+	+	−

The four accepted implications form a generating system for the implications which are valid in our chosen universe because the unknown implications are not valid. The number 15625 is a counterexample for all three of the unknown implications, and the accepted list is also the final one in this case: $U = P_{13}$.[24]

Starting from version 4.14 of the program ConImp mentioned in the introduction such an algorithm of attribute exploration is implemented, which "behaves" almost as described above.[25]

4 Conclusion

Since all certain or possible logical information about possibly incomplete contexts can be formulated by sets of clauses or attribute implications with disjoint premise and conclusion, the three-valued Kleene-logic is an adequate tool to evaluate (such) logical statements.

of two primes". And in a similar way this method will often run into — old or new — open problems in the field of investigation.

[24] Cf. Fact 2, above, for the notation.

[25] If one runs with version 4.14 or with an earlier version of ConImp into a proposal of an implication $A \to B$, which one can neither prove nor disprove, one should not "accept it as uncertain" by the corresponding option of the program, but one should produce by oneself the (list of) fictitious objects for those attributes m in the conclusion, for which $A \to \{m\}$ can neither be proved nor disproved. ConImp will then automatically ask for $A \to B \setminus \{m\}$, if $B \setminus \{m\} \setminus A \neq \emptyset$, etc. Yet in versions earlier than 4.14 question marks will be changed to crosses — if possible — also in fictitious objects.

Attribute exploration is a very useful tool to get knowledge about the dependencies between attributes of an "unknown" context (universe). Our interactive algorithm only asks questions which are necessary to compute the set of all attribute implications which are valid in the context. Even if the expert can not answer all questions, he gets maximal information about the valid implications with respect to his knowledge and about what has been left open.

References

B86/99. P.Burmeister. (assisted by A.Rust, P.Scheich and N.Newrly). *ConImp — A program on formal concept analysis of one-valued contexts* (running under MS-DOS). Darmstadt University of Technology, 1986/99.

B91. P.Burmeister. *Merkmalimplikationen bei unvollständigem Wissen.* In: W.Lex (ed.): *Proceedings: Arbeitstagung Begriffsanalyse und Künstliche Intelligenz (Clausthal-Zellerfeld 6. - 8. 10. 1988)*; Technische Universität Clausthal, 1991, pp. 15–46.

B00. P.Burmeister. *ConImp — Ein Programm zur Formalen Begriffsanalyse.* In: G.Stumme, R.Wille (eds.) *Begriffliche Wissensverarbeitung: Methoden und Anwendungen*, Springer, 2000. See also
Formal Concept Analysis with ConImp: Introduction to the Basic Features; at http://www.mathematik.tu-darmstadt.de/∼burmeister

G00. B.Ganter. *Begriffe und Implikationen.* In: G.Stumme, R.Wille (eds.) *Begriffliche Wissensverarbeitung: Methoden und Anwendungen*, Springer, 2000.

GW99. B.Ganter, R.Wille. *Formal Concept Analysis — Mathematical Foundations.* Springer, 1999.

H00. R.Holzer. *Begriffliche Methoden bei der Behandlung unvollständigen Wissens.* Department of Mathematics, Darmstadt University of Technology: Doctoral thesis, in preparation.

W89. R.Wille. *Knowledge acquisition by methods of formal concept analysis.* In: E.Diday (ed.): *Data analysis, learning symbolic and numeric knowledge.* Nova Science Publishers, New York – Budapest, 1989, pp. 365–380.

Wo93. K.E.Wolff. *A first course in formal concept analysis.* In: F.Faulbaum (ed.): *SoftStat'93, Advances in Statistical Software 4,* pp. 429–438.

Logic-Based Networks: Concept Graphs and Conceptual Structures

Peter W. Eklund

Knowledge, Visualisation and Ordering Laboratory
School of Information Technology, Griffith University
Southport, Queensland 9726, AUSTRALIA
p.eklund@gu.edu.au

Abstract. Logic-based networks are semantic networks that support reasoning capabilities. In this paper, knowledge processing within logic-based networks is viewed as three stages. The first stage involves the formation of concepts and relations: the basic primitives with which we wish to formulate knowledge. The second stage involves the formation of well-formed formulas that express knowledge about the primitive concepts and relations once isolated. The final stage involves efficiently processing the wffs to the desired end. Our research involves each of these steps as they relate to Sowa's *conceptual structures* and Wille's *concept lattices*. Formal Concept Analysis gives us a capability to perform concept formation via symbolic machine learning. Concept(ual) Graphs provide a means to describe relational properties between primitive concept and relation types. Finally, techniques from other areas of computer science are required to compute logic-based networks efficiently. This paper illustrates the three stages of knowledge processing in practical terms using examples from our research.

1 Introduction

The research reported in this paper investigates the viability of conceptual graphs (CGs [24,25]) as a knowledge representation [20,10,12,18,19], its effectiveness as a graphical aid to cognition [10,11,2,3], and how a formal theory of order can be used to compute conceptual structures in an efficient way [9,7].

This paper is structured in five sections. I demonstrate the use of formal concept analysis (FCA) by way of a project for developing a medical retrieval system in Section 2. There (and again in Section 3) I show how FCA can be used to efficiently visualise and filter a document collection [6]. The approach re-uses a medical thesaurus as a collection of attributes extracted from a document corpus. Each document is in turn treated as an object. In small attribute collections this approach has merit [4] but it becomes visually intractable when large numbers of attributes are used. Section 2 shows how we can relieve visual complexity and preserve the efficiency of the lattice-based approach. This work, extended

B. Ganter and G.W. Mineau (Eds.): ICCS 2000, LNAI 1867, pp. 399–420, 2000.
© Springer-Verlag Berlin Heidelberg 2000

```
{{Problem List }
     { 0. Small cell ca left lung, 8 cycles chemotherapy 1990, plus
       radiotherapy 1991. 1. Pulmonary embolus. 2. Glaucoma.
       3. Peptic ulcer. 4. Cholecystectomy. 5. Appendicectomy.
       6. Oophorectomy. 7. Right sided pneumonia and neutropaenia. }}
  {{Discharge Treatment}
      { Coloxyl with senna 2 tabs bd, Ventolin 90 sec 4 hrly prn,
       Ranitidine 300mg bd, Mylanta 20 mls tds prn,
       Nifedipine 10mg tds, Panadeine forte 1-2 tabs 4 hrly prn,
       Dipiverine Hydrochloride .1 2 dps bd both eyes. }}
    {{ Information to Patient }
      { Patient aware of diagnosis and limited prognosis.
        Knows to present to LMO or RAH with any problems. }}
  {{Summary of Admission}
      { 68 year old woman, well known to S2. Discharged one week ago.
        Day after discharge, developed increasing SOB with
        yellow/white sputum production. Felt unwell, but denies
        rigors or chills. Using Ventolin regularly with no
        improvement. Transferred by ambulance to RAH. }}
  {{Examination}
      { ; mildly tachypnoeac, RR 30, not cyanosed, looks unwell,
        febrile 38.9, dehydrated, HR 120/reg, BP 140/90, JVP NR,
        HS dual + nil, no ankle swelling, peripheral pulses all
        present, TML, PN dull left base, BS vesicular, reduced
        at left base, crackles right anterior chest. Abdominal
        and neurological examination unremarkable. }}
  {{Investigation} { }}
     {{ Progress }
      { a steady improvement made. Freely mobile around the ward
        without oxygen on discharge. }}
  {{Follow Up }
     { Chest Clinic/Dr. Holmes LMO to perform MBA20 prior to
      OPD appt. }} {{ Copies }{ lmo,file,ur. }}
```

Fig. 1. A typical medical discharge document with the sub-headings.

in Section 3 to filter email [7], is more related to FCA, but has its analog in concept(ual) graphs[1].

In Section 4, I illustrate the practical outcomes of the work of Prediger and Wille [22] on concept graphs. This theoretical framework was coded as software by Bernd Groh and reported in [15]. Groh's software demonstrates how Wille's Power Context Family (PCF) can be interpreted as concept(ual) graphs and provides us with an application that integrates FCA, relational databases and a query interface using conceptual structures[2]. Finally, in Section 5, a Web-

[1] A more complete version of this relationship is reported in these proceedings, R. Cole and G. Stumme, "CEM – A Conceptual email Manager" in this volume.

[2] see P. Eklund, B. Groh, G. Stumme and R. Wille, "A Contextual-Logic Extension of TOSCANA"— elsewhere in this volume.

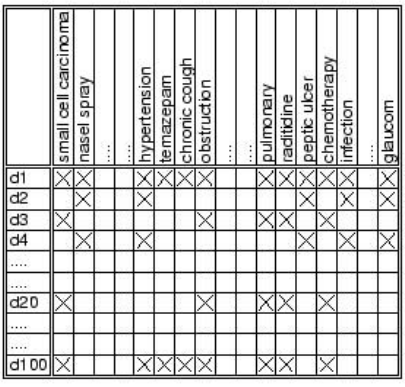


Fig. 2. An abbreviated view of the document context, the objects are document numbers written down the rows and attributes are stemmed MeSH terms written across the columns. In this example there are 100 documents in the context. In the actual context there are slightly less than 4,000 documents and more than 150,000 MeSH terms.

accessible knowledge representation toolkit is profiled [18,19]. Our claim is that WebKB was the world's first precision Web-based information retrieval engine using conceptual graphs. Many of the lessons learned in its development are relevant to emerging Web standards such as XML/RDF.

2 An Application of Formal Concept Analysis

The first stage of knowledge processing within logic-based networks involves the formation of concepts, i.e. the basic primitives such as objects, attributes, relations, and concepts to formulate knowledge. For the formalisation of those primitives, we adopted the approach of *Formal Concept Analysis* (FCA). FCA has been developed during the last twenty years and has already successfully applied to knowledge processing [27]. The Mathematics of FCA has been described in Ganter and Wille [14].

Our first experience using FCA was to develop a medical retrieval system in cooperation with the Royal Adelaide Hospital. Using 4,000 patient discharge documents, one of which is shown below in Fig 1, we found a number of electronic medical thesaurus and built a data context with documents as objects and medical terms as attributes: shown in Fig. 2. The project can therefore be considered as a logic-based network where the concepts are the medical terms and the relations and occurrences of those terms.

Term identification and extraction from texts is a difficult problem. One of the first tasks is to semi-automate term identification. Fig. 4 shows an interface written for the clinician to located and extract MeSH [1] terms from the documents. The top left frame of Fig. 4 shows a document, the interface to

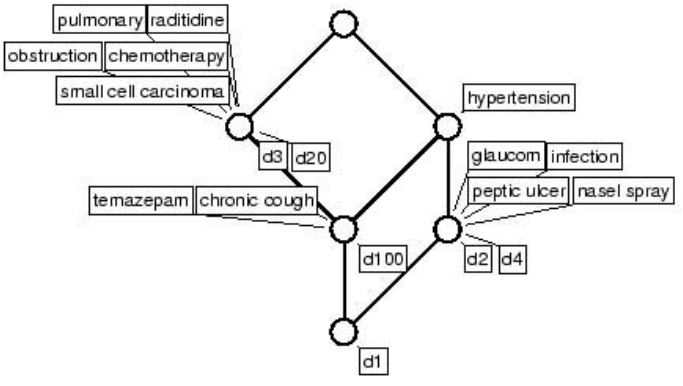


Fig. 3. A conceptual scale can be thought of as a *theme* that reduces the display and computational complexity. This figure shows a conceptual scale evident in the formal context shown in Fig. 2.

the top-right allows a text string to be entered, cross-referenced with aliased MeSH terms, and located in the text. The clinician is required to confirm likely matches, abbreviations and close spellings: once identified they are later recalled. This idea can be further extended to hyper-linking MeSH terms to navigate the document collection [16]. If we are interested in documents that contain the terms "carcinoma" and "Drug Therapy <1>" for example, the first document in this set of 214 is shown in Fig. 5.

One of the principles of our work is the human centred nature of reasoning and decision making. We achieve a dramatic reduction in the complexity of both algorithmic and visualisation components by placing the human operator at the centre of the information filtering process. The human selects a theme and this determines the output (these themes are called *conceptual scales* in the FCA literature). This idea is shown in Fig 6. With respect to logic-based networks the concepts are the MeSH terms (and their synonyms) and the relations are the presence (or absence) of Mesh terms in the text. Fig. 4 is the interface that permits these relations to be asserted. Both the computer and the user are engaged in the process of concept identification.

Placing the human at the centre of the information flow is obvious but often overlooked in intelligent systems. By asking the user to refine the scope of the exploration by defining a conceptual scale (theme) the complexity of the problem and its representation are decreased. In some ways the results of our work with medical texts are inconclusive. As an information retrieval metaphor, the concept lattices and conceptual scales have not been bench-marked against more established IR techniques but the intuitions are promising. Complexity can therefore be reduced by considering the following;

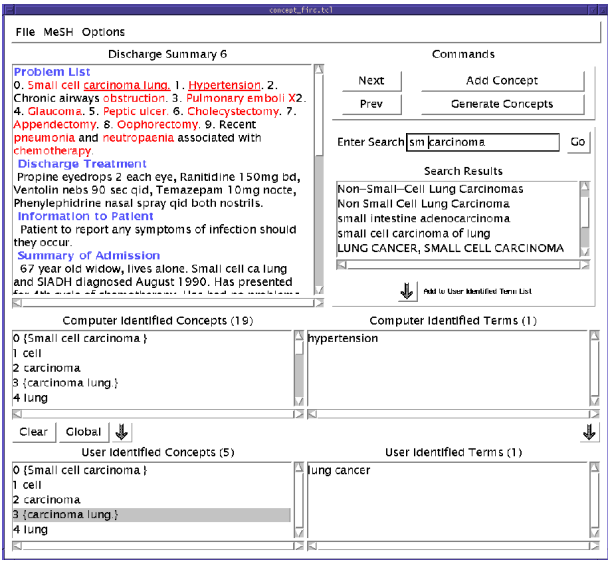


Fig. 4. The MeSH concept extraction program — shows the text (top-left) together with the MeSH terms that are contained within it (middle-left). The program was written by Richard Cole in TCL/TK.

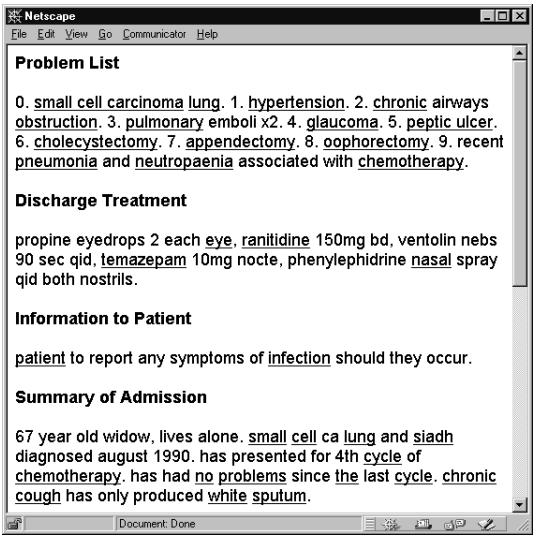


Fig. 5. First of 214 generated HTML documents hyper-linked on the MeSH terms "carcinoma" and "Drug Therapy <1>".

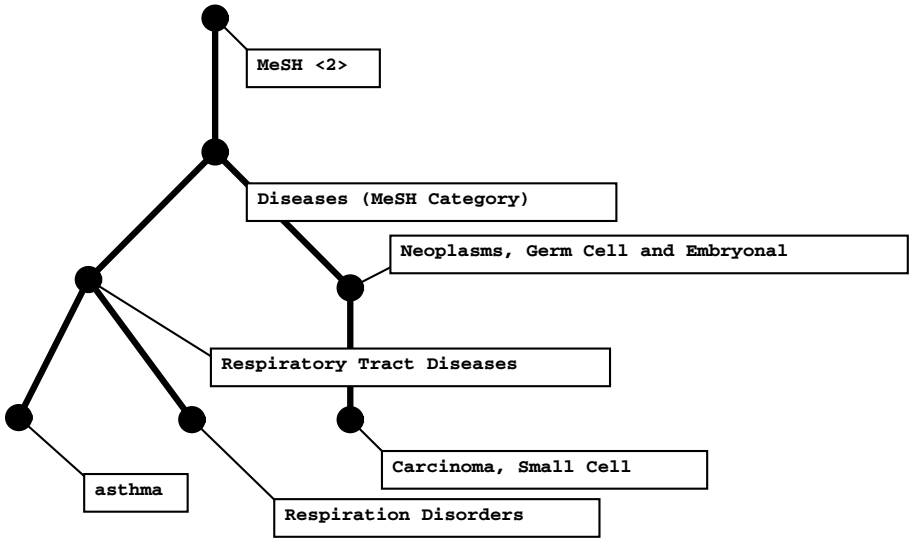


Fig. 6. Folding Poset Viewer with MeSH Concepts, displaying the theme created by the user: exploring a possible relationship between asthma and carcinoma.

1. reducing the attribute sets M to $\underline{M}$ by selecting only those attributes representing a "focus" of interest. This is described as a user-defined *theme* or *conceptual scale*.
2. by reducing the size of the object set G to $\underline{G}$ by collapsing documents into equivalent classes, i.e. if two or more documents contain the same set of attributes they can be considered as a single equivalence class of documents.
3. the analysis of the distribution of attributes in the document space can lead to efficient encoding techniques that reduce the space complexity of the representation.

Our emphasis is on visual outcomes used for text data mining. Showing that the visual complexity of the lattice representation can be used to explore a document collection in an intuitive human-centred interface is an important sub-goal. The graphical interface allows key terms to be identified from a medical dictionary of terms. It builds a theme (conceptual scale), the concept lattice which is a $\bigvee$-homomorphic image of the whole data context. Fig. 7 shows a nested line diagram of a simple scale exploring the relationship between smoking and carcinoma.

Feedback on the creation of the conceptual scale is provided in a number of ways. All synonyms of the key term to be scaled are shown and the number of documents containing a key term are displayed — this gives feedback on the uniqueness and relevance of the search term. For any given term, its children can be listed as well as its parents — this gives the user an easy way of constraining

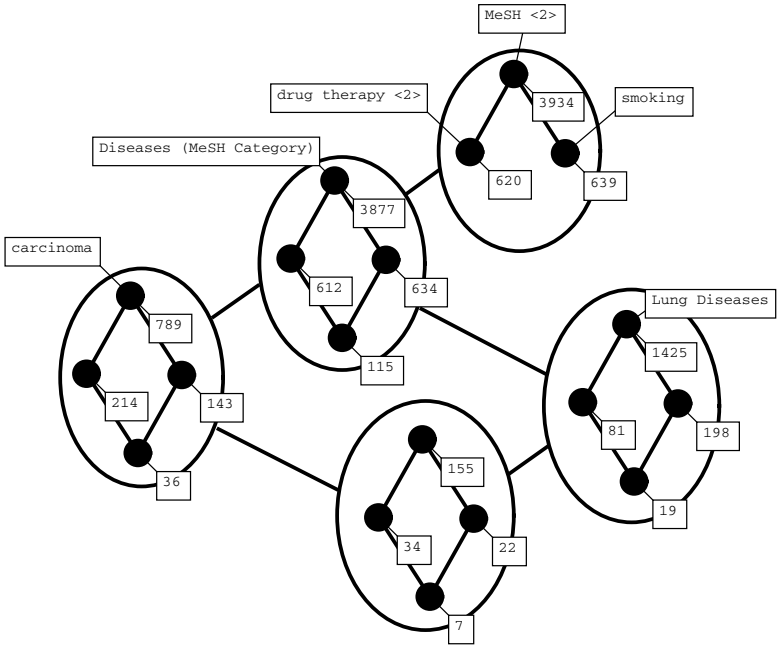


Fig. 7. A nested line diagram exploring the relationship between Carcinoma, Chemotherapy and Smoking. The numbers attached to vertices indicate the number of documents that cluster to that vertex.

search to include more documents through the use of a more general term, or alternatively, specialise a term to focus the search on fewer documents. Once the conceptual scale has been defined it may be saved together with an appropriate labelled line diagram of its concept lattice. W.r.t. logic-based networks, the conceptual scale is like a well-formed formula. The metaphor here is that the scale can only be a structural combination of concepts organised hierarchically. When two terms are added to the scale, their least upper bound is also added.

The presentation of the lattice is important and some specific lattice drawing algorithms can be imported from the graph layout literature [8,26] although this issue is far from conclusively researched [5]. Of further aid is the progressive (un)folding of the finite lattice, this is user-controlled, but provides a mechanism for reducing the display complexity.

3 Email Concept Analysis (CEM)

An extention of the application of FCA to document processing is to look at emails as documents. The CEM program [6] follows from the medical document system described above and embodies much of the elements our research phi-

losophy: it is useful, it relies on humans to boot-strap its knowledge by creating a conceptual ontology of terms and definitions and how these manifest. It is flexible, it allows the operator to design his theme. It is efficient, it uses fast algorithms that compute in polynomial time [7]. Unlike the medical document project the terms are user-defined types. Each types has a "classifier" that tells the information retrieval system how to associate a document to a type.

The CEM program is intended as a framework to create a visual and flexible view over email. Its purpose is to pose questions about the ways in which keywords combine in the email in order to extract (and find) interesting information. The CEM program is based on a *formal context* (G, M, I) where: the object set G consists of emails, the attribute set M consists of terms for emails, and I is the relation between emails and their assigned terms.

$a \to b$ is used to denote the implication determined by a formal context that: every object of the context that has attribute a will also have attribute b. In the case of email, we allow the user to state a set of implications that are enforced on the context. The implications are expressed by an order relation $\leq$ giving rise to an ordered attribute set $(M, \leq)$ (the order relation is reflexive, transitive, and anti-symmetric). A formal context, $\langle G, M, I \rangle$, is said to respect the set of implications given by $(M, \leq)$ if $(g, m) \in I$ and $m \leq n$ always imply $(g, n) \in I$. Consequently, if an email g has attribute m and the user has indicated that $m \leq n$ then the email will also have attribute n.

Each of the attributes may be associated with a classifier for associating emails. Two examples of classifiers are shown below. An email is said to be of the type "From Eklund" if it contains in the email "from" field the string "p.eklund@gu.edu.au". The type "KVO Meeting" is an email from "p.eklund@gu.edu.au" with the email subject "KVO Meeting".

```
"From Eklund"          From: p.eklund@gu.edu.au
"KVO Meeting"          From: p.eklund@gu.edu.au
                       Subject: KVO Meeting
```

The user is asked to create an implication ordering $<$ between admissible types.

```
"From Eklund" < "From KVO Group"
"From KVO Group" < "KVO Group"
```

In the logic-based network framework, the concepts are the types, the relations between the concepts and documents is determined by the user-defined "classifier". The type hierarchy gives us a way of formulating a coherent and connected theme/scale over the email. Inference patterns are "suggested" by the cardinality of emails that distribute over the lattice. The reflexive transitive closure of the statements made by the user gives $(M, \leq)$.

A conceptual scale (or theme) is determined by a set of types selected by the user. The user adds types to the conceptual view by searching over type names. When an type is selected it is added to the diagram showing the hierarchical ordering of the types. The diagram is completed by all types (ordered in advance) greater than a selected type. For instance, selecting the attribute "From Peter Deer" in Fig. 8 (top-left) forces the addition of "DSTO").

The concept lattice of the conceptual scale is shown below the scale in Fig. 8; filled circles represent concepts completing the hierarchy of typed concepts

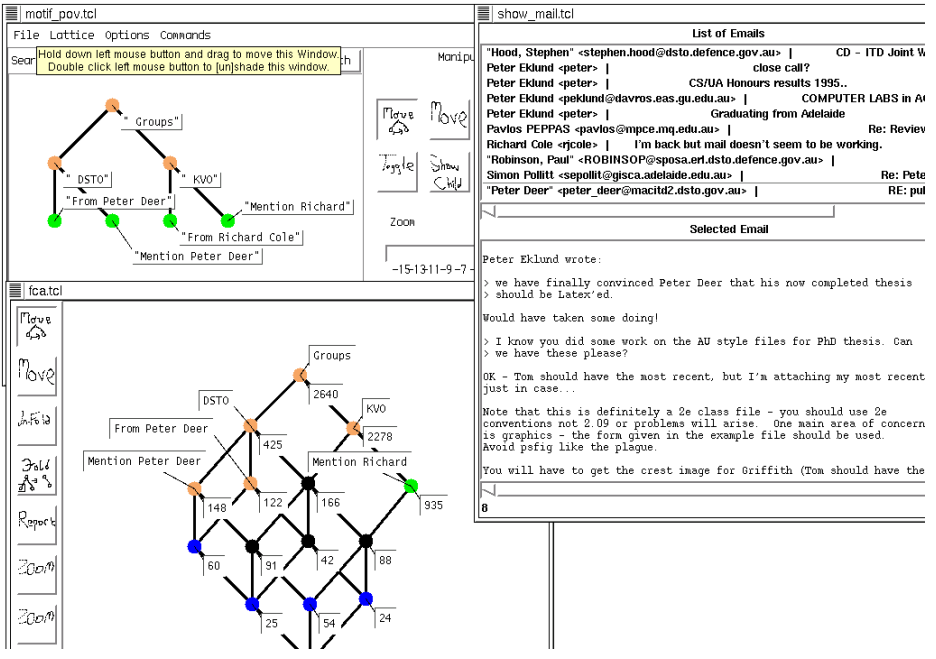


Fig. 8. The CEM Program Interface. Top-left is the theme editor, bottom-left the concept lattice resulting from the theme. On the top-right email headers and lower-right the email contents.

to obtain the whole lattice. The extent of the infimum of a selection of types concepts (example: "From Peter Deer", "Mention Richard") consists of all objects having the selected attributes; the number of those objects is attached to the circle representing the infimum (24 in the lower-left of Fig. 8). In Fig. 8 (bottom-left), the extent numbers show that there are 2,640 emails related to some research group. The 2,640 is split into 425 related to "DSTO" and 2,278 to "KVO", 166 emails are related to both.

The concept lattice may be large and difficult to comprehend. A successful method to draw larger concept lattices is representation by nested line diagrams. This is grounded on the construction of direct products of lattices. To draw a concept lattice using a lattice product we divide the attribute set M_s (user selected) into two sets M_1 and M_2. The two attribute sets can then be used to construct two concept lattices $\underline{\mathfrak{B}}(G, M_1, I \cap G \times M_1)$ and $\underline{\mathfrak{B}}(G, M_2, I \cap G \times M_2)$. Then the following mapping defines a $\bigvee$-preserving embedding of $\underline{\mathfrak{B}}(G, M_s, I)$ into the direct product of the two constructed concept lattices: $(A, B) \mapsto (\ (B \cap M_1)', B \cap M_1),\ (B \cap M_2)', B \cap M_2)\ )$. In Fig. 9, a nested line diagram of $\underline{\mathfrak{B}}(G, M_s, I)$ is shown. The elements of the first concept lattice with $M_1 := \{$"Groups" , " DSTC" , "From Melfyn" , "Mention Melfyn"$\}$ is represented by large circles each of which contains a copy of the diagram for the sec-

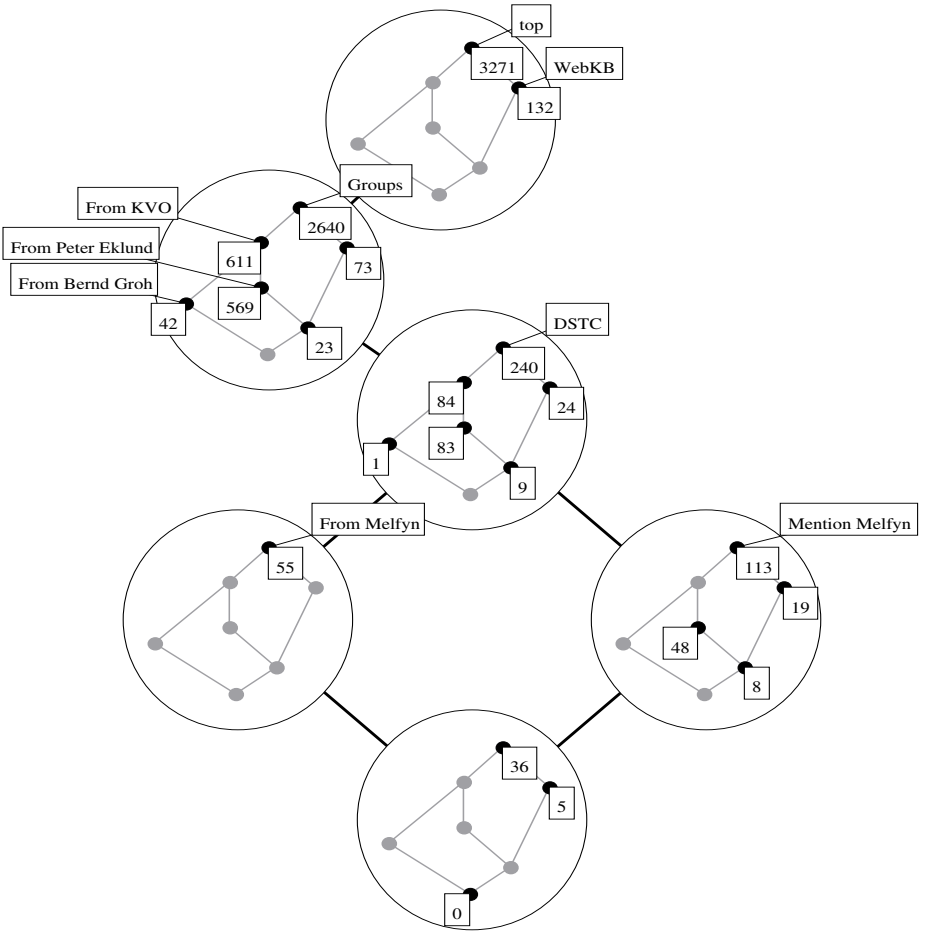


Fig. 9. Nested Line Diagrams reduce visual complexity.

ond concept lattice, $M_2 := \{$"top", "From KVO", "From Bernd Groh", "From Peter Eklund"$\}$, "WebKB"$\}$.

This nesting of the line diagrams of the two lattices is a graphical representation of the direct product of the lattices. The elements of the lattice product that are not mapped by concepts of $\underline{\mathfrak{B}}(G, M_s, I)$ are shown by grey circles. Grey circles indicate implications that come from the data rather than from the hierarchy, i.e. implicitly present in the data but not in the intended theory.

In Fig. 9, where the outer scale is M_1 and the inner scale M_2, there are 113 emails that mention Melfyn (lower-left ellipse) and 569 that are from "Peter Eklund" (middle vertex in upper-left ellipse). Of the 113 emails that mention Melfyn there are 48 that are also "From Peter Eklund" (middle vertex in lower-left ellipse). Of the 113 emails that mention Melfyn 48 (almost half) come from

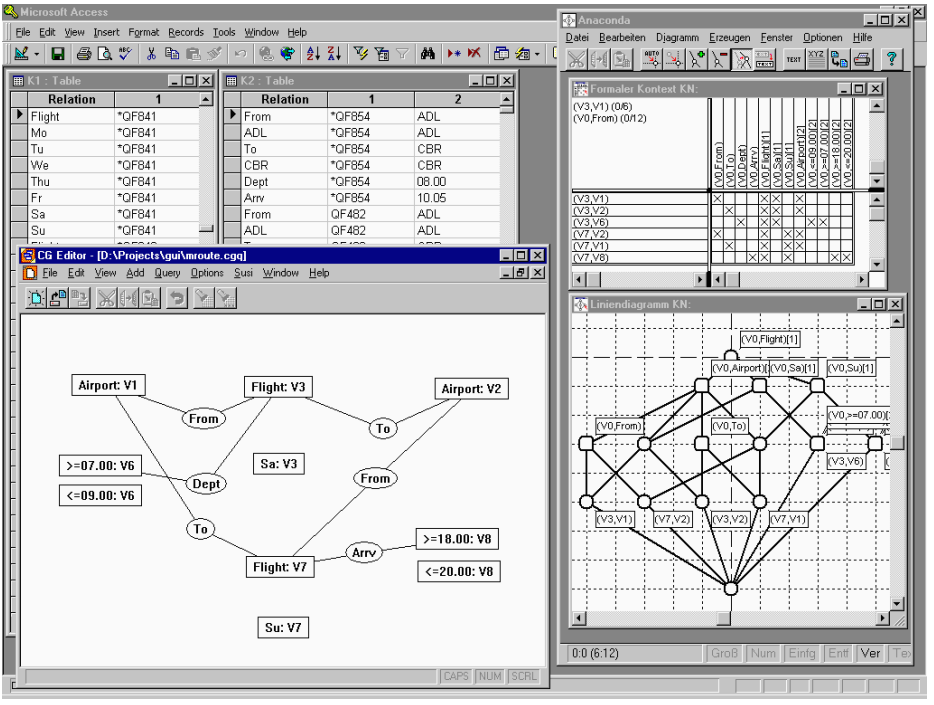


Fig. 10. Bernd Groh's CGPCF TOSCANA integration. The lower left shows a CG, top-left the relational database formulation of a schedule (as knowledge-base), top-right cross-table as a Power Context Family (PCF), and bottom-right the formal concept lattice of the cross-table.

Peter Eklund. Melfyn is mentioned mostly by Peter Eklund and Melfyn (likely as a signature). Over half the email sent by Peter Eklund concerning the DTSC mentions "Melfyn". These are some of the simple inferences that can be read from the nested line diagram in this simple example.

4 Concept(ual) Graphs - CGPCF

Conceptual Graphs (CG) can be used to describe more completely facts about the world but they have sometimes lacked a precise semantics and a scalable implementation. Wille developed the theory of a *power context families* (PCFs) that presents *concept graphs* as algebraic entities called a PCF. Bernd Groh developed an algorithm that can derive concept graphs from a PCF [15]. Bernd implemented this theory and released it as a CG implementation with a back-end relational database management system shown in Fig. 10.

CGPCF is an example of the convergence of theoretical and empirical theory building, it is neither a general purpose programming language nor an appli-

cation program, it is a powerful, scalable and general technique for storing, searching and retrieving complex data formulated as a knowledge-base.

From the point of view of logic-based networks, we have come up with a data structure that contains explicit representations of concepts and relations connecting concepts. Relations are therefore no longer user-determined but rather user-defined. The problems that this imposes are many. If relations between concepts are explicitly contained with the text we need mechanisms that facilitate their extraction. The technical question is how to generate wff from the PCF and how to use those formulas for inference. This is a current area of investigation.

5 WebKB

In terms of logic-based networks we now come full circle. WebKB [18,19] is an important system because it is the first of its type that allows knowledge to be described from the Web. WebKB can be thought of as a practical example of logic-based networks at work to create a "semantic Web". In this sense it is ahead of the emerging Web standards in XML that will permit knowledge embedded into Web documents.

WebKB is a system for annotating document elements in Web-pages with conceptual graphs that are representative of semantic meaning. This is an activity that in WebKB is done by human hand. However, we have learned in this paper that it is possible to generate conceptual graphs automatically from Web documents exploiting type and relation hierarchies & concept and term identification techniques: storing this knowledge as a power context family from which a CG can automatically be derived. Once derived, the CGs can be used to infer more knowledge. We call this idea of bringing the semantic content of two or more documents together *knowledge fusion*. The result is a program called HibKB (which I will demo at ICCS2000). In the meanwhile it is worthwhile reviewing WebKB since it demonstrates a high-level Web-based inferencing system for CGs and how such a system can hyper-link a knowledge-base and document collection.

Retrieving semantic content on the Web is an open research problem. Despite this, some of the infrastructure that supports this task is becoming available. Manually structuring Web documents using syntactic mark-up languages such as XML[3] allows Web-robots to retrieve relatively precise information using string and structure-matching techniques. However, the Web robot approach is not scalable because fine-grained information is retrieved only when documents are richly structured and the querier understands this structure: the exact tag names and their order. Using representation languages as meta-data is a solution to the problem but it imposes several requirements on the representation language.

A first requirement is that the notation be intuitive and concise enough to be read and understood by people. Most current knowledge-oriented meta-data

[3] *http://www.w3.org/XML/*

languages are built on top of XML, e.g. RDF[4] and OML[5]. Choosing XML ensures that standard XML-based tools can parse and exchange meta-data. But standard XML tools are of little interest in managing knowledge-oriented meta-languages since specialised editors, analysers and inference engines are required.

Another requirement is that an author should be able to render knowledge statements visible. This is particularly true when visual or control languages are used. Then, for example, a knowledge base and its associated documentation can be integrated within the same document and both accessed and managed using classic information retrieval techniques (string-search, structure navigation via table of contents, etc) as well as knowledge-based techniques.

Another requirement of a meta-data language is that it should be both precise and general enough to allow users to represent any Web-accessible information at a desired level of precision. This means the meta-data language should have an expressive formal model. Any formalism equivalent to first-order logic permitting the use of nested "contexts" is an appropriate candidate, e.g. the Knowledge Interchange Format (KIF)[6] or Conceptual Graphs (CGs)[7] [24]. It is important not to restrict the expressivity of the language but it is reasonable to forgo some features in return for efficiency. This means that an inference engine may ignore some features of the knowledge representation language in the same way that a HTML browser can exploit some feature tags but not others.

In summary, the three requirements for *precise, flexible and scalable* knowledge and information retrieval implies firstly an easy to use notation that is intuitive, precise and expressive and secondly, the capacity to insert knowledge anywhere within a Web document. WebKB[8] [18] responds to these requirements. It interprets knowledge statements stored within Web-accessible documents. The knowledge representation language is specified with a tag attribute, e.g. `<KR language="CG">`. WebKB interprets the linear notation of CGs as well as some user-friendly notations: a formalised English, a frame-like notation for CGs, and structures that relate document elements by semantic relations[9].

CGs were selected because they have a graphical notation and a linear notation, both concise and relatively intuitive, and secondly because we can reuse two CG inference engines (CoGITo [17] and Peirce [13]). The inference engines exploit subsumption defined between formal terms for calculating specialisation between CGs — and therefore between queries and facts in a knowledge base. Hence, statements and queries can be made at varying levels of precision.

WebKB permits virtual documents [21] by combining lexical, structural and knowledge-based data management by proposing commands for searching and joining CGs, Unix-like file management commands working on Web-accessible

[4] *http://www.w3.org/RDF/*
[5] *http://wave.eecs.wsu.edu/CKRMI/OML.html*
[6] *http://logic.stanford.edu/kif/kif.html*
[7] *http://meganesia.int.gu.edu.au/~phmartin/WebKB/doc/CGs.html*
[8] *http://meganesia.int.gu.edu.au/~phmartin/WebKB/*
[9] See "Conventions and Notations for Knowledge Representation and Retrieval", P. Martin in these proceedings.

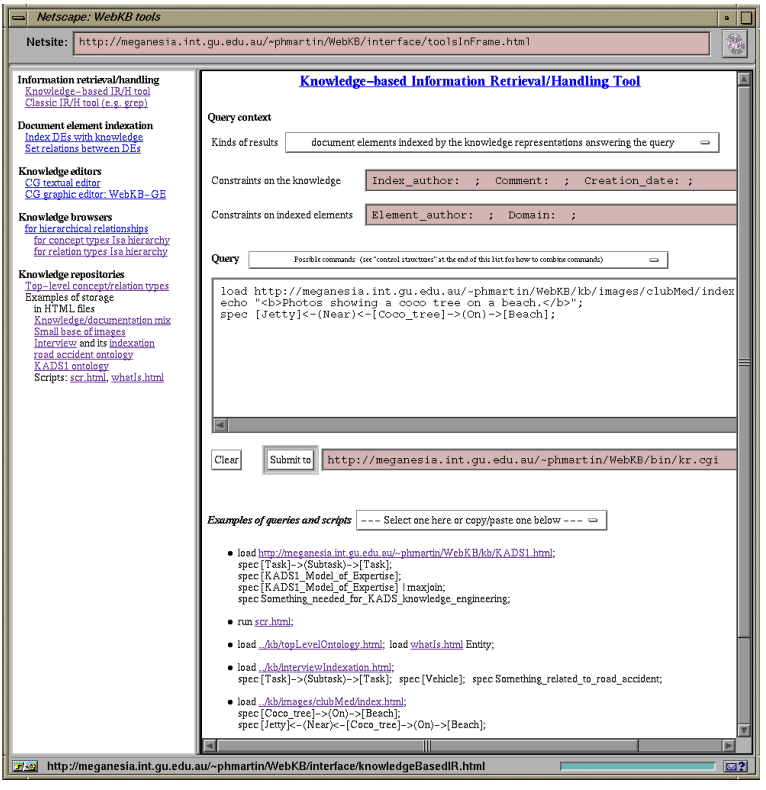


Fig. 11. The WebKB menu and knowledge-based Information Retrieval/Handling Tool. The example query shows how a document containing CGs (indexing images) is loaded into the WebKB processor and then how the command **spec** searches for specialisations of the CG `[Jetty]<-(Near)<-[Coco-tree]->[On]->[Beach]`.

documents and a simple Unix shell-like script language to combine commands. These commands may be inserted in documents and sent to WebKB programs.

5.1 Representing Knowledge

To compare the alternatives for knowledge representation and retrieval on the Web, Fig. 12 shows how a simple sentence may be represented with CGs, with KIF and with RDF. The sentence is:

"John believes that Mary has a cousin who has the same age as her".

The CG representation (Fig. 12 top) seems simpler than the others. The semantic network structure of CGs (i.e. typed concepts connected by typed relations) has three advantages. Firstly, it restricts the notation without compromising expressivity — this reduces computational overhead when comparing CGs. Secondly, it encourages users to use explicit relations between concepts, and finally

```
<KR language="CG">
load "http://www.bar.com/topLevelOntology";    //Import this ontology
Age < Property;     //Declare Age as a subtype of Property
Cousin(Person,Person) {Relation type Cousin};

[Person:"John"]<-(Believer)<-
         [Descr: [Person:"Mary"]-{(Chrc)->[Age: *a];
                                   (Cousin)->[Person]->(Chrc)->[*a];
                                 }]; </KR>
```

```
<KR language="KIF">
load "http://www.bar.com/topLevelOntology";    //Import this ontology
(Define-Ontology Example (Slot-Constraint-Sugar topLevelOntology))
(Define-Class Age (?X)  :Def (Property ?X))
(Define-Relation Cousin(?s ?p) :Def (And (Person ?s) (Person ?p)))

(Exists ((?j Person))
  (And (Name ?j John) (Believer ?j
                         '(Exists ((?m Person) (?p Person) (?a Age))
                              (And (Name ?m Mary) (Chrc ?m ?a)
                              (Cousin ?m ?p) (Chrc ?p ?a)
                        ))))) </KR>
```

```
<!-- RDF notation; assumed location: http://www.bar.com/example -->
<RDF xmlns="http://www.w3.org/TR/WD-rdf-schema#"
     xmlns:t="http://www.bar.com/topLevelOntology#">

  <Class ID="Age"><subClassOf resource="t:Property"/></Class>

  <PropertyType ID="Cousin" comment="Relation type Cousin">
                    <range  resource="t:Person"/>
                    <domain resource="t:Person"/></PropertyType> </RDF>

<RDF xmlns="http://www.w3.org/TR/WD-rdf-schema#"
     xmlns:x="http://www.bar.com/example#"
     xmlns:t="http://www.bar.com/topLevelOntology#">

  <t:Person bagID="Statement_01">
              <t:Name>Mary</t:Name>
              <t:Chrc><x:Age ID="age"></x:Age></t:Chrc>
              <x:Cousin><t:Person><t:Chrc resource="x:age"/></t:Cousin>
  </t:Person>
  <Description aboutEach="#Statement_01" t:Believer="John"/> </RDF>
```

Fig. 12. Comparing knowledge representation with CGs, KIF and RDF.

it permits a better visualisation of relations between concepts — relations can be rendered as an edge and concepts and a vertex.

Users want flexibility. They do not always have time to declare and order some of the terms used when representing knowledge. This is the case when indexation is for private knowledge organisation purposes or when detail is omitted during work in progress. To permit this, and still allow the system to perform some minimal semantic checks, basic declared relation types are used and concept

types left undeclared. The rationale for this is that when knowledge statements are made from concepts linked by basic relations, the complexity is concentrated within concept types and only a limited set of relation types are necessary. WebKB processes 200 basic relation types[10] which collect common thematic, mathematical, spatial, temporal, rhetorical and argumentative relations types. This collection of basic types is sufficient for most work.

Secondly, WebKB can use relation signatures to give types to the undeclared terms used as concept types. For instance, in the top-level ontology in WebKB, the relation types *Input, Output, Agent, Method, SubProcess* and *Purpose* are all defined to have a concept of type *Process* as the first argument. WebKB can infer from this ontology that *Knowledge_design* must be a subtype of *Process* if used with the above relation types.

The ontology of types becomes an important tool for a system like WebKB. WebKB controls type signatures and knowledge vocabulary so it needs to be a rich and diverse collection. In WebKB, we merged the WordNet[11] ontology — 120,000 words linked to 90,000 concept types — into our top-level ontology.

Subsumption is an inference that determines whether one knowledge statement can be inferred from another by specialisation or generalisation. From our previous example, `[Cat]->(On)->[object]` is a generalisation of `[Cat]->(On)->[Table]`. Likewise, `[Cat]->(On)->[Table]->(Attr)-[Legs:{*}@4]`, a specialisation of `[Cat]->(On)->[Table]` and `[Cat]->(On)->[object]`. This process of comparison is the main mechanism for search and inference in WebKB. When knowledge statements follow the same conventions they can be readily compared. Using a common and basic relation vocabulary is thus important. A common convention for primitive concepts and complex relations makes comparison more difficult. Take for example the sentence "Mary is 20 years old". Following our convention a simple relation "`(Chrc)`" is used, `[Person:"Mary"]->(Chrc)->[Age:@20]`, the inverse convention would use `(Age)` as a relation type with the simple concept type `[Integer]`, `[Person:"Mary"]->(Age)->[Integer: 20]`.

We call a Document Element (DE) any textual/HTML data, for example a sentence, a connection, a reference to an image or an entire document. This definition excludes binary data but includes textual knowledge statements. WebKB allows users to index any DE of a Web-accessible document with knowledge statements, or connect DEs from the same or different documents using relations. Fig. 13 shows an example of each.

The above notations allow the statements and the indexed DEs to be in different documents. Thus, any user may index any element of a document on the Web. Fig. 11 presents a general interface for knowledge-based queries, showing how a document containing knowledge is loaded into WebKB.

WebKB also allows the document owner to index an image by a knowledge statement directly stored in the "`alt`" field of the HTML "`img`" tag. We use this special case of indexation to present a simple illustration of WebKB's features. This example, shown in Fig. 14, is a good synthesis but is in no way representative

[10] *http://meganesia.int.gu.edu.au/~phmartin/WebKB/kb/topLevelOntology.html*
[11] *http://www.cogsci.princeton.edu/~wn/*

```
$(Indexation
   (Context: Language: CG;
      Ontology: http://www.bar.com/topLevelOntology.html;
      Repr_author: phmartin; Creation_date: Mon Sep 14 02:32:21 1998;
      Indexed_doc: http://www.bar.com/example.html; )
   (DE: {2nd occurence}  the red damaged vehicle )
   (Repr: [Color: red]<-(Color)<-[Vehicle]->(Attr)->[Damaged] ) )$
$(DEconnection
   (Context: Language: CG;
      Ontology: http://www.bar.com/topLevelOntology.html;
      Repr_author: phmartin; Creation_date: Mon Sep 14 02:53:36 1998;)
   (DE: {Document: http://www.bar.com/example.html} )
   (Relation: Summary)
   (DE: {Document: http//www.bar.com/example.html}
               {section title: Abstract}))$
```

Fig. 13. A language for knowledge indexing or connecting any Web-accessible document element.

of the general use of WebKB — it is not representative because it mixes the indexed source data (in this case, a collection of images), their indexation, and a customised interface to query them, in a single document. Typically, these elements would be split into separate documents. The result of the query shown in Fig. 14 is displayed in Fig. 15.

Because WebKB proposes knowledge representation and query commands, and a script language, we have not felt the need to give it a lexical and structural query language as precise as Harvest, WebSQL or WebLog. Instead, we have implemented Unix-like text processing commands for exploiting Web-accessible documents. The command list includes: `cat, grep, fgrep, diff, head, tail, awk, cd, pwd, wc` and `echo`. A hyper-link path-exploring command "`accessibleDocFrom`" is also provided. This command lists the documents directly and indirectly accessible from given documents within a maximal number of hyper-links. For example, the following command lists the HTML documents accessible from http://www.foo.bar/foo.html (maximum 2 levels) including the string "knowledge" in their HTML source code.

```
accessibleDocFrom -maxlevel 2
   -HTMLonly http://www.foo.bar/foo.html | grep knowledge
```

WebKB includes commands for displaying specialisations or generalisations of concept and relation types or of an entire CG. At present, queries for CG specialisations retrieve only connected CGs: the processor cannot retrieve paths between concepts specified in a query. If a retrieved CG indexes a document element, it can be presented instead of the CG (Fig. 15 gives an example). In both cases, hyper-links are generated to reach the source of each answer presented in its original document.



Fig. 14. Images, knowledge indexations and a customised query interface contained within a same document. The example query shows how the command "`spec`" which looks for specialisations of a CG can be used to retrieve images indexed by CGs. The results are shown in Fig. 15).

Specialisation gives the user freedom in the way queries can be formulated: searches may be done at a general level and subsequently refined according to the results. However, the exact names of types must be known. WebKB allows the user to supply only a substring of a type in a query CG, if prefixed by the wildcard character %. WebKB replaces the substring by declared types including the substring. Replacements violating relation signatures or individual types are discarded. For example, `spec [%thing]` will trigger the generation and execution of `spec [Something]`, `spec [thingone]` and `spec [thingtwo]`.

Knowledge query commands may be combined with the WebKB script language to generate complex documents, perform consistency tests on knowledge bases, or solve problems procedurally. The WebKB site provides many examples of queries and scripts. For example, one script solves a classical resource allocation problem, the Sisyphus-I room allocation problem[12].

[12] *http://meganesia.int.gu.edu.au/~phmartin/WebKB/kb/sisyphus1.html*

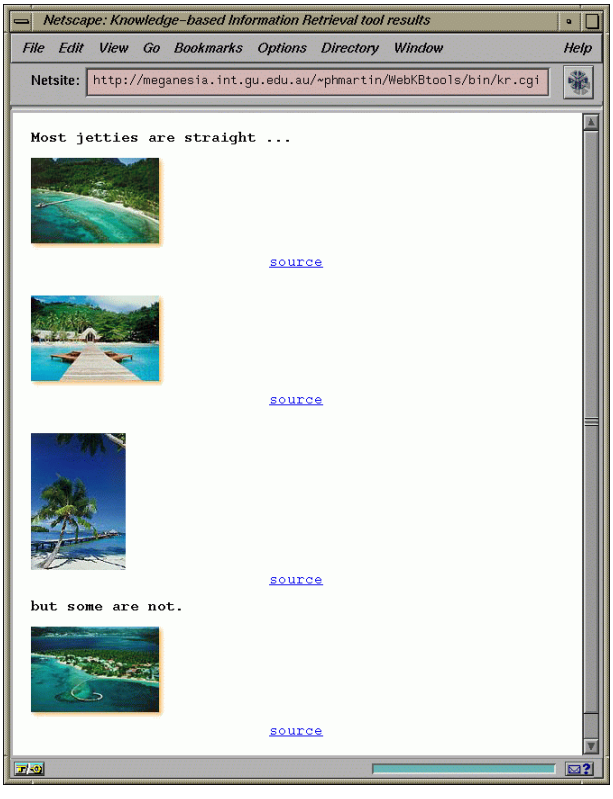


Fig. 15. The document generated in response to the query in Fig. 14.

One way to generate new knowledge in WebKB is by joining CGs. Various kinds of joins over CGs may be used but WebKB only proposes joins which, given a set of CGs, create a new CG specialising each of the source CGs. The result is inserted in the CG base although it may not represent anything true for the user. This does however provide a device for accelerating knowledge representation. For instance, CGs related to a type may be collected and automatically merged via a command such as this one: `spec [TypeX] | maxjoin`. The result may then serve as a basis for the user to create a type definition for TypeX. The following is a concrete example for the maximal join command.

```
> maxjoin  [Cat]->(On)->[Mat] [Cat:Tom]->(Near)->[Table]
 [Cat:Tom]- { (On)->[Mat];
              (Near)->[Table]; }
```

Ontology servers, such as the Ontolingua ontology server[13] and Ontosaurus[14], support shared knowledge repositories. However, Ontology servers are not usable

[13] *http://WWW-KSL-SVC.stanford.edu:5915/*
[14] *http://www.isi.edu/isd/ontosaurus.html*

for managing large quantities of knowledge and, apart form AI-Trader [23], they do not allow indexation and retrieval of parts of documents. Finally, support of cooperation between the users is essentially limited to consistency enforcement, annotations and structured dialogues, as in APECKS[15], Co4[16] and Tadzebao[17].

We are extending WebKB to handle a knowledge repository. We are considering five issues which address scalability: (i) the implementation of the knowledge-based system reuses FastDB[18], a scalable multi-user persistent object repository, and (ii) algorithms allowing the exploitation of large-scale dynamic taxonomies efficiently[19]; (iii) visualisation techniques (handling term-aliases and view generation) to avoid lexical conflicts and enable users to focus on certain kinds of knowledge; (iv) protocols allowing users to solve semantic conflicts via the insertion of new terms and relations in the common ontology and, in some cases, the knowledge of other users; (v) conventions to improve the automatic comparison of knowledge by different users.

6 Conclusion

The purpose of this paper has been to profile our research involving concept(ual) graphs. There are some general recommendations that can be drawn: (i) useful intelligent systems result where general frameworks instantiate real application problems; (ii) assumptions about user intentions are minimised when the human operator is placed at the centre of the information system. In this way the combinational complexity of an information system can also be minimised; (iii) this in turn facilitates information flows. The hypothesis of our work is that real progress in intelligent system results from this these three recommendations.

To this end we have demonstrated a number of intelligent computer systems: a document filtering and retrieval system for medical texts, a simular system for the recovery and analysis of email, a system called CGPCF that fuses formal concept analysis with relational databases and conceptual graphs and our Web-accessible precision-based information retrieval system, WebKB.

Acknowledgements

Special thanks to members of the KVO laboratory who have prepared the subject matter for this paper: Drs. Philippe Martin, & Francois Modave, Richard Cole, Bernd Groh, Thomas Tilley and Natalyia Roberts. I would also like to thank Prof. Rudolf Wille for his corrections. Any remaining errors are my own.

[15] *http://www.psychology.nottingham.ac.uk/staff/Jenifer.Tennison/APECKS/*
[16] *http://ksi.cpsc.ucalgary.ca/KAW/KAW96/euzenat/euzenat96b.html*
[17] *http://ksi.cpsc.ucalgary.ca:80/KAW/KAW98/domingue/*
[18] *http://www.ispras.ru/ knizhnik/fastdb.html*
[19] *http://www.cs.sfu.ca/cs/people/GradStudents/fall/personal/index.html*

References

1. *1998 MeSH, Annotated Alphabetic List.* National Technical Information Service, U.S. Department of Commerce, Springfield, VA 22161, 1998.
2. A. L. Burrow and P.W. Eklund. Visual structure representations and conceptual graphs. In *Proceeedings of the 3rd International Workshop on PEIRCE*, pages 45–52. ICCS '94, 1994.
3. A.L. Burrow, S.E. Pollitt, and P.W. Eklund. WebKB-GE - A Visual Editor for Canonical Conceptual Graphs. In *Proceedings of the 6th International Conference on Conceptual Structures ICCS '98*, LNAI 1453, pages 111–118. Springer, 1998.
4. C. Carpineto and G. Romano. A lattice conceptual clustering system and its application to browsing retrieval. *Machine Learning*, 24:95–122, 1996.
5. R. Cole. Automated layout of concept lattices using force directed placement and genetic algorithms. In J. Edwards, editor, *The 23rd Australasian Computer Science Conference*, volume 22 of *Australian Computer Science Communications*, pages 31–42, Los Alamitos, CA, 2000. IEEE Press.
6. R. Cole and P. Eklund. Analyzing an email collection using formal concept analysis. In *Proceedings of the European Conf. on Knowledge and Data Discovery*, LNAI 1704. Springer, 1999.
7. R. Cole and P. Eklund. Scalability in formal concept analysis. *Computational Intelligence*, 15(1):11–27, 1999.
8. M. K. Coleman and D. Stott-Parker. AGLO — Publications and Implementation. *Software – Practice and Experience*, pages 1415–1438, December 1996.
9. P. Eklund. Using path-algebras to encode directed graph knowledge-bases. In *4th Annual Workshop on Conceptual Structures*, Detroit, MN, 1989.
10. P. Eklund and J.M. Kellett. Prospects for conceptual graphs in knowledge acquisition interfaces. In *Proceedings of the 3rd European Workshop on Knowledge Acquisition*, 1989.
11. P. Eklund, J. Leane, and C. Nowak. GRIT: A GUI for Conceptual Structures. In *Proceeedings of the 2rd International Workshop on PEIRCE.* ICCS '93, 1993.
12. P.W. Eklund and P.H. Martin. WebKB indexation and document retrieval using conceptual structures. In *Proceedings of IEEE Conference on Intelligence Information Processing ICIPS '98*, pages 217–221. IEEE Press, 1998.
13. G. Ellis. *Managing Complex Objects.* PhD thesis, Computer Science Department, The University of Queensland, 4072, Queensland, Australia, 1995.
14. B. Ganter and R. Wille. *Formal Concept Analysis: Logical Foundations.* Springer, 1999.
15. B. Groh and P. Eklund. Algorithms for creating relational power context families from conceptual graphs. In *Proceedings 7th International Conference on Conceptual Structures*, LNAI 1640, pages 389–400. Springer, 1999.
16. B. Groh and P.W. Eklund. Dynamic hyper-linking by querying for a FCA-based query system. In *Proceedings of the Forth Australiasian Document Computing Symposium*, pages 9–14, 1999.
17. O. Haemmerlé. *CoGITo: une plate-forme de développement de logiciels sur les graphes conceptuels.* PhD thesis, Montpellier II University, 1995.
18. P. Martin and P. Eklund. Embedding knowledge in Web documents. In *The 8th International World Wide Web Conference, (WWW8)*, pages 324–341. Elsevier, 1999. Also published in Int. Journal of Computer Telecommunication Networking, Vol 31, 1403–1419, 1999.

19. P. Martin and P. Eklund. Knowledge indexation and retrieval and the Word Wide Web". *Intelligent Systems — Special Issue on Knowledge Management and Knowledge Distribution over the Internet*, May, 2000.
20. P.H. Martin. The WebKB set of tools: a common scheme for shared WWW annotations, shared knowledge bases and information retrieval. In M. Keeler L. Searle D. Lukose, H. Delugach and J. Sowa, editors, *Proceedings of the CGTools Workskop at the 5th International Conference on Conceptual Structures ICCS '97*, LNAI 1257, pages 595–588. Springer, 1997.
21. J. Nanard, M. Nanard, A. Massotte, A. Joubert, H. Betaille, and J. Chauché. Integrating knowledge-based hypertext and database for task-oriented access to documents. In *DEXA'93*, LNAI 720. Springer, 1993.
22. S. Prediger and R. Wille. The lattice of concept graphs of a relationally scaled context. In *Conceptual Structures: Standards and Practices*, volume 1640 of *LNAI*, pages 401–411. Springer, 1999.
23. A. Puder and K. Romer. Generic Trading Service in Telecommunication Platforms. In *Proceedings of the International Conference on Conceptual Structures (ICCS '97)*, LNAI 1257, pages 551–565. Springer, 1997.
24. J Sowa. *Conceptual Structures : Information Processing in Mind and Machine*. Addison-Wesley, 1983.
25. J. Sowa. Conceptual graphs summary. In L. Gerholz T. Nagle, J. Nagle and P. Eklund, editors, *Conceptual structures: Current theory and practice*, pages 3–52. Ellis Horwood, 1992.
26. R. Wille. Line diagrams of hierarchical concept systems. *International Classification*, 11:77–86, 1984.
27. R. Wille. Conceptual graphs and formal concept analysis. In M. Keeler H. Dulugach and D. Lukose, editors, *The 4th International Conference on Conceptual Structures*, number 1257 in Lecture Notes on Computer Science. Springer, 1997.

Conceptual Knowledge Discovery and Data Analysis

Joachim Hereth[1], Gerd Stumme[1], Rudolf Wille[1], and Uta Wille[2]

[1] Technische Universität Darmstadt, Fachbereich Mathematik, Schloßgartenstr. 7, D–64289 Darmstadt, Germany,
{hereth, stumme, wille}@mathematik.tu-darmstadt.de
[2] Jelmoli AG, Data Management, Postfach 3020, Ch–8021 Zürich, Switzerland;
wille_u@jelmoli.ch

Abstract. In this paper, we discuss Conceptual Knowledge Discovery in Databases (CKDD) in its connection with Data Analysis. Our approach is based on Formal Concept Analysis, a mathematical theory which has been developed and proven useful during the last 20 years. Formal Concept Analysis has led to a theory of conceptual information systems which has been applied by using the management system TOSCANA in a wide range of domains. In this paper, we use such an application in database marketing to demonstrate how methods and procedures of CKDD can be applied in Data Analysis. In particular, we show the interplay and integration of data mining and data analysis techniques based on Formal Concept Analysis. The main concern of this paper is to explain how the transition from data to knowledge can be supported by a TOSCANA system. To clarify the transition steps we discuss their correspondence to the five levels of knowledge representation established by R. Brachman and to the steps of empirically grounded theory building proposed by A. Strauss and J. Corbin.

Contents

1. Conceptual Knowledge Discovery in Databases
2. Conceptual Data Analysis
3. From Data to Knowledge
4. Procedures of Conceptual Knowledge Discovery

1 Conceptual Knowledge Discovery in Databases

Conceptual Knowledge Discovery in Databases (*CKDD*) has been developed in the field of *Conceptual Knowledge Processing*. Based on the mathematical theory of *Formal Concept Analysis*, CKDD aims to support a human-centered process of discovering knowledge from data by visualizing and analyzing the formal conceptual structure of the data. Implementing the basic methods of Formal Concept

B. Ganter and G.W. Mineau (Eds.): ICCS 2000, LNAI 1867, pp. 421–437, 2000.
© Springer-Verlag Berlin Heidelberg 2000

Analysis, the management system TOSCANA has been used as a knowledge discovery tool in various research and commercial projects (cf. [35]). The general approach of CKDD and the qualities of TOSCANA as a KDD support tool have previously been discussed in [27] with respect to Brachman and Anand's fundamental requirements for knowledge discovery support environments (cf. [4]). Therefore, the basic notions and the philosophical background of CKDD are only briefly summarized in this paper. For a comprehensive presentation of the mathematical foundations of Formal Concept Analysis see [10]; basics of Conceptual Knowledge Processing are explained in [31],[32],[33],[35].

The overall theme and contribution of the volume *"Advances in Knowledge Discovery and Data Mining"* [7] is a process-centered view of KDD considering KDD as an interactive and iterative process between a human and a database that may strongly involve background knowledge of the analyzing domain expert. In particular, R. S. Brachman and T. Anand [4] argue in favor of a more human-centered approach to knowledge discovery support referring to the constitutive character of human interpretation for the discovery of knowledge and stressing the complex, interactive process of KDD as being led by human thought.

Following Brachman and Anand, CKDD pursues a human-centered approach to KDD based on a comprehensive notion of knowledge as a part of human thought and argumentation. The *landscape paradigm* of knowledge underlying CKDD is based on the pragmatic philosophy of Ch. S. Peirce [16] where knowledge is understood as always being incomplete, formed and continuously assured by human discourse within an intersubjective community of communication (cf. [35]). Emphasizing the intersubjective character of knowledge, CKDD considers knowledge communication as an important part of the overall discovery process with respect to both the dialog between user and system, and also as a part of human communication and argumentation. Therefore, a major focus of CKDD is to provide knowledge discovery support that guarantees a high transparency of the discovery process and a representation of its (interim) findings to support human argumentation and establishment of intersubjectively assured knowledge. CKDD especially supports a wide-ranging and unpredictable interactive exploration of the data ("data archaeology", cf. [5]) where the software tools TOSCANA and Chianti serve as a knowledge discovery support environment in which CKDD applications can be efficiently implemented (see [27]).

2 Conceptual Data Analysis

CKDD is based on methods and procedures of *Conceptual Data Analysis* that allow the analysis of given data by examination and visualization of their conceptual structure. The derived graphical representations have proven to be useful for making the data communicable in addition to identifying conceptual relationships in the data. Knowledge is discovered in interaction with the data during an iterative process which activates techniques of Conceptual Data Analysis and is guided by theoretical preconceptions and declared purposes of the domain expert. In the following paragraphs, we briefly introduce the basic notions and

procedures of Conceptual Data Analysis using an application in database marketing.

Based on a philosophically grounded formalization of concept (see [34]), Conceptual Data Analysis allows data to be mathematically treated and processed. Formal Concept Analysis, the mathematical theory underlying Conceptual Data Analysis, formalizes concept and conceptual hierarchy to reflect the philosophical understanding of a concept as a unit of thought constituted by its extension and its intension. The extension comprises all objects belonging to the concept while the intension consists of all attributes valid for those objects. To allow a mathematical description of extension and intension, Formal Concept Analysis always starts with a formal context:

Definition 1. *A* formal context *is a set structure* $\mathbb{K} := (G, M, I)$ *where* G *and* M *are sets and* I *is a binary relation between* G *and* M *(i. e.* $I \subseteq G \times M$*). The elements of* G *and* M *are called (*formal*)* objects *and* attributes*, respectively, and* gIm *(*$\Leftrightarrow (g, m) \in I$*) is read: "the object* g has *the attribute* m*".* Derivations *are defined by* $X' := \{m \in M \mid \forall g \in X : gIm\}$ *for* $X \subseteq G$ *and* $Y' := \{g \in G \mid \forall m \in Y : gIm\}$ *for* $Y \subseteq M$*. A* formal concept *of the formal context* $\mathbb{K}$ *is a pair* (A, B) *with* $A \subseteq G$*,* $B \subseteq M$*,* $A = B'$*, and* $B = A'$*; the sets* A *and* B *are called the* extent *and the* intent *of the formal concept* (A, B)*. The subconcept-superconcept-relation is formalized by*

$$(A_1, B_1) \leq (A_2, B_2) :\Longleftrightarrow A_1 \subseteq A_2 \quad (\Longleftrightarrow B_1 \supseteq B_2).$$

The set of all formal concepts of $\mathbb{K}$ *together with the order relation* $\leq$ *is always a complete lattice, called the* concept lattice *of* $\mathbb{K}$ *and denoted by* $\underline{\mathfrak{B}}(\mathbb{K})$*.*

The concept lattices can be graphically represented by line diagrams which have been proven to be useful representations for the understanding of conceptual relationships in data. Before we illustrate this by examples, we introduce the notion of a *many-valued context* as a formalization of data tables that reports, for objects under consideration, specific values with respect to given attributes. In order to obtain a concept lattice of a many-valued context, the context has to be formally transformed to a formal context (also called a *one-valued context*). This transformation is performed by using *conceptual scales* which reflect specific interpretations of the data.

Definition 2. *A many-valued context is a set structure* $\mathbb{K} := (G, M, W, I)$ *where* G*,* M*, and* W *are sets and* I *is a ternary relation between* G*,* M*, and* W *(i.e.* $I \subseteq G \times M \times W$*) such that* $(g, m, w_1) \in I$ *and* $(g, m, w_2) \in I$ *always imply* $w_1 = w_2$*. The elements of* G*,* M*, and* W *are called objects, attributes, and attribute values, respectively, and* $(g, m, w) \in I$ *is read: "the object* g *has the attribute value* w *for the attribute* m*". An attribute* m *may be considered as a (partial) mapping from* G *to* W*; therefore,* $m(g) = w$ *is often written instead of* $(g, m, w) \in I$*. A conceptual scale for an attribute* $m \in M$ *is a one-valued context* $\mathbb{S}_m := (G_m, M_m, I_m)$ *with* $m(G) \subseteq G_m$*. The context* $\mathbb{R}_m := (G, M_m, J_m)$ *with* $gJ_m n :\Longleftrightarrow m(g)I_m n$ *is called the realized scale for the attribute* $m \in M$*. The*

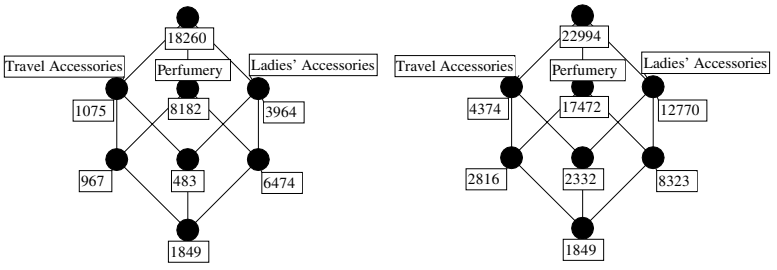


Fig. 1. Line diagrams showing the cross-selling between travel accessories, perfumery, and ladies' accessories

derived context *of* $\mathbb{K}$ *with respect to the conceptual scales* $\mathbb{S}_m := (G_m, M_m, I_m)$ $(m \in M)$*is the formal context* $(G, \bigcup_{m \in M} \{m\} \times M_m, J)$ *with* $gJ(m,n) :\Longleftrightarrow m(g)I_m n$*; its concept lattice is considered as the* concept lattice *of the many-valued context* $\mathbb{K}$ scaled *by the conceptual scales* $\mathbb{S}_m := (G_m, M_m, I_m)$ $(m \in M)$*. A many-valued context together with a collection of appertaining conceptual scales with line diagrams of their concept lattices is called a* conceptual data system.

Conceptual data systems can be implemented with the management system TOSCANA (see [29]). For a chosen conceptual scale, TOSCANA presents a line diagram of the corresponding concept lattice indicating all objects stored in the database in their relationships to the attributes of the scale, thus allowing users to navigate through the data and to analyze specific sets of objects by activating scales that interpret relevant aspects of the given data. Conceptual data systems stored in a database and implemented with a management system such as TOSCANA are called *conceptual information systems*.

In the following paragraphs, we illustrate how conceptual data analysis may be performed with a TOSCANA information system implemented to support the database marketing of a Swiss department store. The conceptual scales together with line diagrams of their concept lattices are derived from a database recording the activity of individual customers with respect to the various departments of the store. The analysis was undertaken to reveal potentials for cross-selling activities. For instance, to select the target group of a direct mail for promoting the ladies' wear department, one may start with unfolding the cross-selling behavior between departments where women typically buy.

The line diagram on the left side in Figure 1 shows the cross-selling behavior between travel accessories, perfumery, and ladies' accessories. The line diagram represents the concept lattice of the realized scale having as formal objects all customers with purchases in at least one of the three departments and having the three formal attributes 'purchased in travel accessories', 'purchased in perfumery', and 'purchased in ladies' accessories' while the binary relation records who bought in which department. The formal concepts of the realized scale are

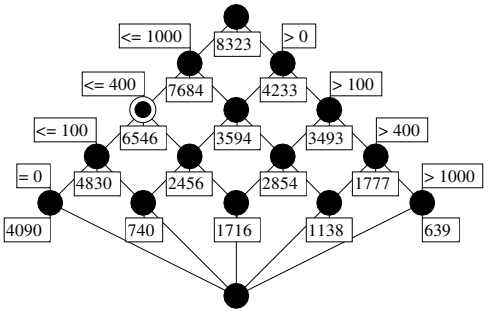


Fig. 2. Line diagram showing sales in women's clothing accrued by perfume and ladies' accessories customers

represented in the diagram by the little circles. The name of a formal object g is always attached to the circle representing the smallest concept with g in its extent (denoted by γg); dually, the name of a formal attribute m is always attached to the little circle representing the largest concept with m in its intent (denoted by μm). This labelling allows to read the context relation from the diagram because of $gIm \iff \gamma g \leq \mu m$, in words:

> *The object g has the attribute m if and only if there is an ascending path of line segments from the circle labelled with the name of g to the circle labelled with the name of m.*

The extent and intent of each concept (A, B) can also be recognized because $A = \{g \in G \mid \gamma g \leq (A, B)\}$ and $B = \{m \in M \mid (A, B) \leq \mu m\}$. The line diagrams in this paper show instead of the object names only the number of those names attached to the appertaining circle. Therefore, the diagram shows that there were 1075 customers who bought travel accessories only, 8182 perfumes only, and 3964 ladies' accessories only, but nothing in either of the other two departments. Furthermore, there were 967 customers who purchased travel accessories and something from perfumery but no ladies' accessories, and 1849 customers who were active in all three departments. From the diagram questions naturally arise, for example, why do 8182 customers buy perfumery goods but no travel or ladies' accessories even though both departments are right next to each other?

For the forementioned mailing select to promote sales in ladies' clothing, interesting are the 6474 + 1849 = 8323 customers because, in general, it is easier to develop active customers into better customers. The diagram on the right hand side in Figure 1 represents the same facts as the left one, but the number of customers are summed from the bottom up. To study the group of perfume and ladies' accessory buyers in further detail, TOSCANA allows users to "zoom into" the circle in the right diagram representing the 8323 customers who bought perfumery goods, ladies' accessories and, in some cases, travel accessories. Figure 2 shows a segmentation of those customers with respect to their previous

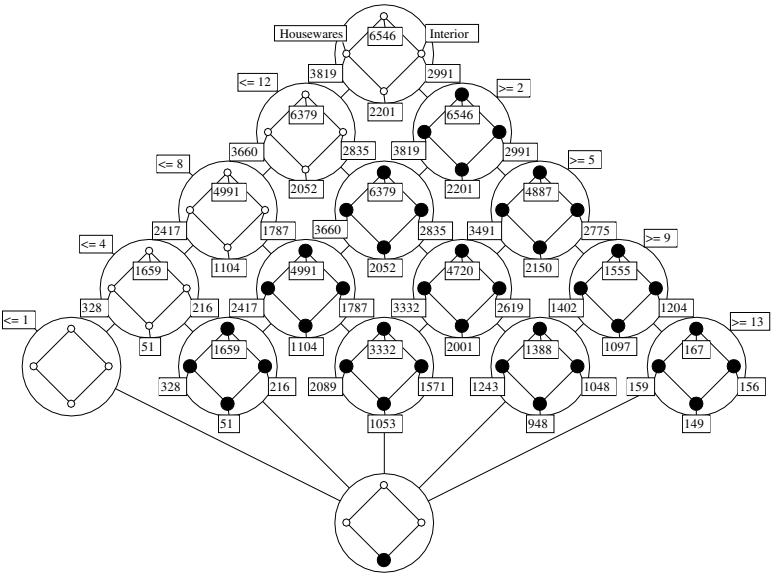


Fig. 3. Nested line diagram combining numbers of visited departments with the cross-selling between Housewares and Interior

activity in the ladies' wear department (formal, business, and casual wear). In this diagram, the number of customers are again summed from the bottom up; for instance, there are 1777 customers in the group of 8323 who spent more than 400 SFr for women's clothing, 639 who spent more than 1000 SFr, and 1138 who spent between 400 and 1000 SFr. The customers with low or no activity in ladies' wear were chosen as the targets of the mailing select, as the rest of the customers were identified as already being good ladies' wear customers.

In Figure 3 the activity of the 6546 customers with 400 or less sFr sales of women's clothing is shown. The *nested line diagram* presents two aspects of the activity of the 6546 customers: the line diagram representing the number of departments in which customers shopped (outer part) is combined with the cross-selling line diagram between housewares and interior (inner part). The circles of the first line diagram have been enlarged so that a copy of the second line diagram could be drawn in each enlarged circle. The nested line diagram can be read like an ordinary one if we replace the lines beween the large circles by parallel lines between the correspondeng circles of the inner diagrams. For instance, we can read from the diagram that there are 4720 customers who shopped in 5 or more but less than 13 departments of the store, and that 2001 of those bought housewares as well as interiors which seems to be a good target group for a direct mailing.

The examples should have made it clear that a TOSCANA information system enables an interactive and iterative process of conceptual data analysis lead-

ing to useful knowledge. The experiences with many TOSCANA systems have shown that domain experts are mostly stimulated by navigating through the graphical representations because they have a rich background knowledge about the appertaining domain and special interests for activating substantial questions. The process of knowledge discovery with TOSCANA systems is always accompanied by a learning process which increases the ability of the user to better understand the goals and possibilities of the specific exploration procedure. All these are reasons for viewing TOSCANA information systems as human-centered support of knowledge discovery, as Brachman and Anand advocated in [4].

3 From Data to Knowledge

In the previous section it is demonstrated through examples of conceptual data analysis how a conceptual information system may function as a *knowledge discovery support environment* that promotes human-centered discovery processes. In this section we want to explain in general the transition from data to knowledge for the discovery processes supported by a TOSCANA system. To clarify the transition steps from data (understood as symbolic representation of realities) to human knowledge, we call upon an analysis of knowledge representations in semantic networks performed by R. Brachman [3] who identified the following five representation levels (cf. [14]):

- *Implementational Level*: The primitives are nodes and links where links are merely pointers and nodes are simply destinations for links. On this level, there are only data structures from which logical forms can be build.
- *Logical Level*: The primitives are logical predicates, operators, and propositions together with a structured index over those primitives. On this level, logical adequacy is responsible for meaningfully prestructuring knowledge.
- *Epistemological Level*: The primitives are conceptual units, conceptual subpieces, inheritance and structuring relations. On this level, conceptual units are determined by their inherent structure and their interrelationships.
- *Conceptual Level*: The primitives are word senses and case relations, object- and action-types. On this level, small sets of language-independent conceptual elements and relationships are fixed and from which all expressible concepts can be constructed.
- *Linguistic Level*: The primitives are arbitrary concepts, words, and expressions. On this level, the primitives are language-dependent, and are expected to change in meaning as the network grows.

The grading of the levels, from implementational to linguistic, orders the representations from simple and abstract to complex and concrete; hence the grading should not misunderstood as a chronological ordering, although there are connections between the grading and the course of the transition from data to knowledge. In the following, the representation levels shall be characterized

according to their functionalities for supporting the process from data to knowledge as performed by a TOSCANA information system.

On the *implementational level*, the basic data structures are defined as one- and many-valued contexts. Already on this elementary level, there are instances for establishing connections to human knowledge, namely the formal objects, attributes, and attribute values of the contexts and the incidence relations between those elements. On this level, data contexts are merely considered as formal set structures without any content. Implementational issues for TOSCANA systems are discussed in [28] in detail.

On the *logical level*, names for the formal objects, attributes, attribute values, and incidence relations are formally taken as logical predicates which allow the composition of further predicates by logical connectives and quantifiers. Syntax and formal contextual semantics of those predicates have been elaborated to the so-called *Terminological Attribute Logic* (see [18],[11]) and *Terminological Concept Logic* (see [2]) which are both related to description logics. Both terminological logics may assist the formation of abstract scales for the methods of conceptual, relational, and logical scaling (see [17],[19]). The management system TOSCANA allows the activation of used logical expressions by representing them as SQL-queries. The combination of abstract scales to larger contexts is also performed on the logical level, namely by various context constructions; the mostly used context construction is the semiproduct which is basic for 'plain conceptual scaling' (see [9]), and the apposition which underlies the nested line diagrams used by TOSCANA as exemplified in Section 2 (see [29]).

The *epistemological level* addresses "the possibility of organizations of conceptual knowledge into units more structured than simple nodes and links or predicates and propositions" [3]. Formal concepts are indeed more internally structured than just a node or a predicate: they unify an object set (the extent) and an attribute set (the intent) so that each of these parts determines the other. Furthermore, the internal structure of the formal concepts gives rise to a conceptual hierarchy which mathematically forms a complete lattice if the formal concepts are those of a given formal context. Thus, the rich mathematical theory of Formal Concept Analysis (see [10]) yields a substantial contribution to Brachman's epistemological level. As Formal Concept Analysis is founded on lattice theory, lattice constructions and decompositions can be activated for establishing more complex concept hierarchies out of simpler ones, and, vice versa, for reducing complex concept hierarchies to simpler ones. Constructions like (sub-) direct products and tensor products of concept lattices and decompositions like subdirect and atlas decompositions have been successfully applied in data analysis and knowledge processing. For supporting the process of knowledge discovery, the visualization of concept lattices and their constructions and decompositions by specific line diagrams are of great importance. Those visualizations (also belonging to the epistomological level) are able to stabilize knowledge acquisition and communication (cf. [32]).

On the *conceptual level*, word senses are represented by the context attributes which lead to a contextual representation of concept intensions. As primitive case

relations, there are defined four basic relations: an object has an attribute, an object belongs to a concept, a concept abstracts to an attribute, and a concept is a subconcept of another concept (cf. [12]). These four relations are basic for the knowledge representation in conceptual information systems because, together with the word senses, they can represent a large amount of language-independent knowledge structures. Such structures are the *concrete scales* of TOSCANA systems which are used to capture the intensional content of an application domain (the extensional side of those scales are still abstract).

On the *linguistic level*, TOSCANA systems work with realized scales which are obtained by actualizing the abstract objects of their concrete scales according to real data. This realization particularly allows to deduce concept graphs representing verbal texts (see [20]). On this level, the knowledge representation is language-dependent so that users of the conceptual information system can best activate their background knowledge and common sense. The navigation through the conceptual landscape of the system, visualized by labelled line diagrams, can be performed successfully because the interplay between formal and material thinking stimulated by the diagrams gives purposeful orientations (cf. [35]).

The given characterization of the five representation levels for TOSCANA information systems shall now be used for explaining the discovery process from data to knowledge. This process can be seen in correspondence with the process of empirically grounded theory building proposed by A. Strauss and J. Corbin in [22] (see also [21]). According to Strauss and Corbin (p.57), empirically grounded theory building starts from data which are broken down, conceptualized, and put back together in new ways to generate a rich, tightly woven, explanatory theory that closely approximates the reality it represents. Although Strauss and Corbin are concentrating on theory building as the most systematic way of forming, synthesizing, and integrating scientific knowledge, their methodology may also apply to structuring and explaining the discovery process from data to knowledge in the more general case. This shall be outlined by means of the TOSCANA system discussed in the previous section.

The first step of breaking down the data is performed to establish the implementational level: the raw data are shaped to obtain elementary data structures which allow further formal treatments. In the case of our example, the raw data are coded in a relational database as a list of purchase transactions, each described by the ID number of the customer, the date, the department, and the purchase amount. From these data, suitable many-valued contexts are derived and represented in a data-warehouse as, for example, a many-valued context with the customers as formal objects structured by the many-valued attributes 'department', 'date', and 'purchase amount'. Establishing one- and many-valued contexts is a first move toward a conceptualization of the data.

The next step of conceptualization is, according to Strauss and Corbin, concerned with categorization. For TOSCANA systems, categorization is performed by methods of conceptual, relational, and logical scaling which, on the logical level, are only understood formally. In Figure 2, an example of a conceptual scale

is shown having formal attributes described by formal expressions which can be represented by SQL-queries in the management system TOSCANA. The apposition construction yielding the nested line diagram in Figure 3, which enlarges the attribute categorization, also belongs to the logical level.

The formal conceptualization is fully elaborated on the epistemological level. The concept lattices and the line diagrams as abstract structures are located on this level such as the formal procedures which make those lattices and diagrams to a successful support of knowledge acquisition and communication. The categorization leading to attributes of an abstract conceptual scale are now embedded into the significantly richer structure of the concept lattice of the scale which becomes human readable by a suitable line diagram. The richness of information given by such graphical representation may be seen in Figure 3; the nested structure shown in this figure reflects a subdirect product construction of the two combined concept lattices.

On the conceptual level the formal structures of the first three levels receive intensional meaning. For instance, the attribute names in Figure 1 are (on this level) understood by their literal meaning; thereby, the intensions of a represented concept can be described by combining all those meanings which belong to the attribute names attached to its superconcepts. Since the numbers in Figure 1 come from actual customers, they obtain their full meaning, discussed in Section 2, only on the linguistic level. On the conceptual level the concept lattices in Figure 1 represent a concrete scale which, according to Strauss and Corbin, may be understood as a intensionally determined dimension for the data to be analysed.

The full support for knowledge discovery is given on the linguistic level where the formal objects also carry meaning and, therefore, the formal concepts can unify intensional and extensional meaning. Of course, if further customers are considered in the presented example then the extensional meaning may change (although the intensional meaning of the concrete scales keeps the same). On this level, we can produce substantial interpretations of the data by suitable comparisions using nested line diagrams as in Figure 3; these diagrams correspond to the axial coding of Strauss and Corbin. Clearly, the rich, tightly woven, suggestive landscape of concept lattices that closely approximates the reality it represents, can serve through its representation by a TOSCANA information system, as a stimulating knowledge discovery support environment.

4 Procedures of Conceptual Knowledge Discovery

In most applications, classical *data analysis* and *decision support* facilities (for instance Online Analytical Processing (OLAP) or statistical packages) are already present when data mining tools are added to the knowledge discovery support environment. For supporting the analyst in the overall process of human-centered knowledge discovery, both decision support and data mining tools should provide a homogeneous environment. In particular, this shows the need of a *unified knowledge representation*. In conceptual information systems, concept lattices

are used as such a unified knowledge representation. TOSCANA information systems have shown their use for data analysis in over 30 implementations. The relationship between conceptual information systems and Online Analytical Processing is discussed in [23].

In the first part of this section, we show how data analysis and data mining techniques based on Formal Concept Analysis may support each other. In the second part, we go one step further: there, we present CHIANTI, a new tool that integrates data mining and data analysis in the framework of Conceptual Knowledge Discovery (CKDD).

4.1 Interplay of Data Analysis and Knowledge Discovery: Association Rules and Frequent Concept Lattices

In this subsection, we discuss how Formal Concept Analysis may support the mining of association rules, and how, vice versa, results of association rules mining may be used for decreasing the complexity of the visualization of traditional data analysis within conceptual information systems. *Association rules* are statements of the type '37 % of the customers buying coffee also buy milk'. The task of mining association rules is to determine all rules that have a certain *confidence* (37 % in the example) and a certain *support* (the percentage of customers buying coffee and milk). Mining association rules can nowadays be considered as one of the core tasks of KDD. Algorithmic aspects of mining association rules within the framework of Formal Concept Analysis are discussed in more detail in [15] and [30].

Improving the mining of association rules by using Formal Concept Analysis techniques. In terms of Formal Concept Analysis, the problem is the following: Let $\mathbb{K} := (G, M, I)$ be a formal context (for instance, G could be the set of transactions registered during a certain time period in the department store, M the set of products (or items) sold by the store, and $(g, m) \in I$ means that item m was purchased in transaction g). Each subset X of M is called an *itemset*. The *support* of X is defined by $\text{supp}(X) := \frac{|X'|}{|G|}$. An *association rule* $X \to Y$ consists of two subsets X and Y of M. We say that the rule $X \to Y$ holds with *support* $\text{supp}(X \to Y) := \frac{|(X \cup Y)'|}{|G|}$ and with *confidence* $\text{conf}(X \to Y) := \frac{\text{supp}(X \cup Y)}{\text{supp}(X)}$ (in short: $X \xrightarrow{s,c} Y$ with $s := \text{supp}(X \to Y)$ and $c := \text{conf}(X \to Y)$). The task is now to compute, for given *minsupp*, *minconf* $\in [0, 1]$, all association rules $X \xrightarrow{s,c} Y$ with $s \geq$ *minsupp* and $c \geq$ *minconf*.

The notion of association rules and their application to large databases was introduced by R. Agrawal, T. Imielinski, and A. Swami in [1]. They stated the problem and provided a first algorithm. Now there are several algorithms for mining association rules in the literature, see for instance [15] for details.

Rules that hold only with a certain confidence have been investigated before by many researchers. For instance, in the framework of Formal Concept Analysis, M. Luxenburger [13] has called them *partial implications*. They are a

generalization of *implications* which play an important role in Conceptual Data Analysis based on Formal Concept Analysis. Implications are association rules which hold for all objects but have no restriction on the support, i. e., they are exactly the association rules with $minconf = 1$ and $minsupp = 0$.

One problem in presenting the mined association rules to the user is that they usually form a long list, from which only very few are of interest to the domain expert. Using the following theorem ([15,26]) one can reduce the list without losing any information:

Theorem. Let $X, Y \subseteq M$. Then $X \overset{s_1,c_1}{\longrightarrow} Y$ and $X'' \overset{s_2,c_2}{\longrightarrow} Y''$ have the same support and the same confidence.

It is based on the fact that, for any frequent itemset Y, the smallest concept intent which contains Y (i. e., Y'') has the same support and hence is also frequent. For the development of algorithms, this property permits the consideration of only concept intents (instead of all itemsets) for determining the set $\mathcal{F}$ of frequent itemsets [15,30]. Especially in strongly correlated data, the algorithm can thereby skip many itemsets.

Using the theorem, one can present a significantly shorter list of association rules without loosing any information. The list is composed of the so-called *Duquenne-Guigues basis for exact association rules* and the *Luxenburger basis for approximate association rules*. Both bases are introduced in [30], together with algorithms for their computation.

Reducing the complexity of data visualization in conceptual information systems by using results from association rule mining. For examining cross-selling (cf. Section 2), the concepts having many attributes – and hence only relatively few objects! – are of special importance. In those cases, one needs the whole line diagram for an analysis of how well cross-selling works. But there are many applications where concepts which differentiate the population too much are not interesting – at least not for a first overview. In that situation, frequent concepts, as defined above, can be utilized. By fixing a threshold *minsupp*, all infrequent concepts of the conceptual scale can be pruned. Then, only the frequent concepts are displayed. For instance, if we want to have a first glance at the distribution of the age of the customers, then the conceptual scale 'Age' may be too detailed. By fixing $minsupp := 25\%$, we prune 18 of the 30 concepts of the scale 'Year of Birth'. The remainder is shown in Figure 4. Two facts can be easily seen a) the birthyear of more than half the credit card customers is unknown, and b) 4690 of all credit card customers were born before 1973. Hence, there are very few customers with a known birthyear who are younger than 25 and have paid with a credit card.

4.2 Integration of Data Analysis and Knowledge Discovery: Guided Learning

In the expression *supervised learning* (as a task of *Machine Learning*), 'learning' is used in a metaphorical way. One expects the software to find an intensional

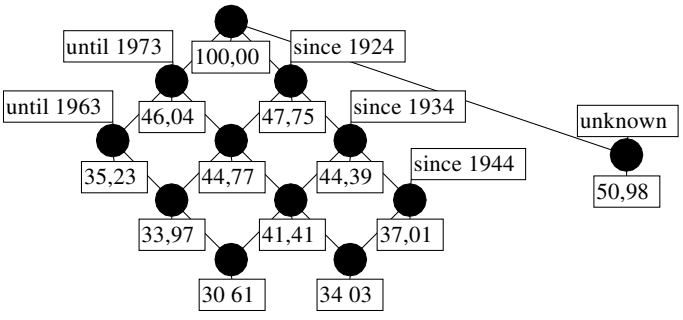


Fig. 4. Conceptual Scale 'Year of Birth' restricted to frequent concepts with *minsupp* = 25%

description of some subpopulation, based on a training set. As CKDD is seen as a human-centered knowledge discovery process, our aim is to support the learning process (in its literal meaning) of a human expert. Human knowledge always relies on background knowledge which is formed by intersubjective argumentation, and only part of this knowledge can be expressed explicitly. Knowledge which can be made explicit may be treated by procedures of Machine Learning. But if one considers *all* aspects of knowledge, then it becomes clear that learning can only be supported by a knowledge discovery environment, but can never be completely automated.

In this setting, we understand *guided learning* as a technical support for the learning process of the human expert.[1] Guided learning shall automatically lead the user to conceptual scales (or combinations of conceptual scales) which are expected to provide interesting information, combined with the freedom of navigating around. As in supervised learning, the problem we tackle is to gain more knowledge about a given subpopulation. The difference is that we do not necessarily require an *explicit* description of the behavior. For instance, we might want to learn (in its literal meaning) more about the differences in buying behavior between high- and low-spending credit card customers.

For this purpose, we have developed the new tool CHIANTI, based on [24] and [25]. CHIANTI takes as input two subpopulations which are defined by SQL queries. In the following example, we have divided the population in two parts: those customers who spent more than 1000 SFr and those who spent less. This tool compares the distribution of the two subpopulations in all scales of the conceptual information system and returns a ranking of all scales. In the ranking, the scales which appear at the top are those where the distribution differs the most. The current implementation of CHIANTI provides two measures for the distribution: The χ^2-measure (hence the name of the program) and the maximum norm. While the first measure takes the differences in all concepts into

[1] The expression 'guided learning' is also used for education and training software, but here we use it to show the analogy to supervised learning.

Skala	Wert
Xselling Housewares/Interior	0,0684
Xselling Food/Wine	0,0570
Xselling Travel Access./Perfumery/Ladies' Accessories	0,0325
Xselling Perfumery/Housewares/Food	0,0324
Xselling Perfumery/Ladies' Fashion	0,0305
Xselling Wine/Men's Fashion/Perfumery	0,0275
Xselling Ladies' Fashion/Men's Fashion/Sports	0,0229
Xselling Sports/Children/Travel Accessories	0,0160
Xselling Men's Clothing (incl. Underwear)	0,0160
Xselling Ladies' Wear	0,0123
Xselling Men's City	0,0009

Fig. 5. Ranking of conceptual scales related to cross-selling

account (the larger ones over proportionally), the second measure only regards the concept with the largest difference. This approach is useful when an easy interpretation of the ranking is desired. At the moment, CHIANTI only works on the contingents (this means that, for the measure, the cardinality of the concept extents is not used, only the number of objects which generate the concept). As the difference of the distributions of the two populations may be more significant in more general concepts (which are not necessarily generated by single objects), the next version of CHIANTI will also analyze concept extents.

Figure 5 shows the ranking of all scales related to cross-selling for the two subpopulations mentioned above with the χ^2-measure. The scale at the top is the scale 'Cross-selling houseware/interieur' which we have already seen as inner scale in Figure 3. This means that among all cross-selling scales, this scale differentiates the two groups the most. The scale 'Cross-selling Housewares/Interior' also appears as topmost scale in the ranking according to the maximum norm.

By combining the topmost scales with the scale 'Money spent $\leq$ / $>$ 1000 SFr' we can analyze the distribution of the two groups in more detail. The combination of this scale together with the scale 'Cross-selling Housewares/Interior' is shown in Figure 6. In the diagram, we have set the top element of each inner scale to 100% in order to facilitate comparison. We see that the high-spending customers buy over-proportionally in the departments Housewares (265% more often) and Interior (322% more often). Furthermore, for this customer group, the cross-selling between both departments is much higher than for the rest: The percentage of high-spending customers who were active in both interior and housewares (36.98%) is much greater than that of low-spending customers (5.56%).

We emphasize that — unlike many other statistical techniques — the ranking of the scales is not the final result, but a suggestion to the analyst of certain combination of scales for analyzing the situation in more detail. The ranking alone does not indicate that the buying behavior in the housewares department determines the value of the customer. In particular, it is not possible to decide automatically if a prominent position in the ranking indicates a cause for or a

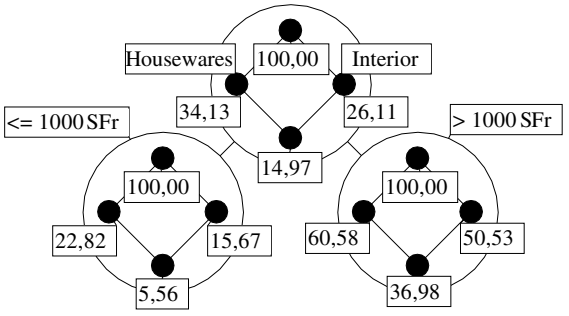


Fig. 6. Customers of the Housewares department differentiated by the amount of money spent

consequence of the different distribution, as is clearly demonstrated by studying the ranking of all the scales. The topmost scales are then all scales related to the amount of money spent. In those scales, one will hardly discover new insights. The next scale is then 'Active Time (in days)'. This scale does not provide an interesting insight either, since it is intuitively clear that a typical customer usually spends less than 1000 SFr in a single transaction; hence to spend more money, he has to visit the department store more than once. The next scale then is the scale 'Cross-selling Housewares/Interior'.

The insight that the scale about the active time is not useful for this kind of analysis can only be gained by referring to the implicit background knowledge of the domain expert. A repository which stores such information explicitly cannot overcome the general problem. There is an almost boundless number of possible combinations of conceptual scales in a conceptual information system which cannot be conceived of in advance. However, it is promising for further research to consider such a repository which 'learns' (in the metaphorical meaning) from the behavior of the analyst which combinations are of interest and which are not.

References

1. R. Agrawal, T. Imielinski, A. Swami: Mining association rules between sets of items in large databases. *Proc. ACM SIGMOD*, 1993.
2. H. Berg: *Terminologische Begriffslogik.* Diplomarbeit. FB Mathematik, TU Darmstadt 1997.
3. R. J. Brachman: On the epistemological status of semantic networks. In: N. V. Findler (ed.): *Associative networks: representation and use of knowledge by computers.* Academic Press, New York 1979, 3–50.
4. R. J. Brachman, T. Anand: The process of knowledge discovery in databases. In [7]
5. R. J. Brachman, P. G. Selfridge, L. G. Terveen, B. Altman, A. Borgida, F. Halper, T. Kirk, A. Lazar, D. L. McGuinnes, L. A. Resnick: Integrated Support for Data Archaeology. *International Journal of Intelligent and Cooperative Information Systems* 2 (1993), 159–185.

6. J.-L. Guigues, V. Duquenne: Familles minimales d' implications informatives resultant d' un tableau de données binaires. *Math. Sci. Humaines* **95**, 1986, 5–18.
7. U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, R. Uthurusamy (eds.): *Advances in Knowledge Discovery and Data Mining.* AAAI/MIT Press, Cambridge 1996.
8. B. Ganter: Algorithmen zur Formalen Begriffsanalyse. In: B. Ganter, R. Wille, K. E. Wolff (eds.): *Beiträge zur Begriffsanalyse.* B.I.-Wissenschaftsverlag, Mannheim 1987, 241–254.
9. B. Ganter, R. Wille: Conceptual scaling. In: F. Roberts (ed.): *Applications of combinatorics and graph theory to the biological and social sciences.* Springer, Berlin-Heidelberg-New York 1989, 139–167.
10. B. Ganter, R. Wille: *Formal Concept Analysis: Mathematical Foundations.* Springer, Berlin-Heidelberg 1999 (Translation of: *Formale Begriffsanalyse: Mathematische Grundlagen.* Springer, Berlin-Heidelberg, 1996).
11. B. Ganter, R. Wille: Contextual Attribute Logic. *Proc. ICCS '99*, LNAI **1640**, Springer, Heidelberg 1999, 377–388
12. P. Luksch, R. Wille: A mathematical model for conceptual knowledge systems. In: H.-H. Bock, P. Ihm (eds.): *Classification, data analysis, and knowledge organization.* Springer, Berlin-Heidelberg 1991, 156–162.
13. M. Luxenburger: Implications partielles dans un contexte. *Mathématiques, informatique et sciences humaines* **113**, 1991, 35–55.
14. G. Mineau, G. Stumme, R. Wille: Conceptual Structures Represented by Conceptual Graphs and Formal Concept Analysis. *Proc. ICCS '99*, LNAI **1640**. Springer, Heidelberg 1999, 423–441
15. N. Pasquier, Y. Bastide, R. Taouil, L. Lakhal: Efficient mining of association rules using closed itemset lattices. *Journal of Information systems*, **24** (1999), 25–46.
16. Ch. S. Peirce: *Collected Papers.* Harvard University Press, Cambridge 1931-35.
17. S. Prediger: Logical scaling in formal concept analysis. In: D. Lukose, H. Delugach, M. Keeler, L. Searle, J. F. Sowa (eds.): *Conceptual Structures: Fulfilling Peirce's Dream.* LNAI 1257. Springer, Berlin-Heidelberg-New York 1997, 332–341.
18. S. Prediger: Terminologische Merkmalslogik in der Formalen Begriffsanalyse. In: G. Stumme, R. Wille (eds.): *Begriffliche Wissensverarbeitung: Methoden und Anwendungen.* Springer, Berlin-Heidelberg 2000, 99–124.
19. S. Prediger, G. Stumme: Theory-Driven Logical Scaling. *Proc. KRDB '99.* (Also in *Proc. DL '99*). CEUR Workshop Proc. **21**+**22**, 1999 (http://SunSITE.Informatik.RWTH-Aachen.DE/Publications/CEUR-WS/)
20. S. Prediger, R. Wille: The lattice of concept graphs of a relationally scaled context. FB4-Preprint, TU Darmstadt 1999.
21. S. Strahringer, R. Wille, U. Wille: Mathematical support for empirical theory building. FB4-Preprint, TU Darmstadt 1999.
22. A. Strauss, J. Corbin: *Basics of qualitative research: grounded theory procedures and techniques.* Sage Publ., Newbury Park 1990.
23. G. Stumme: On-Line Analytical Processing with Conceptual Information Systems. *Proc. 5th Intl. Conf. on Foundations of Data Organization*, 12.–13. November 1998, 117–126 (to be published by Kluwer)
24. G. Stumme: Exploring Conceptual Similarities of Objects for Analyzing Inconsistencies in Relational Databases. *Proc. Workshop on Knowledge Discovery and Data Mining, 5th Pacific Rim Intl. Conf. on Artificial Intelligence.* Singapore, Nov. 22–27, 1998, 41–50.
25. G. Stumme: Dual Retrieval in Conceptual Information Systems. in: A. P. Buchmann (ed.): *Datenbanksysteme in Büro, Technik und Wissenschaft.* Springer, Heidelberg 1999, 328–342

26. G. Stumme: Conceptual Knowledge Discovery with Frequent Concept Lattices. FB4-Preprint, TU Darmstadt 1999
27. G. Stumme, R. Wille, U. Wille: Conceptual Knowledge Discovery in Databases Using Formal Concept Analysis Methods. In: J. M. Żytkow, M. Quafofou (eds.): *Principles of Data Mining and Knowledge Discovery.* Proc. of the 2nd European Symposium on PKDD '98, Lecture Notes in Artificial Intelligence **1510**, Springer, Heidelberg 1998, 450–458.
28. F. Vogt: *Formale Begriffsanalyse mit C++: Datenstrukturen und Algorithmen.* Springer, Berlin–Heidelberg–New York 1996.
29. F. Vogt, R. Wille: TOSCANA – a graphical tool for analyzing and exploring data. In: R. Tamassia, I. G. Tollis (eds.): *Graph Drawing '94.* Lecture Notes in Computer Science 894. Springer, Berlin-Heidelberg-New York 1995, 226-233.
30. R. Taouil, Y. Bastide, N. Pasquier, G. Stumme, L. Lakhal: Mining bases for association rules based on Formal Concept Analysis. *Proc. ECAI 2000* (submitted)
31. R. Wille: Concept Lattices and Conceptual Knowledge Systems. *Computers & Mathematics with Applications*, *23*, 1992, 493-515.
32. R. Wille: Begriffliche Datensysteme als Werkzeug der Wissenskommunikation. In H. H. Zimmermann, H.-D. Luckhardt, A. Schulz (eds.): *Mensch und Maschine – Informationelle Schnittstellen der Kommunikation.* Univ.-Verl. Konstanz, 1992, 63–73.
33. R. Wille: Plädoyer für eine philosophische Grundlegung der Begrifflichen Wissensverarbeitung. In: R. Wille, M. Zickwolff (eds.): *Begriffliche Wissensverarbeitung: Grundfragen und Aufgaben.* B.I.-Wissenschaftsverlag, Mannheim 1994, 11–25.
34. R. Wille: Begriffsdenken: Von der griechischen Philosophie bis zur Künstlichen Intelligenz heute. Dilthey-Kastanie, Ludwig-Georgs-Gymnasium Darmstadt 1995, 77–109.
35. R. Wille: Conceptual Landscapes of Knowledge: A Pragmatic Paradigm for Knowledge Processing. In: *Proc. of KRUSE '97.* Vancouver, August 11–13, 1997, 2–13.

CEM – A Conceptual Email Manager

Richard Cole[1] and Gerd Stumme[2]

[1] School of Information Technology, Griffith University, Gold Coast Campus, PMB 50, Gold Coast Mail Centre QLD 9726, Australia; r.cole@gu.edu.au
[2] Technische Universität Darmstadt, Fachbereich Mathematik, Schloßgartenstr. 7, D–64289 Darmstadt, Germany; stumme@mathematik.tu-darmstadt.de

Abstract. CEM is an email management system which stores its email in a concept lattice rather than in the usual tree structure. By using such a conceptual multi-hierarchy, the system provides more flexibility in retrieving stored emails. The paper presents the underlying mathematical structures, discusses requirements for their maintenance and presents their implementation.

1 Motivation

The way standard email management systems store mails is directly derived from the tree structure of file management systems. This has the advantage that trees have a simple structure which can easily be explained to novice users. The disadvantage is that at the moment of storing an email the user already has to foresee the way she is going to retrieve the mail later. The tree structure forces her to decide at that moment which criteria to consider as primary and which as secondary. For instance, when storing an email regarding the organization of a conference, one has to decide whether to organize one's directories like `mineau/iccs2000/program_commitee` or like `conferences/iccs/iccs2000/organisation/mineau`. This problem arises especially if a user cooperates with overlapping communities on different topics.

In this paper, we present the *Conceptual Email Manager CEM*. It uses a formal context as its structure for storing email rather than a tree. This allows the user to retrieve emails via a concept lattice following different paths. For the example above this means that one need not decide which of the two paths to use for storing. For retrieving the mail later, one can consider any combination of the catchwords[1] in the two paths.

Concept lattices are defined in the mathematical theory of Formal Concept Analysis [12]. A concept lattice is derived from a binary relation which assigns attributes to objects. In our application, the objects will be all emails stored by the system, and the attributes will be catchwords like 'conferences', 'mineau', and 'organisation'. We assume the reader to be familiar with the basic notions of

[1] By catchwords we mean small natural language phrases under which the user may meaningfully classify documents.

B. Ganter and G.W. Mineau (Eds.): ICCS 2000, LNAI 1867, pp. 438–452, 2000.
© Springer-Verlag Berlin Heidelberg 2000

Formal Concept Analysis, and refer otherwise to [3] and to proceedings of past ICCS conferences.

There are related approaches to the above stated problem. For instance the concept of a *virtual folder* was introduced in a program called View Mail (VM) [6]. A virtual folder is simply a collection of email documents retrieved in response to a query. The virtual folder concept has more recently been popularized by a number of open source projects, e. g. [8]. Our system differs from those projects in both the understanding of the underlying structure via formal concept analysis, and the implementation.

Our approach is also related to the library information system implemented in the Center of Interdisciplinary Studies at Darmstadt University of Technology [7]. That system is based on the management system TOSCANA for Conceptual Information Systems [11]. The retrieval component of both our system and the library system provide basically the same functionality. The difference lies in the support for the user maintaining and updating the email collection. This is due to the fact that, while in the library system maintenance is allowed only to the librarian and/or a knowledge engineer, in an email management system storing emails is an essential and often used feature which requires some semi-automatic support for an untrained user.

In the next section, we will describe the mathematical structures of the Conceptual Email Manager. Requirements for their maintenance are discussed in Section 3. Issues related to an implementation of the requirements are discussed in Section 4. The paper is concluded by an outlook on future work.

In this paper we endeavor to precisely define the behavior of a natural user interface for managing emails based on Formal Concept Analysis. Although designing the interface to exhibit simple and rational behavior to the user, the exact semantics with respect to the underlying program structures the reader will find are rather detailed.

2 Structures Underlying CEM

We assume that the reader is familiar with the following two basic notions of Formal Concept Analysis: formal context and concept lattice. Definitions and examples can be found in [3] or in previous ICCS proceedings.

In this section, we describe the system on a structural level; we abstract from implementation details. They will be discussed in Section 3. Basically, we can distinguish three fundamental structures:

1. A *formal context* which assigns to each email a set of catchwords;
2. a *hierarchy* on the set of catchwords in order to define an information ordering over the catchwords;
3. and a mechanism for creating *conceptual scales* which are used within a graphical interface for the retrieval of emails.

These three structures are discussed in detail in the remainder of this section.

2.1 Assigning Catchwords to Emails

In the conceptual email manager, we use a *formal context* (G, M, I) for storing the emails and for assigning catchwords to them. The set G contains all emails stored in the system, the set M contains all catchwords. For the moment we consider M to be unstructured. (In the next subsection however, we will introduce a hierarchy on it.)

The relation I indicates which emails are assigned to which catchwords. In the example given in the introduction, the user might want to assign all the catchwords 'mineau', 'iccs2000', 'program_commitee', 'conferences', 'iccs', and 'organisation' to the new email. The incidence relation is generated in a semi-automatic process: (i) an automatic string-search algorithm may recognize words within sections of an email and suggest relations between the email and some attributes, (ii) the user may accept the suggestion or modify it, and (iii) she also may attach user defined attributes to the email. In Section 3, we will discuss how the user is supported in this assignment process. At the moment, we suppose that the relation is already given.

Instead of a tree of disjoint folders and sub-folders, we consider the concept lattice $\mathfrak{B}(G, M, I)$ as navigation space. The formal concepts replace the folders. In particular, this means that emails can appear in different concepts. The most general concept contains all emails. The deeper the user gets in the hierarchy, the more specific are the concepts, i. e., the smaller is the number of emails they contain. Even so the user may, using general catchwords only, still obtain a great search depth from the conjunctions present in the concept lattice.

2.2 A Hierarchy on the Catchwords

In order to support the semi-automatic assignment of catchwords to the emails, we additionally provide the set M of catchwords with a partial order $\leq$. For this *subsumption hierarchy*, we assume that the following *compatibility condition* holds:

$$\forall g \in G,\ m, n \in M\colon \quad (g, m) \in I,\ m \leq n \ \Rightarrow\ (g, n) \in I \qquad (\ddagger)$$

(i. e., the assignment of catchwords to emails respects the hierarchy on the catchwords). Hence for assigning catchwords to emails, it is sufficient to assign the most specific catchwords only. All more general catchwords will be added automatically by the system. The maintenance of the hierarchy will be discussed in the two following sections.

As an example, the user may want to say that 'iccs' is a more specific catchword than 'conferences', and that 'iccs2000' is more specific than 'iccs' (i. e., 'iccs2000'≤'iccs'≤'conferences'). Emails regarding the production of this paper are then assigned by the authors to the catchword 'iccs2000' only (and maybe additionally to catchwords like 'cole' or 'stumme', and to 'papers'). When the authors want to retrieve these emails, they do not need to remember that they stored them under 'iccs2000'. They will also find them under the more general catchword 'conferences'. If this catchword provides a list of emails that is too

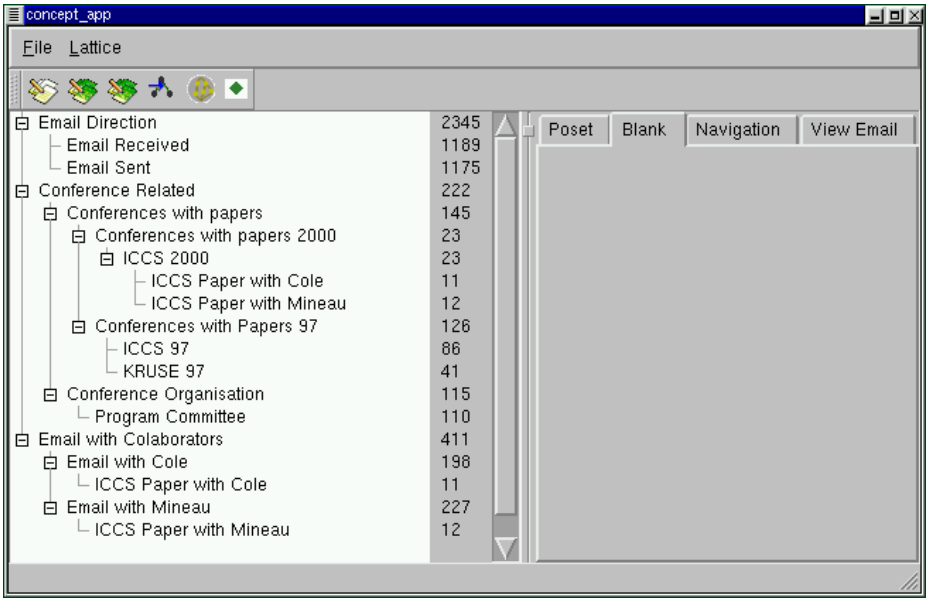


Fig. 1. Part of a catchword hierarchy

long, then they can either refine the search by taking a sub-term like 'iccs' or alternatively by adding another catchword, for instance 'cole'. The next subsection describes the structures which support the user in this kind of navigation.

While we note that it is not required by the theory that a particular structure be imposed on the hierarchy it is likely that the user will impose some structural notions on $(M, \leq)$. One appealing and natural notion is to split the hierarchy into three parts: One part related to contents of the emails, e. g., if an email is related to a conference or not, if it is used for its organization, etc. A second part related to the sender or receiver of the email. And a third part describing aspects of the mailing process (whether it is an inbound or an outbound mail etc.). An example of a hierarchy is given in Figure 1. (The right window of the screenshot is explained in Section 4.)

Even when the hierarchy imposed on the catchwords by the user is a tree, the resulting concept lattice — which we use as the search space — is by no means a forest. Consider for example the concept generated by the conjunction of the two catchwords 'ICCS 2000' and 'conference organization'. It will have at least two incomparable super-concepts, namely the one generated by the catchword 'ICCS 2000' and the one generated by the catchword 'conference organization'. In general, all we know is that the resulting concept lattice is embedded as a join-semilattice in the lattice of all order ideals of $(M, \leq)$ (i. e., all subsets X of M s. t. $x \in X$ and $x \leq y$ imply $y \in X$). [2]

[2] The use of this structure in the framework of knowledge discovery in databases is analyzed in more detail under the name of *power scale* in [5]. Refer also to the theorem of Birkhoff (stated for instance in [3, Theorem 39]).

2.3 Conceptual Ccales for Navigating through the Set of Emails

Conceptual scaling has been introduced in order to deal with many-valued attributes. Often attributes are not one-valued, as for instance with the catchwords given above, but instead allow a range of values. This is modeled by a *many-valued context*. A many-valued context is roughly equivalent to a relation of a relational database with one field being a primary key. As one-valued contexts are special cases of many-valued contexts, conceptual scaling can also be applied to one-valued contexts in order to reduce the complexity of the visualization.

In this paper, we only deal with one-valued formal contexts. Readers who are interested in the exact definition of many-valued contexts and the use of conceptual scaling in this more general case are referred to [3]. Applied to one-valued contexts, conceptual scales are used to determine the concept lattice which arises from one vertical 'slice' of a large context:

Definition 1. *A* conceptual scale *for a subset $B \subseteq M$ of attributes is a (one-valued) formal context $\mathbb{S}_B := (G_B, B, \ni)$ with $G_B \subseteq \mathfrak{P}(B)$. The scale is called* consistent *with respect to $\mathbb{K} := (G, M, I)$ if $\{g\}' \cap B \in G_B$ for each $g \in G$. For a consistent scale $\mathbb{S}_B$, the context $\mathbb{S}_B(\mathbb{K}) := (G, B, I \cap (G \times B))$ is called its* realized scale.

Conceptual scales are used to group together related attributes. They are determined as required by the user, and the realized scales are derived from them when a diagram is requested by the user.

The Conceptual Email Manager stores all scales which the user has defined in previous sessions. To each scale, she can assign a unique name. This is modeled by a mapping.

Definition 2. *Let $\mathcal{S}$ be a set, whose elements are called* scale names. *The mapping*

$$\alpha: \mathcal{S} \to \mathfrak{P}(\mathcal{M})$$

defines for each scale name $s \in \mathcal{S}$ a scale $\mathbb{S}_s := \mathbb{S}_{\alpha(s)}$.

For instance, the user may introduce a new scale which classifies the emails according to being related to a conference by adding a new element 'Conference' to $\mathcal{S}$ and by defining $\alpha(\text{Conference}) := \{\text{CKP '96, AA 55, KLI '98, Wissen '99, ICCS 2000}\}$.

Observe that $\mathcal{S}$ and M need not be disjoint. This allows for instance the following construction which deduces conceptual scales directly from the subsumption hierarchy: Let $\mathcal{S} := \{m \in M \mid \exists n \in M : n < m\}$, and define, for $s \in \mathcal{S}$, $\alpha(s) := \{m \in M | m \prec s\}$ (with $x \prec y$ if and only if $x < y$ and there is no z s. t. $x < z < y$). This means that all catchwords $m \in M$ which are neither minimal nor maximal in the hierarchy are at the same time considered as the name of scale $\mathbb{S}_m$ and as catchword of another scale $\mathbb{S}_n$ (where $m \prec n$). In this paper, we will call scales constructed this way *default scales*.

This last construction has first been presented in [10] for defining a hierarchy of conceptual scales for the library information system [7]. In [10], however, only

this special construction was considered. It turns out that, in general, a more flexible construction is desirable. In the library information system, for instance, one is also interested in scales for the minimal elements in $(M, \leq)$. Each such scale $\mathbb{S}_m$ has as attributes the upper covers of m (i. e., all $n \in M$ with $m \prec n$). This construction is made possible by using the function α which we have introduced in this paper.

3 Requirements of the Conceptual Email Manager

In this section, we discuss requirements of a conceptual email manager based on the paradigm of Formal Concept Analysis. In the following section we shall explain how our implementation responds to these requirements.

The requirements may be divided along the same lines as the underlying mathematical structures defined in Section 2. Briefly stated the requirements are:

1. to assist the user in building, browsing and modifying the catchword hierarchy;
2. to help the user modify the scale function α;
3. to allow the user to manage the assignment of catchwords to email documents; and
4. to assist the user in searching the conceptual space of emails for both individual emails, and also conceptual groupings of emails.

In addition to the requirements stated above, a good email system needs to be able to send, receive and display emails by processing the various email formats and interacting with the current popular email protocols. Since these requirements are already well understood and implemented by existing email programs they will not be discussed further in detail in this paper.

Browsing and Modifying the Catchword Hierarchy. The catchword hierarchy is a partially ordered set $(M, \leq)$ where each element of M is a catchword. Listed below are requirements related to browsing and modifying of the catchword hierarchy.

1. The program should display graphically the structure of the partial order $(M, \leq)$. The ordering relation must be clearly evident to the user.
2. It must be possible, via a series of graphical manipulations initiated by the user and implemented in the program to add and to delete elements and to alter the ordering relation. It should be possible to create any partial order within a reasonable size limit.

Modifying the Scale Function. The user must be able to modify the scale function α, explained in Section 2. Therefore the tool should provide a suitable visualization of the function. The program must allow an overlap between the set $\mathcal{S}$ of scale names, and the set M of catchwords.

Managing the Assignment of Catchwords to Emails. The program should store the formal context (G, M, I) and ensure that the compatability condition (‡) is always satisfied. It is inevitable that the program will have sometimes to modify the formal context, after a change is made to the catchword hierarchy, in order to satisfy the compatability condition. This modification can be made either automatically, or via an interactive process where the user is asked whether the changes should be made.

The program must support two mechanisms for the association of catchwords to emails. Firstly there should be a mechanism as described in Section 2.1 by which emails are semi-automatically associated with catchwords based on the email content. Secondly the user should be able to view and modify the association of catchwords with emails.

Navigating the Conceptual Space. The program should assist the navigation of the conceptual space of the emails by drawing line diagrams of concept lattices arising from conceptual scales [3]. These line diagrams should extend to locally nested line diagrams [9,10]. The program must allow the retrieval and viewing of emails that form the extension of concepts displayed in these line diagrams.

4 Implementation

This section divides the description of the implementation of our conceptual email manager, CEM, into a structure similar to that presented in Section 3.

4.1 Catchword Hierarchy

Browsing the Hierarchy. The user is presented with a view of the hierarchy, $(M, \leq)$ as a tree widget,[3] shown in Figure 1. The tree widget has the advantage that most computer users are familiar with its operation, and that it provides a compact representation (in the sense of space used on the screen) of a tree structure.

The catchword hierarchy, being a partially ordered set, has a more general structure than that of a tree. No limitation is placed by the program on the structure of the partial order in general. Following is a definition of the tree derived from the catchword hierarchy with the purpose of defining the contents and structure of the tree widget.

Let $(M, \leq)$ be a partially ordered set and denote the set of all sequences of elements from M by M^* (including the empty sequence ε). Then the labeled tree derived from the catchword hierarchy is comprised by $(T, \sqsubseteq, \texttt{label})$ where $T := \{(m_1, \ldots, m_n) \in M^* \mid m_i \prec m_{i+1},\ m_n \in \max(M)\} \cup \{\varepsilon\}$, $w_1 \sqsubseteq w_2$

[3] A widget is a graphical user interface component with a well defined behaviour usually mimicking some physical object.

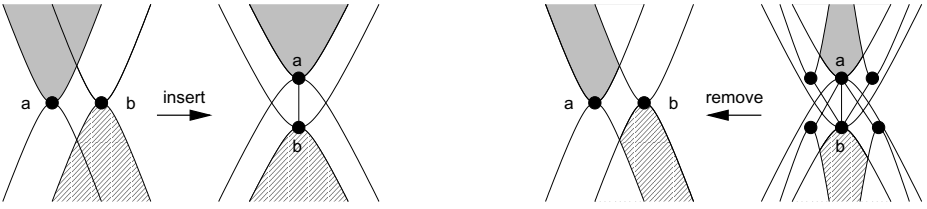


Fig. 2. `Insert` and `removal ordering` operation

iff w_2 is a suffix of w_1, and $\texttt{label}{:}T \setminus \{\varepsilon\} \to M$ is the function defined by $\texttt{label}(m_1, \ldots, m_n) := m_n$.

Each tree node is identified by a path from a catchword to the top of the catchword hierarchy. Although the tree representation has the disadvantages that elements from the partial order occur multiple times in the tree and that the tree can become large, the saving of space and the regular structure are our reasons to prefer it to other order representations. If the user keeps the number of elements with multiple parents in the partial order to a small number then the tree is manageable.

Modifying the Hierarchy $(M, \leq)$. The program provides four operations for modifying the hierarchy: `insert catchword`, `remove catchword`, `insert ordering` and `remove ordering`. More complex operations provided to the user, for example moving an item in the taxonomy, are resolved internally to sequences of these four operations. In this section we denote the order filter (also called the up-set) of m as $\uparrow m := \{x \in M \mid m \leq x\}$, the order ideal (also called the down-set) of m as $\downarrow m := \{x \in M \mid x \leq m\}$, the lower cover of m as $\prec_m := \{x \in M \mid x \prec m\}$, and the upper cover of m as $\succ_m := \{x \in M \mid x \succ m\}$.

The operation `insert catchword` simply adds a new catchword to M, and leaves the $\leq$ relation unchanged. This means that the new catchword is incomparable to all other catchwords. The `remove catchword` operation takes a single parameter $a \in M$, and simply removes a from M and $((\downarrow a) \times \{a\}) \cup (\{a\} \times (\uparrow a))$ from the ordering relation.

The operation `insert ordering` takes two parameters $a, b \in M$ and inserts into the relation $\leq$, the set $(\downarrow b) \times (\uparrow a)$. The operation has been drawn in the left diagram in Figure 2 which serves as a form of Venn-Diagram for the up-sets and down-sets of a and b before and after the insert operation. The shading gives an indication of corresponding regions.

The insertion of the ordering $b \leq a$ into $\leq$ will require the insertion of the set $\{g \in G \mid (g, b) \in I\} \times (\uparrow a \setminus \uparrow b)$ into I. The portion of M whose image under the relation I will require an update is the upper shaded part in the rightmost diagram in Figure 2.

The operation `remove ordering` takes two parameters $a, b \in M$ where a is an upper cover of b. The `remove ordering` operation removes from $\leq$ the set $((\downarrow b) \setminus (\downarrow(\prec_a \setminus \{b\}))) \times ((\uparrow a) \setminus \uparrow(\succ_b \setminus \{a\}))$. The right diagram in Figure 2 may be used to visualize the remove operation. Similarly to the insert operation, the

removal of the ordering $b \leq a$ from $\leq$ will require a re-computation of the image in I under the elements from $\{a\} \times ((\uparrow a) \setminus \uparrow(\succ_b \setminus \{a\}))$. This region has been shaded in the upper right of Figure 2.

4.2 Modifying the Scale Function

The set $\mathcal{S}$ of scale names, as explained in Section 2, is not necessarily disjoint from M, thus the tree representation of M already presents a view of a portion of $\mathcal{S}$. In order to reduce the complexity of the graphical user interface, we make $\mathcal{S}$ equal to M. That is: all catchwords are scale names, and all scale names are catchwords. Such an assumption is made possible by the definition, given in Section 2, of the default scale for a catchword. A result of this definition is that catchwords with no lower covers will map, under the scale function, α, to the empty set.

The function α maps each catchword m to a set of catchwords. The program displays this set of catchwords, when requested by the user, using a dialog (see Figure 3). The dialog box contains a set of catchwords available for membership in $\alpha(m)$. In Figure 3 this set of candidates has been restricted to the down-set of m. An icon (either a green tick or a red cross) is used to indicate membership (or not) in the set of catchwords given by $\alpha(m)$. By clicking on the icon the user can change the definition of $\alpha(m)$.

By displaying only the down-set of m in the dialog box, the program restricts the definition of α to $\alpha(m) \subseteq (\downarrow m)$. This restriction has an effect on the "remove ordering operation" defined on $(M, \leq)$. When the ordering of $a \leq b$ is removed the image of the function α for attributes in $\uparrow a$ is automatically checked and if necessary modified.

The program has an intended mode of operation for expert users in which the restriction on the definition of $\alpha(m) \subseteq \downarrow m$ is lifted. In this mode the user has all catchwords available for inclusion in $\alpha(m)$, and he may choose the set $\mathcal{S}$ of scale names to be different from the set M of catchwords.

When the function α is changed by the user then the set $\{\mathbb{S}_s \mid s \in \mathcal{S}\}$ of scales is changed automatically. This update occurs regardless of the mode of operation. The new/modified scales can then be used directly for navigating in the concept space as described in Section 4.4.

4.3 Associating Emails with Catchwords

Each member of $(M, \leq)$ is associated with a query term, which in this application is a set of *section/word pairs*. For our purposes a *section* of an email is either a header field, e. g. the "From:" field, or the section "body" which is composed of the parts[4] of the email directly encoding text. More formally stated: Let H be the set of sections found in the email documents, W the set of words found

[4] The MIME extension to the email format allows an email document to have multiple parts. These multiple parts are sometimes referred to as attachments.

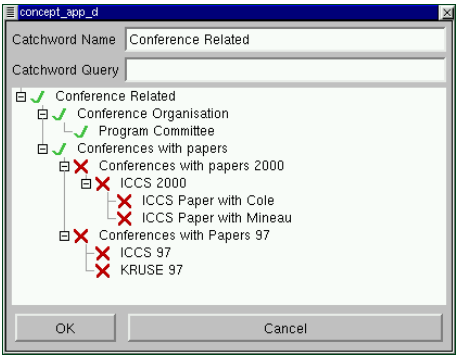


Fig. 3. Dialog for editing α(Emails with Cole)

in the email documents, then the function $\texttt{query}\colon M \to \mathfrak{P}(H \times W)$ attaches to each attribute a set of section/word pairs.

Let G be a set of email documents. Five relations, Q, R, R^+, R^-, and I are defined for managing the different ways in which email documents may be associated with catchwords. $Q \subseteq G \times (H \times W)$ is a relation between documents and section/word pairs. The relation member $(g,(h,w)) \in Q$ indicates that document g has word w in section h. Q is stored via an inverted file index and is only updated when new email is presented to the system. The relation $R \subseteq G \times M$ is derived from the relation Q and the function $\texttt{query}$ via: $(g,m) \in R$ iff $(g,(h,w)) \in Q$ for some $(h,w) \in \texttt{query}(m)$. The relation R is only used as an intermediate step and is calculated from Q as required by the program.

The relations R^+ and R^- store user judgments saying whether or not an email should have a catchword m. These judgments will "over-rule" the relation R. We impose the constraint

$$((\uparrow R^+) \cap (\downarrow R^-)) \stackrel{!}{=} \emptyset \quad (\#)$$

on the two relations R^+ and R^-, saying that a user is not allowed to contradict himself. I. e., he is not allowed, for $m \geq n$, to assign (g,m) to R^- and (g,n) to R^+.

The relation I respecting the compatibility condition ($\ddagger$) is derived from the relations R, R^+ and R^- using the following operator: For any relation $J \subseteq G \times M$, we define $J^\ddagger := \{(g,m) \in G \times M \mid \exists n \in M\colon (g,n) \in J,\ n \leq m\}$. We obtain I as the incidence relation for the formal context (G,M,I) mentioned in Section 2 by $I := ((R \setminus R^-) \cup R^+)^\ddagger$.

These five relations are required to accommodate the different ways in which an email may be associated with catchwords. Q and R associate emails with catchwords via an automatic process based on content and queries attached to catchwords, R^+ and R^- associate email based on user input, and I combines these two sources with the hierarchy defined over the catchwords. By separating the relations for automatic associations of catchwords to emails from the relations

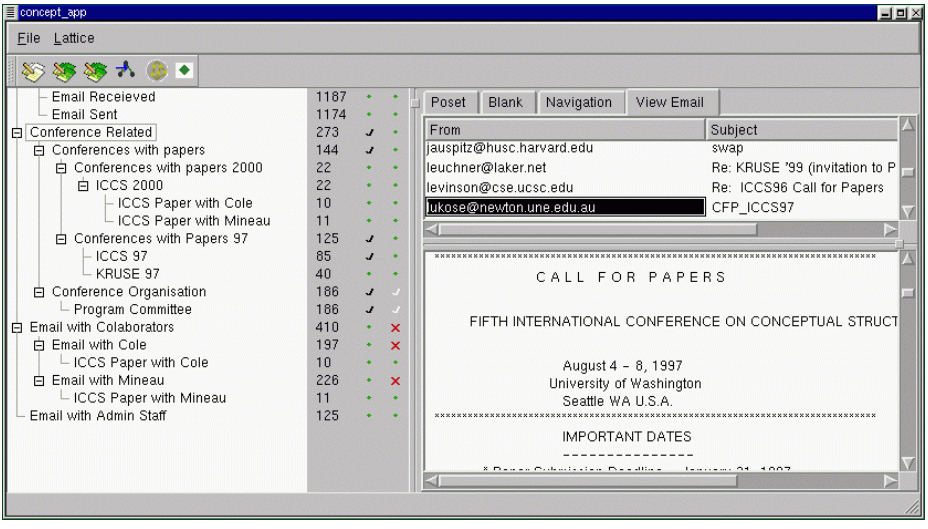


Fig. 4. Interface for viewing email and associating email with catchwords

for user defined associations, the program maintains a pure keyword index into the email collection. Relations R and I are derived from Q, R^+, and R^-, and so need not be stored. Storing I however greatly reduces the time complexity of the program.

When a batch of new emails, G_b, is presented to the program, the relation Q is updated automatically by inserting new pairs, Q_b, into the relation. The modification of Q into $Q \cup Q_b$ will cause an insertion of pairs R_b into R according to $\texttt{query}(m)$ and then subsequently an insertion of new pairs I_b into I. The definitions are:

$$
\begin{aligned}
Q_b &\subseteq G_b \times (H \times W) \\
R_b &= \{(g, m) \mid \exists\, (h, w) \in \text{query}(m) \text{ and } (g, (h, w)) \in Q_b\} \\
I_b &= \{(g, m) \mid \exists\, m_1 \leq m \text{ with } (g, m_1) \in R_b\}
\end{aligned}
$$

The user can modify the association of emails with catchwords in two ways. Firstly by changing the relations R^+ and R^- and secondly by making modifications to the `query` function. In order to explain the user interface for making modifications to R^+ and R^- we introduce the following notation. For an email $g \in G$, we define the restriction of any relation $J \subseteq G \times M$ to this email by $J_g := J \cap (\{g\} \times M)$. For the purpose of brevity of expression we shall say m belongs to J_g if $(g, m) \in J_g$.

The user is able to view individual emails as shown in Figure 4. In this mode icons are attached to catchwords in the tree widget displayed to the left of the email. These icons indicate to the user how each of the catchwords is related to the displayed email by R, R^-, and R^+. The user is able to change the relations R^- and R^+ by interacting with the icons.

1. If m is not in $R_g^{\ddagger}$, $(R_g^+)^{\ddagger}$, or R_g^- then no icon is displayed.
2. If m is in $R_g^{\ddagger}$ then a yellow tick (shown as white in Fig. 4) is displayed.
3. If m is in R_g^- then a red cross is displayed.
4. If m is in $(R_g^+)^{\ddagger}$ then a green (shown as black in Fig. 4) tick is displayed.

All combinations of these icons which do not include at the same time a red cross and a green tick are possible.

The user can then determine that the displayed email has a catchword in I if there is either a green tick or a yellow tick in the absence of a red cross. The program provides two basic operations, `associate attribute` and `disassociate attribute` from which more complex operations for use in the user interface may be constructed. The `associate attribute` operation takes two parameters, an email document, and a catchword m. The operation inserts the pair (g, m) into R^+, and removes, for all $n \geq m$, (g, s) from R^-. Similarly the operation `disassociate attribute` takes two parameters, an email and a catchword. The operation inserts (g, m) in R^- and removes, for all $n \leq m$, (g, n) from R^+. The construction of the two operators guarantees that the constraint (#) is always satisfied.

The user is also able to influence the way that R is derived from Q by modifying the `query` function. The user is able to modify $\alpha(m)$ using the Scale Query field in the dialog box shown in Figure 3. After any such modification to the `query` function the relations R and I are modified accordingly.

New emails presented to the system for automated indexing cause a modification to the inverted file index consisting only of new entries. The insertion of new email documents into an inverted file index is an efficient operation. The complexity of inserting each document is $O(1)$. When the user makes a modification to either R^+ or R^- of a removal or insertion of (g, m) this will cause all catchwords in the order filter of m, or order ideal, resp., to be updated in I. The expense of such an update depends on how I is stored but is likely to be $O(log(n))$ where n is the average number of documents per attribute.

It is useful for the system to maintain the relation R^+ for special catchwords dependent on observation by the program of the users behavior. Two examples of such catchwords are "read emails" for emails that the user has displayed at some time, and "unseen emails" for emails that the user has not yet been notified of.

4.4 Navigating the Conceptual Email Space

To assist the user in navigating the conceptual space of emails, the program draws simple line diagrams and (locally) nested line diagrams. A simple line diagram is used to visualize a single scale, while nested line diagrams are used to visualize combinations of scales. The concept lattices, from which the nested line diagrams are drawn, are computed from the contexts given by $\mathbb{S}_{\alpha(s)}$. The contexts are calculated using the algorithm reported in [1], and the concept lattices are calculated from these contexts via Ganter's algorithm [3].

The user may navigate the conceptual space of emails documents for different purposes:

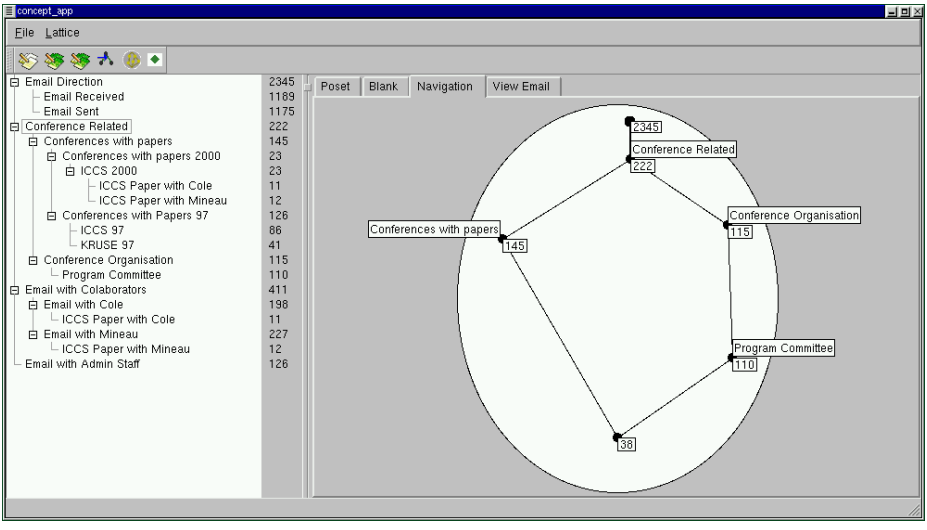


Fig. 5. Concept Lattice derived from the Scale for "Conference Related".

1. to find collections of emails thematically linked;
2. to review the precision and recall of queries attached to catchwords by comparing them with catchwords based on user judgments (for the purpose of refining them for improving the `query` function); and
3. to review patterns of communication between different groups.

Of these purposes the first is the most useful to common users of the program. A simple scenario in which the user has this first purpose is presented here. Imagine a researcher who was in the Program Committee (PC) of ICCS '97 and was at that time co-authoring with other members of the PC for the same conference. For the organization of a conference in the year 2000, she wants to retrieve some facts about the organization of ICCS '97. But she only remembers that she exchanged this information with one of the people she was co-authoring with for ICCS '97, and that it was only one tiny part of a mail covering all kinds of topics.

The researcher may begin her search by requesting a line diagram for the scale named "Conference Related". This scale is shown in Figure 5. It shows that from her 2344 emails in total, there are 222 emails related to conferences, 145 of which are related to conferences with papers submitted and 110 of which are related to both conference organisation and program committees. The researcher decides that the email she is looking for is likely to be under the catchword "Conferences with Papers". As there are too many emails in its extent to be read through, she may for instance want to expand the concept. By choosing the scale $\mathbb{S}_{\text{Conferences 1997}}$, she obtains Figure 6.

Now the researcher can for instance check the 19 mails related to "ICCS '97" and "Conference Organization/Program Committee". If she still doesn't find

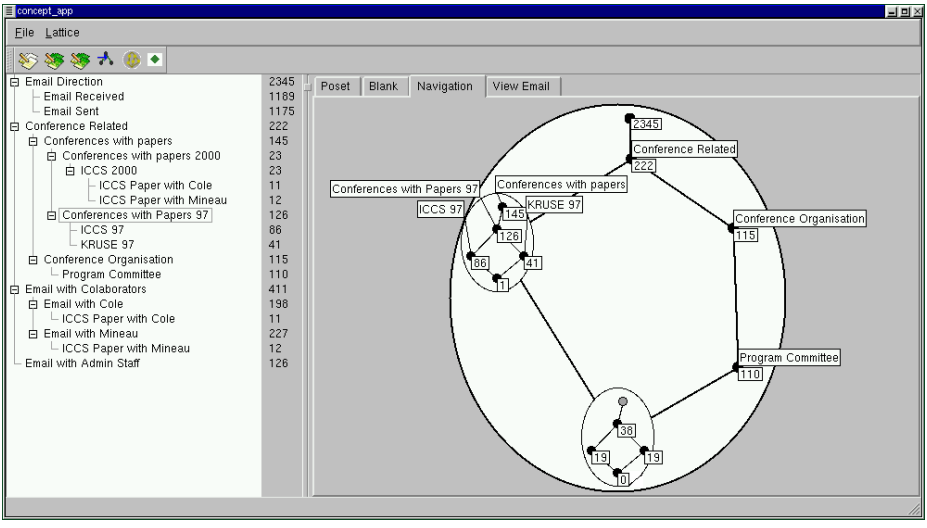


Fig. 6. Concept Lattice derived from the Scales for "Conference Related" and "Conferences with Papers".

the email she is looking for there, then she has to check either the 86 papers related to "ICCS '97" or even all 115 emails under the catchword "Conference Organization". Before doing this, however, she might want to differentiate these concepts further, e. g. by zooming into them with the scale "Members of ICCS '97 Program Committee". If this scale doesn't exist yet, then she can create it on the fly using the widget for modifying the scale function (and eventually store it for further use).

Note that with a classical, tree-structured search hierarchy (where one usually has the names of the correspondents on the highest level), one would be forced to scan all branches starting with the names of the co-authors before one can tell the system constraints like "Conference Related".

5 Outlook

Having completed a prototype implementation of CEM (available on request from the first author, the next step is to evaluate its operation in daily use and to measure is scalability with respect to large data sets and distributed collections of email. We also consider allowing the user to impose more structure on M including conjunctive implications and negation, following the mathematical foundation presented in [4], as well as a more expressive language for the `query` function which allows for instance disjunctive queries.

Although CEM has been in this paper applied to email documents it has a more general use as a document management system. The next step therefore is to extend the current architecture to allow the user to associate catchwords with

files accessible remotely via the internet and also locally with the users private collection. This next step has the challenge of dealing with a large number of protocols and file formats. The emergence of standards such as XML and RDF gives some hope for a general and unified method for processing of this myriad of data formats. Looking further ahead one can consider how a conceptual file management system might be used in a group environment or at an enterprise level where several users contribute to the structure of the hierarchy and the association of catchwords with files.

References

1. R. Cole, P. Eklund: Scalability in formal concept analysis: a case study using medical texts. *Computational Intelligence* Vol. 15, Number 1, 1999, 11–27
2. A. Fall: Dynamic taxonomical encoding using sparce terms. *Proc. ICCS '96*. LNAI **1115**, Springer, Heidelberg 1996, 278–292.
3. B. Ganter, R. Wille: *Formal Concept Analysis: mathematical foundations*. Springer, Heidelberg 1999 (Translation of: Formale Begriffsanalyse: Mathematische Grundlagen. Springer, Heidelberg 1996)
4. B. Ganter, R. Wille: Contextual Attribute Logic *Proc. ICCS '99*. LNAI **1115**, Springer, Heidelberg 1999, 377–388.
5. J. Hereth: *Formale Begriffsanalyse im Data Warehousing*. Diploma thesis, TU Darmstadt 2000
6. K. Jones: View Mail Users Manual. *http://www.wonderworks.com/vm*. 1999
7. T. Rock, R. Wille: Ein TOSCANA-System zur Literatursuche. In: G. Stumme and R. Wille (eds.): *Begriffliche Wissensverarbeitung: Methoden und Anwendungen*. Springer, Berlin-Heidelberg 2000, 239–253
8. W. Schuller: *http://gmail.linuxpower.org/*. 1999
9. G. Stumme: Local Scaling in Conceptual Data Systems. *Proc. ICCS '96*. LNAI **1115**, Springer, Heidelberg 1996, 308–320.
10. G. Stumme: Hierarchies of Conceptual Scales. *Proc. Workshop on Knowledge Acquisition, Modeling and Management*. Banff, 16.–22. Oktober 1999, Vol. 2, 5.5.1–18
11. F. Vogt, R. Wille: TOSCANA — A graphical tool for analyzing and exploring data. In: R. Tamassia, I. G. Tollis (eds.): *Graph Drawing '94*, LNCS **894**, Springer, Heidelberg 1995, 226–233
12. R. Wille: Restructuring lattice theory: an approach based on hierarchies of concepts. In: I. Rival (ed.): *Ordered sets*. Reidel, Dordrecht–Boston 1982, 445–470

A Contextual-Logic Extension of TOSCANA

Peter Eklund[1], Bernd Groh[1], Gerd Stumme[2], and Rudolf Wille[2]

[1] School of Information Technology, Griffith University, Gold Coast Campus, PMB 50
Gold Coast Mail Centre QLD 9726, Australia; {p.eklund, b.groh}@gu.edu.au
[2] Fachbereich Mathematik , Technische Universität Darmstadt,
Schloßgartenstr. 7, D–64289 Darmstadt;
{stumme,wille}@mathematik.tu-darmstadt.de

Abstract. The aim of this paper is to indicate how TOSCANA may be extended to allow graphical representations not only of concept lattices but also of concept graphs in the sense of Contextual Logic. The contextual-logic extension of TOSCANA requires the logical scaling of conceptual and relational scales for which we propose the Peircean Algebraic Logic as reconstructed by R. W. Burch. As graphical representations we recommend, besides labelled line diagrams of concept lattices and Sowa's diagrams of conceptual graphs, particular information maps for utilizing background knowledge as much as possible. Our considerations are illustrated by a small information system about the domestic flights in Austria.

Contents

1. TOSCANA
2. Set-theoretical Semantics of Contextual Logic
3. Conceptual and Relational Scales
4. Graphical Representations

1 TOSCANA

TOSCANA has been developed at Darmstadt University of Technology as a computer program for analyzing and exploring data by methods of *Formal Concept Analysis* (see [KSVW94], [VW95], [GW99]). It has been used in a wide range of application domains such as medicine, psychology, social sciences, linguistics, information sciences, machine and civil engineering etc. (cf. [GSW98], [SW00]). A typical application combines TOSCANA and a (relational) database to a TOSCANA-*system* allowing the representation, the maintainence, and the activation of information so that users of the system may gain actual knowledge about interesting aspects of the relevant application domain. For the human-machine interaction, TOSCANA offers labelled (nested) line diagrams of concept lattices, representing conceptual relationships of the stored data, and allows the navigation through the data by changing from one line diagram to another (and so on). Although the graphical representations of concept lattices have been proven useful in numerous applications, there are many cases in which such

B. Ganter and G.W. Mineau (Eds.): ICCS 2000, LNAI 1867, pp. 453–467, 2000.
© Springer-Verlag Berlin Heidelberg 2000

representations of conceptual hierarchies are not sufficient and, in addition, representations of (non-hierarchical) relations become desirable. Those cases have stimulated the extension of Formal Concept Analysis to *Contextual Logic*, a formal logic semantically based on formal contexts from which concept lattices are derivable and also concept graphs, combining formal concepts and conceptual relations (see [Wi97], [Pr98], [PW99]). The aim of this paper is to indicate how TOSCANA may be extended to allow graphical representations not only of concept lattices but also of concept graphs in the sense of Contextual Logic.

Before discussing a contextual-logic extension of TOSCANA, we explain through an example how TOSCANA can be used to establish a conceptual information system based on methods of Formal Concept Analysis. The data for the example, which have already been considered in [PW99], are described in Figure 1 by a data table about the domestic flights in Austria. For a TOSCANA-system, such table is usually considered as a *many-valued context* (G, M, W, I) for which G is the object set comprising all listed flights, M is the attribute set consisting of the attributes "Airline", "Departure Airport", "Departure Time", "Arrival Airport", "Arrival Time", "Days", and "Aircraft", W is a set containing all attribute values described by the entries in the columns of the table, and I is the ternary relation between G, M, and W indicating which object has which value for which attribute.

The first step of establishing a TOSCANA-system for the many-valued context represented in Figure 1 is to turn it into a number of formal contexts, called *conceptual scales*, grasping the information coded by the data. Such transformation should be guided by purposes which we assume in our example to lie in the support of flight information. For our explanations, we choose the seven conceptual scales "Connections", "Departure Time (Hours)", "Departure Time (Minutes)", "Arrival Time (Hours)", "Arrival Time (Minutes)", "Days", and "Airline/Aircraft". The concept lattice of the conceptual scale "Connections" is presented in Figure 2; it yields the information about the departure and arrival airports of the 75 domestic flights which are denoted by their flight number; for instance, the flight 1583 departs from Innsbruck and arrives at Graz (surprisingly, there is no flight from Graz to Innsbruck). The information about the departure time of the flights is well represented by the two interordinal scales "Departure Time (Hours)" and "Departure Time (Minutes)" where the first has the attributes $\leq 6, \ldots, \leq 23$ and $\geq 6, \ldots, \geq 23$ and the second has the attributes $\leq 00, \leq 05, \ldots, \leq 50, \leq 55$ and $\geq 00, \geq 05, \ldots, \geq 50, \geq 55$ (cf. [PW99], Fig. 3); for instance, the departure time 15.10 of flight 1583 is uniquely determined by the attributes ≤ 15 and ≥ 15 of the first scale and ≤ 10 and ≥ 10 by the second scale. The arrival time is represented analogously by two interordinal scales. The choice of interordinal scales for representing time has been well proved because they allow the expression of time intervals by formal concepts. The conceptual scale "Airline/Aircraft" has the attributes "VO", "OS", "F70", "DH8", and "CRJ" yielding a six-element concept lattice. The concept lattice of the conceptual scale "Days", restricted to the flights from Vienna to Innsbruck, is presented in Figure 3.

Flight	Airline	Departure		Arrival		Days	Aircraft
		Airport	Time	Airport	Time		
070	VO	Vienna	07.50	Innsbruck	08.40	1-6	F70
071	VO	Innsbruck	06.25	Vienna	07.20	1-5	F70
072a	VO	Vienna	10.20	Innsbruck	11.35	6	DH8
072b	VO	Vienna	10.50	Innsbruck	12.05	1-5, 7	DH8
073a	VO	Innsbruck	08.35	Vienna	09.45	67	DH8
073b	VO	Innsbruck	09.05	Vienna	09.55	1-5	F70
074	VO	Vienna	13.55	Innsbruck	15.10	2-5, 7	DH8
075	VO	Innsbruck	11.40	Vienna	12.50	1-5	DH8
076a	VO	Vienna	17.45	Innsbruck	18.40	1-6	F70
076b	VO	Vienna	18.40	Innsbruck	19.55	7	DH8
077	VO	Innsbruck	15.35	Vienna	16.45	2-5	DH8
078a	VO	Vienna	20.35	Innsbruck	21.25	1-4	F70
078b	VO	Vienna	21.30	Innsbruck	22.45	7	DH8
078c	VO	Vienna	21.40	Innsbruck	22.35	5	CRJ
330	VO	Linz	06.20	Salzburg	06.50	1-6	CRJ
331	VO	Salzburg	11.20	Linz	11.45	1-5	CRJ
332	VO	Linz	16.05	Salzburg	16.35	1-5	CRJ
333	VO	Salzburg	21.50	Linz	22.15	1-5, 7	CRJ
409	VO	Graz	12.10	Linz	12.45	1-5	CRJ
410	VO	Linz	16.10	Graz	16.50	1-5	CRJ
412	VO	Linz	10.35	Graz	11.10	1-5	CRJ
413	VO	Graz	06.15	Salzburg	06.50	1-5	CRJ
415	VO	Graz	17.30	Salzburg	18.10	1-5	CRJ
416	VO	Salzburg	21.50	Graz	22.25	1-5, 7	CRJ
417	VO	Graz	17.15	Linz	17.45	7	CRJ
501	VO	Klagenfurt	06.00	Salzburg	06.45	1-5	DH8
502	VO	Salzburg	21.55	Klagenfurt	22.40	1-5, 7	DH8
531*	VO-OS	Linz	06.00	Vienna	06.45	1-6	DH8
532*	VO-OS	Vienna	10.40	Linz	11.20	1-5, 7	DH8
533*	VO-OS	Linz	08.35	Vienna	09.25	1-7	DH8
534*	VO-OS	Vienna	22.15	Linz	23.00	1-5, 7	DH8
536a*	VO-OS	Vienna	17.10	Linz	17.55	5	DH8
536b*	VO-OS	Vienna	17.15	Linz	17.55	1-4, 7	DH8
537*	VO-OS	Linz	12.00	Vienna	12.50	1-5, 7	DH8
538*	VO-OS	Vienna	20.30	Linz	21.15	1-7	DH8
539*	VO-OS	Linz	18.15	Vienna	19.00	1-5, 7	DH8
540*	VO-OS	Vienna	10.45	Graz	11.30	1-7	DH8
541*	VO-OS	Graz	06.05	Vienna	06.45	1-6	DH8
542*	VO-OS	Vienna	13.50	Graz	14.35	1-5	DH8
543*	VO-OS	Graz	08.50	Vienna	09.35	1-7	DH8
544*	VO-OS	Vienna	17.20	Graz	18.00	1-7	DH8
545*	VO-OS	Graz	11.55	Vienna	12.35	1-5	DH8
546*	VO-OS	Vienna	19.40	Graz	20.20	1-7	DH8
547*	VO-OS	Graz	15.30	Vienna	16.15	1-5, 7	DH8
548*	VO-OS	Vienna	22.30	Graz	23.10	1-5, 7	DH8
549*	VO-OS	Graz	18.25	Vienna	19.05	1-5, 7	DH8
550*	VO-OS	Vienna	07.25	Klagenfurt	08.15	1-5	DH8
551*	VO-OS	Klagenfurt	06.00	Vienna	06.50	1-6	DH8
552*	VO-OS	Vienna	10.40	Klagenfurt	11.30	1-7	DH8
553*	VO-OS	Klagenfurt	08.40	Vienna	09.30	1-7	DH8
554*	VO-OS	Vienna	13.55	Klagenfurt	14.50	1-5	DH8
555*	VO-OS	Klagenfurt	11.55	Vienna	12.45	1-7	DH8
556*	VO-OS	Vienna	17.10	Klagenfurt	18.00	1-7	DH8
557*	VO-OS	Klagenfurt	15.15	Vienna	16.10	1-5	DH8
558*	VO-OS	Vienna	19.50	Klagenfurt	20.45	1-7	DH8
559*	VO-OS	Klagenfurt	18.20	Vienna	19.10	1-7	DH8
560*	VO-OS	Vienna	22.30	Klagenfurt	23.20	457	DH8
561*	VO-OS	Klagenfurt	21.00	Vienna	22.00	457	DH8
590*	VO-OS	Vienna	10.25	Salzburg	11.20	1-7	DH8
591*	VO-OS	Salzburg	17.15	Vienna	18.10	7	DH8
593*	VO-OS	Salzburg	08.15	Vienna	09.15	1-7	DH8
594*	VO-OS	Vienna	17.35	Salzburg	18.35	1-7	DH8
595*	VO-OS	Salzburg	11.45	Vienna	12.40	1-7	DH8
596a*	VO-OS	Vienna	20.25	Salzburg	21.20	6	DH8
596b*	VO-OS	Vienna	20.35	Salzburg	21.30	1-5, 7	DH8
597*	VO-OS	Salzburg	19.05	Vienna	20.00	1-7	DH8
1557	VO	Klagenfurt	16.00	Vienna	16.50	7	DH8
1583	VO	Innsbruck	15.10	Graz	15.55	7	CRJ
1596	VO	Vienna	14.05	Salzburg	15.05	5	DH8
2980	VO	Innsbruck	06.10	Salzburg	06.40	1-7	DH8
2981	VO	Salzburg	12.30	Innsbruck	13.00	1-7	DH8
2983	VO	Salzburg	16.40	Linz	17.05	1-5	CRJ
2984	VO	Innsbruck	14.35	Salzburg	15.10	1-7	DH8
2985	VO	Salzburg	21.40	Innsbruck	22.05	1-7	DH8
2986	VO	Innsbruck	10.20	Salzburg	10.55	7	DH8

Fig. 1. A data table about the *domestic flights in Austria*

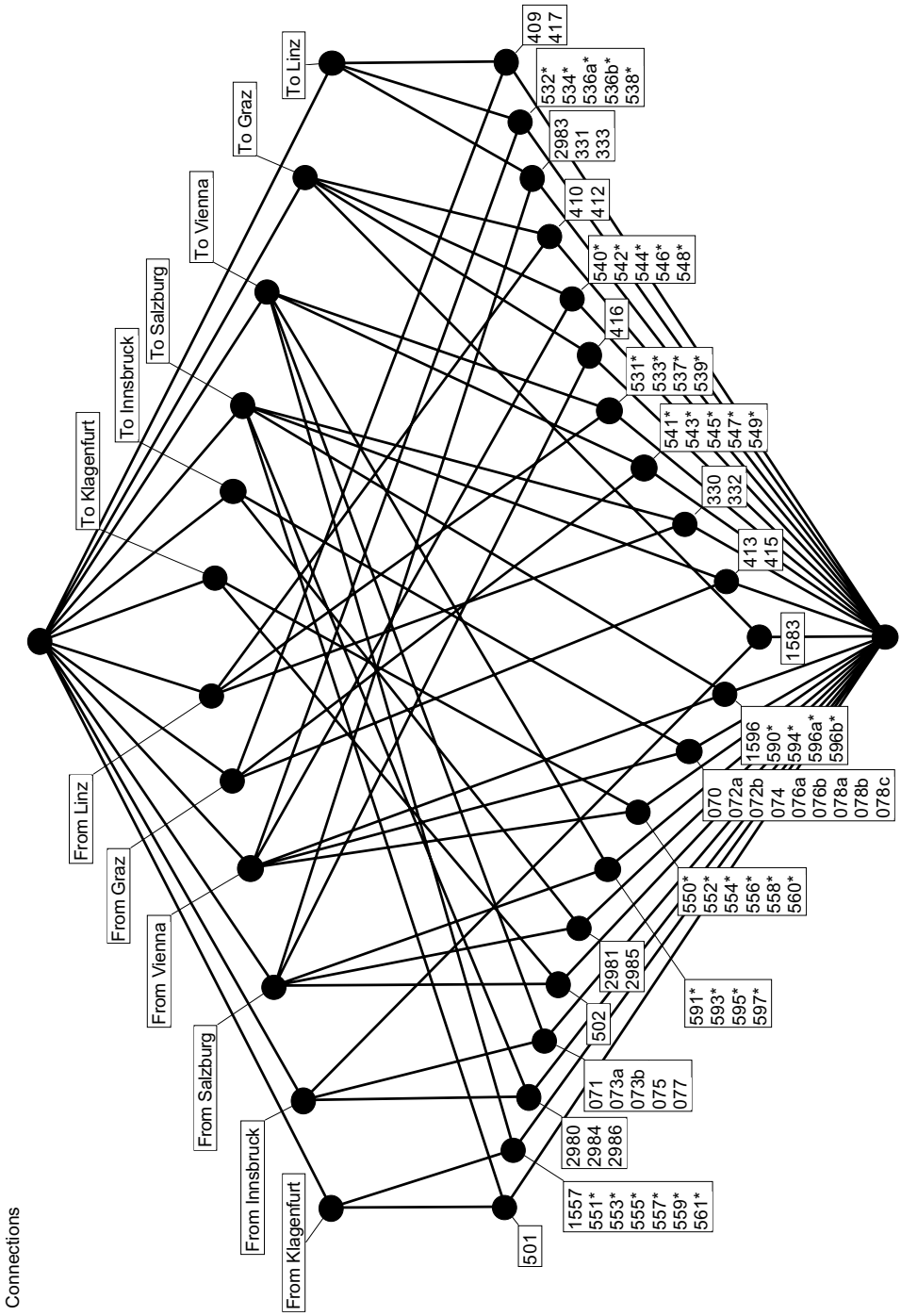


Fig. 2. The concept lattice of the conceptual scale "Connections"

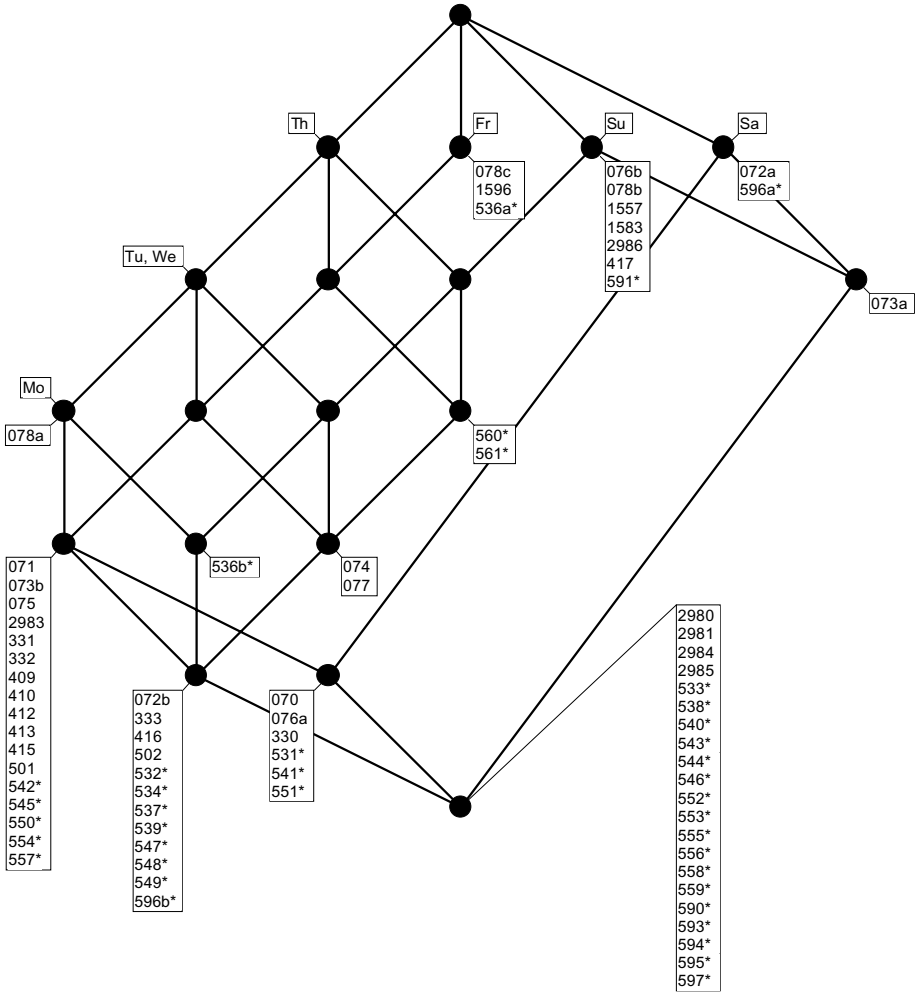


Fig. 3. The concept lattice of the conceptual scale "Days", restricted to the flights from Vienna to Innsbruck

The conceptual scales of a TOSCANA-system, each given by a formal context together with a line diagram of its concept lattice, are viewed on three levels of abstraction to allow high flexibility of their use: First, a conceptual scale is considered as an *abstract scale* having only (clarified) abstract objects and attributes without a particular meaning. Secondly, a conceptual scale is considered as a *concrete scale* having still abstract objects but meaningful attributes with respect to an application domain. Thirdly, a conceptual scale is considered as a *realized scale* having now both: meaningful objects and attributes with respect to an application domain. Abstract scales may be used in different applications,

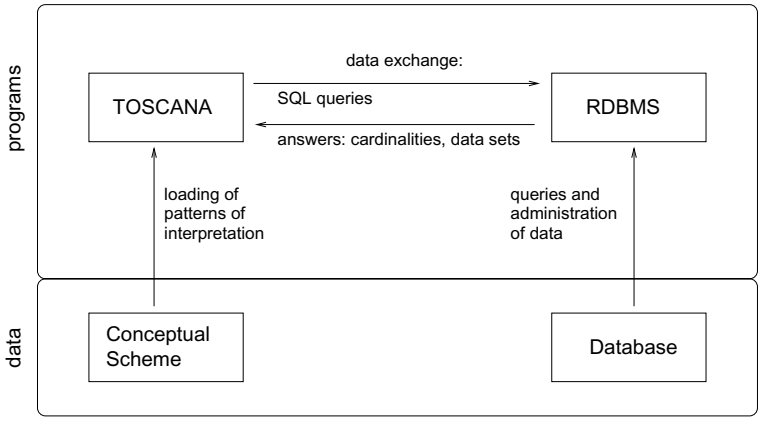


Fig. 4. The basic architecture of a TOSCANA-system

while concrete scales are designed for a specific application, but allow different realizations concerning the set of objects (for instance, concerning actual revisions of the flights in the database on domestic flights in Austria).

The basic architecture of a TOSCANA-system is described in Figure 4. According to that, a TOSCANA-system divides into a data and a program part. In the data part, the information about the objects of the application domain is coded in the (relational) database, while the domain-specific concrete scales (including line diagrams of their concept lattices) and SQL-queries for the objects of the application domain are collected in the *conceptual scheme*; in our example this means that the data table of Figure 1 is stored in the database and that the relevent seven conceptual scales, understood as concrete scales, are available in the conceptual scheme of the data part. For the represention of concrete scales with line diagrams of their concept lattices in the conceptual scheme, the drawing program ANACONDA yields a coding by the description language CONSCRIPT (see [Vo96]). The drawings of the concept lattices has to be provided in advance since, in general, well readable line diagrams cannot be generated automatically.

The program part consists of TOSCANA and a relational database management system (RDBMS). At runtime, TOSCANA loads the conceptual scheme and connects it to a RDBMS which accesses the database in which the information of the application domain is stored. The system offers the user the list of conceptual scales, among which the user may choose one or more scales. If the user chooses one scale, then TOSCANA computes the realized scale and the labelling of the line diagram of its concept lattice by querying the actual objects in the database, and displays the labelled line diagram on the screen. Figure 2, showing the concept lattice of the realized scale "Connections", is the result of such an action. If the user chooses two (or more) scales, then she can either zoom into a concept of the first scale to obtain a conceptual refinement of this concept by the concept lattice of the second scale, or she can represent the concept lattices of both scales simultaneously by a nested line diagram. Figure 3 results, after choosing the conceptual scales "Connections" and "Days", from zooming

in Figure 2 into the ninth object node from the left. A detailed description of TOSCANA-systems can be found in [Na99].

The explanations concerning the TOSCANA-system of the domestic flights in Austria might convince that such an information system could be useful; for instance, the object nodes of Figure 2 partition the flights according to their departure and arrival airports and lead, after drilling down to a suitable object node, further information about departure and arrival time, week days, airline, and even aircraft, relevant to flights from specifically chosen departure and arrival airports. Nevertheless, the representation of all relationships within conceptual hierarchies does not seem sufficient for human-machine communications. Therefore, an extension of TOSCANA should be developed which allows more direct representations of non-hierarchical relations. Such an extension, mathematically based on Contextual Logic, is described in the following sections.

2 Set-Theoretical Semantics of Contextual Logic

Contextual Logic has been developed as a mathematization of the traditional philosophical logic with its doctrines of concepts, judgments, and conclusions [Wi00]; for that, the mathematical theory of concepts is taken from Formal Concept Analysis, and the mathematical theory of judgments is derived from Sowa's Theory of Conceptual Graphs [So84]. The set-theoretical semantics of Contextual Logic is based on power context families which are defined as follows: a *power context family* is a sequence $\mathbb{K} := (\mathbb{K}_0, \mathbb{K}_1, \ldots, \mathbb{K}_n)$ $(n \geq 2)$ of formal contexts $\mathbb{K}_k := (G_k, M_k, I_k)$ with $G_k \subseteq (G_0)^k$ for $k = 1, \ldots, n$. The formal concepts of $\mathbb{K}_k$ $(1 \leq k \leq n)$ represent by their extents k-ary relations on the object set G_0; they are therefore called the k-ary *relational concepts* of $\mathbb{K}$.

According to [PW99], the *formal judgments*, also called *concept graphs*, of a power context family can be represented (up to logical equivalence) as the elements of all the direct products

$$\prod_{(k,g)\in U} [\gamma_k g)$$

for which U is a finite subset of $\bigcup_{k=0,\ldots,n} \{k\} \times G_k$ satisfying the implication

$$(k, g) \in U \text{ and } g = (g_1, \ldots, g_k) \Longrightarrow (0, g_1), \ldots, (0, g_k) \in U \quad (1 \leq k \leq n),$$

and $[\gamma_k g)$ is the principal filter of $\mathfrak{B}(\mathbb{K}_k)$ generated by the smallest concept having g in its extent. An element $\mathfrak{a} := (\mathfrak{a}_{(k,g)} \mid (k, g) \in U)$ of the product can be understood as the family of the *atomic concept graphs* $[\mathfrak{a}_k : g]$ with $(k, g) \in U$ where the so-called *conceptual instance* $[\mathfrak{a}_0 : g]$ is the element $\mathfrak{a}$ of the (one-factor) product with $U := \{(0, g)\}$ and $\mathfrak{a}_{(0,g)} = \mathfrak{a}_0$ and, for $k = 1, \ldots, n$, the so-called *relational instance* $[\mathfrak{a}_k : g]$ is the element $\mathfrak{a}$ of the product with $U := \{(k, g), (0, g_1), \ldots, (0, g_k)\}$ (if $g = (g_1, \ldots, g_k)$) and $\mathfrak{a}_{(k,g)} = \mathfrak{a}_k$, $\mathfrak{a}_{(0,g_1)} = \mathfrak{u}_0, \ldots, \mathfrak{a}_{(0,g_k)} = \mathfrak{u}_0$ (in general, $\mathfrak{u}_k$ denotes the largest concept of $\mathbb{K}_k$); instead of $[\mathfrak{a}_k : g]$ we also write $(\mathfrak{a}_k; [\mathfrak{u}_0 : g_1], \ldots, [\mathfrak{u}_0 : g_k])$. Notice that a pair $(\mathfrak{a}_k, g)$

with $\mathfrak{a}_k \in \mathfrak{B}(\mathbb{K}_k)$ and $g \in G_k$ forms an atomic concept graph if and only if $g \in Ext(\mathfrak{a})$.

The formal judgments described in the preceding paragraph may be called *concept graphs in atomistic normal form*. From a concept graph in atomistic normal form one can deduce its equivalent concept graphs using the following equivalences (cf. [PW99], Prop. 1):

(1) $([\mathfrak{a}_t : g] \mid t \in T)$ is equivalent to $[\bigwedge_{t\in T} \mathfrak{a}_t : g]$,
(2) $([\mathfrak{a} : g_t] \mid t \in T)$ is equivalent to $[\mathfrak{a} : \{g_t \mid t \in T\}]$,
(3) $(\mathfrak{a}; [\mathfrak{b}_1 : B_1], \ldots, [\mathfrak{b}_k : B_k])$
is equivalent to $([\mathfrak{a}_k : B_1 \times \cdots \times B_k], [\mathfrak{b}_1 : B_1], \ldots, [\mathfrak{b}_k : B_k])$,
(4) $((\mathfrak{a}; [\mathfrak{b}_1 : B_1], \ldots, [\mathfrak{b}_k : B_k]), (\tilde{\mathfrak{a}}; [\tilde{\mathfrak{b}}_1 : \tilde{B}_1], \ldots, \ldots, [\tilde{\mathfrak{b}}_l : \tilde{B}_l]))$
is equivalent to
$((\mathfrak{a}; [\mathfrak{b}_1 : B_1], \ldots, [\mathfrak{b}_k : B_k]), (\tilde{\mathfrak{a}}; [\tilde{\mathfrak{b}}_1 : \tilde{B}_1], \ldots, [\tilde{\mathfrak{b}}_l : \tilde{B}_l]))/\theta(i, j)$
if $[\mathfrak{b}_i : B_i] = [\tilde{\mathfrak{b}}_j : \tilde{B}_j]$.

The equivalences (1), (2), and (3) are applied to suitable subfamilies of a family of elementary concept graphs to obtain another family of elementary concept graphs (in [PW99], an *elementary concept graph* is understood as a concept graph with at most one relational concept). The equivalence (4) allows us to identify equal conceptual instances related to different relational concepts, i. e., (4) is used to split up a concept graph in elementary concept graphs and to combine concept graphs to one concept graph.

Formal judgments (i.e. concept graphs), which may be easily turned into textual statements of "plain English" (see [ME99]), are useful for expressing specific information contained in a given power context family; they are the means "to make the data talk". Therefore, it is desirable to derive from the data in a relational database a power context family allowing one to deduce informative concept graphs. For doing this, the method of *relational scaling* has been proposed in [PW99]: If a relational database is given in form of Codd's relational model $R \subseteq \prod_{t\in T} A_t$, then a derived power context family $\mathbb{K} := (\mathbb{K}_0, \mathbb{K}_1, \ldots, \mathbb{K}_n)$ of formal contexts $\mathbb{K}_k := (G_k, M_k, I_k)$ $(0 \leq k \leq n)$ is formed by suitable subsets $G_0 \subseteq \bigcup_{t\in T} A_t$ and $G_k \subseteq (G_0)^k$ together with corresponding attribute sets giving purpose-oriented meaning to the data (the elements of G_k should be restrictions of elements of $\prod_{t\in T} A_t$).

Understanding the data table in Figure 1 as a representation of a relational model in the sense of Codd, we may derive a power context family $(\mathbb{K}_0, \mathbb{K}_2)$ by the following definitions of object and attribute sets (cf. [PW99]):

$$
\begin{aligned}
G_0 := {} & \{070, 071, \ldots, 2986\} \cup \\
& \{Graz, Innsbruck, Klagenfurt, Linz, Salzburg, Vienna\} \cup \\
& \{06.00, 06.05, \ldots, 23.20\} \cup \\
& \{1, 2, 3, 4, 5, 6, 7\} \\
G_2 := {} & \{(070, Vienna), (071, Innsbruck), \ldots, (2986, Innsbruck)\} \cup \\
& \{(070, Innsbruck), (071, Vienna), \ldots, (2986, Salzburg)\} \cup \\
& \{(070, 07.50), (071, 06.25), \ldots, (2986, 10.20)\} \cup \\
& \{(070, 08.40), (071, 07.20), \ldots, (2986, 10.55)\} \cup \\
& \{(070, 1), \ldots, (070, 6), (071, 1), \ldots (071, 5), \ldots, (2986, 7)\} \\
M_0 := {} & \{flight, airline, airport, time, days, aircraft\} \cup \\
& \{Graz, Innsbruck, Klagenfurt, Linz, Salzburg, Vienna\} \cup \\
& \{\geq 20min, \geq 25min, \geq 30min, \geq 3km, \geq 4km, \geq 5km, \geq 12km, \geq 16km\} \cup \\
& \{\leq 06.00, \leq 06.05, \ldots, \leq 23.20, a.m., p.m.\} \cup \\
& \{\geq 06.00, \geq 06.05, \ldots, \geq 23.20, a.m., p.m.\} \cup \\
& \{Mo, Tu, We, Th, Fr, Sa, Su\} \cup \\
& \{VO, OS, F70, DH8, CRJ\} \\
M_2 := {} & \{From, To, Dept, Arrv, FlDays\}
\end{aligned}
$$

The attributes "$\geq 20min$", "$\geq 25min$","$\geq 30min$" indicate the connecting times at the corresponding airport and the attributes "$\geq 3km$","$\geq 4km$", "$\geq 5km$","$\geq 12km$","$\geq 16km$" indicate the distance from the airport to the corresponding city; furthermore, the attributes "$From$" resp. "To" apply to the pairs consisting of a flight number and a departure resp. arrival airport; analogously, the attributes "*Dept*" resp. "*Arrv*" are understood with respect to the departure resp. arrival times. The incidence relations I_0 and I_2 of the formal contexts $\mathbb{K}_0$ and $\mathbb{K}_2$ reflect the declared or obvious meaning of the attributes with respect to the given objects.

3 Conceptual and Relational Scales

In this section we discuss how a contextual-logic extension of a TOSCANA-system may be developed on the basis of a power context family $\mathbb{K} := (\mathbb{K}_0, \mathbb{K}_1, \ldots, \mathbb{K}_n)$ of formal contexts $\mathbb{K}_k := (G_k, M_k, I_k)$ $(0 \leq k \leq n)$. It seems most natural that the extended TOSCANA program is designed for the purpose of combining several TOSCANA-systems, one for each formal context $\mathbb{K}_k$; in this setting, the scales for $\mathbb{K}_0$ are called the *conceptual scales*, while the scales for $\mathbb{K}_k$ are called the *k-relational scales* $(k = 1, \ldots, n)$. Of course, the TOSCANA-systems of the formal contexts of a power context family should not be independent because satisfactory information based on the data coded in the power context family has also to combine information out of different formal contexts of the family. This combination can be performed by the method of *logical scaling*, introduced in Formal Concept Analysis by S. Prediger (see [Pr97]). In the case of the power context family $\mathbb{K}$, logical scaling is combining attributes out of $\bigcup_{j=0}^{n} M_j$ to obtain new attributes for the formal contexts $\mathbb{K}_k$; as a necessary construction tool, we propose the *term formation* of the *"Peircean Algebraic*

Logic" (PAL) which R. W. Burch created as "an attempt to amalgamate various systems of logic that Peirce developed over his long career" (see [Bu91]).

According to PAL, the basic operations to combine attributes of $\bigcup_{j=0}^{n} M_j$ are the *negation* $\neg$, the *permutations* $\pi \in S_j$, and two so-called *joins*, ι_1 and ι_2, extensionally defined for k-ary attributes m and l-ary attributes $\tilde{m}$ $(k \leq l)$ by

$$
\begin{aligned}
&(1) \quad (\neg m)^{I_k} := G_k \setminus m^{I_k},\\
&(2) \quad (\pi m)^{I_k} := \{(g_{\pi(1)}, \ldots, g_{\pi(k)}) \mid (g_1, \ldots, g_k) \in m^{I_k}\},\\
&(3)\ (\iota_1 m)^{I_{k-2}} := \{(g_1, \ldots, g_{k-2}) \mid \exists g_{k-1} \exists g_k : (g_1, \ldots, g_k) \in m^{I_k} \text{ and } g_{k-1} = g_k\},\\
&(4)\ (\iota_2(m, \tilde{m}))^{I_{k+l-2}} := \{(g_1, \ldots, g_{k-1}, \tilde{g}_2, \ldots, \tilde{g}_l) \mid \exists g_k \exists \tilde{g}_1 : (g_1, \ldots, g_k) \in m^{I_k}\\
&\qquad\qquad \text{and } (\tilde{g}_1, \ldots, \tilde{g}_l) \in \tilde{m}^{I_l} \text{ with } g_k = \tilde{g}_1\};
\end{aligned}
$$

furthermore, the k-ary attributes $\bot_k$, $\top_k$, and id_k with $(\bot_k)^{I_k} := \emptyset$, $(\top_k)^{I_k} := G_k$, and $(id_k)^{I_k} := \{(g_1, \ldots, g_k) \mid g_1 = \cdots = g_k \in G_0\}$ are introduced as constants. In general, the *compound attributes* of the power context family $\mathbb{K}$ are derived from the unary attributes of $M_0 \cup M_1$, the k-ary attributes of M_k $(2 \leq k \leq n)$, and the constants by recursively applying the negation $\neg$, the permutations π, or the joins ι_1 or ι_2. For suitably choosing compound attributes, it is useful to know further operations derivable from the basic ones, for instance (cf. [Bu91]):

$$
\begin{aligned}
&(5) \quad (\kappa^i m)^{I_{k+1}} := \{(g_1, \ldots, g_{i-1}, g_i, g_i, g_{i+1}, \ldots, g_k) \mid (g_1, \ldots, g_k) \in m^{I_k}\},\\
&(6) \quad (\delta^i m)^{I_{k-1}} := \{(g_1, \ldots, g_{i-1}, g_{i+1}, \ldots, g_k) \mid (g_1, \ldots, g_k) \in m^{I_k}\},\\
&(7) \quad (\xi^{ij} m)^{I_{k-1}} := \{(g_1, \ldots, g_{j-1}, g_{j+1}, \ldots, g_k) \mid\\
&\qquad\qquad \exists g_j : (g_1, \ldots, g_k) \in m^{I_k} \text{ and } g_i = g_j\},\\
&(8)\ (\chi(m, \tilde{m}))^{I_{k+l}} := \{(g_1, \ldots, g_k, \tilde{g}_1, \ldots, \tilde{g}_l) \mid\\
&\qquad\qquad (g_1, \ldots, g_k) \in m^{I_k} \text{ and } (\tilde{g}_1, \ldots, \tilde{g}_l) \in \tilde{m}^{I_k}\};\\
&(9) \quad (\eta(m, \tilde{m}))^{I_l} := \{(g_1, \ldots, g_l) \mid (g_1, \ldots, g_l) \in \tilde{m}^{I_l} \text{ with } (g_1, \ldots, g_k) \in m^{I_k}\}.
\end{aligned}
$$

The so-called *comma operation* κ^i "is a briliant device that Peirce was using with great skill and effect" ([Bu91], p.76), the operation δ^i is Bernay's *"Streichung"* operation, and the operations ξ^{ij} and χ are called *hook identity* and *product* in [Bu91]. The *insertion operation* η allows us to activate attributes of smaller arity in contexts with objects of greater arity; for instance, $\eta(m, \top_l)$ is a compound attribute of $\mathbb{K}_l$ with the extent $\{(g_1, \ldots, g_l) \in G_l \mid (g_1, \ldots, g_k) \in m^{I_k}\}$.

How the *term formation* by the basic operations of PAL and their derivatives may be used to obtain a TOSCANA-system for a power context family, shall be demonstrated by our example of the domestic flights in Austria. Examples of conceptual scales for the power context family described in Section 2 are given by the labelled line diagrams in Figure 2, 3, and 4 in [PW99]. Figure 5 below shows a nested line diagram combining representations of two binary relational scales. The relational scale of the outer diagram is determined by the four binary attributes *From, To, Dept*, and *Arrv*, while the relational scale of the inner diagram has the binary compound attributes *Graz, Innsbruck, Klagenfurt, Linz, Salzburg, Vienna, a.m.,* and *p.m.* which are formed from unary attributes by using the insertion operation together with $\top_2$ and the conversion π_2 (i.e. extensionally, the exchange of place 1 and 2 in an object pair). The comparison

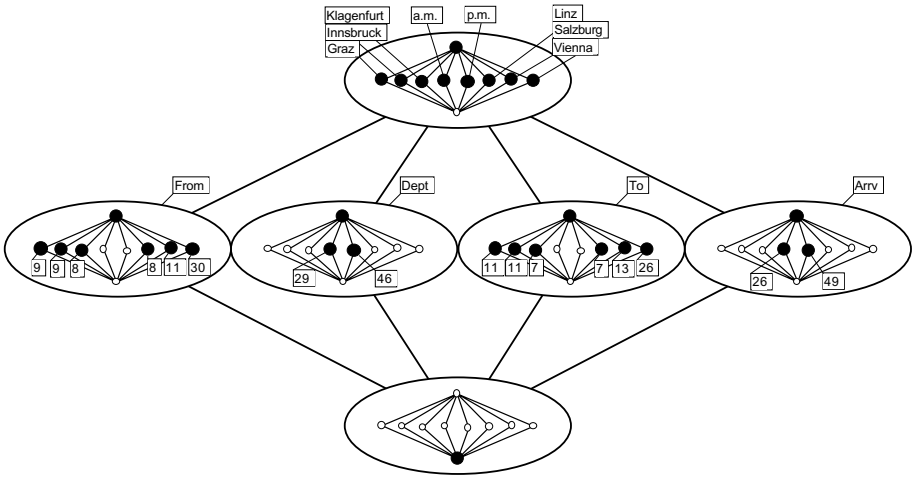


Fig. 5. The concept lattice of the apposition of two binary relational scales

of Figure 5 with Figure 2 makes clear the advantage of using relational scales. Of course, further relational attributes and scales (even of greater arity) are desirable; here we only mention the binary compound attribute $From\text{-}To := \iota_2(\pi_2(From), To)$, the application of which will be shown in the next section.

For using Peircean Algebraic Logic to create compound attributes and conceptual and relational scales based on a power context family, there is not only the argument that this logic combines well with the Contextual Logic as developed until now, but also *Burch's thesis* that "all procedures of relational constructions are formalizable in PAL" ([Bu91], p.122). This thesis is analogous to Church's thesis about computability, of course, less tested, but nevertheless convincing.

4 Graphical Representations

For a TOSCANA-system with a conceptual scheme containing conceptual and relational scales based on a power context family, there is still the question how to *readably represent* concept graphs derived from the conceptual scheme in connection with the actual database. Small concept graphs may be read off from labelled line diagrams of the concept lattices of appositions of relational scales, as the concept graphs about the flight connections between two airports from the nested line diagram in Figure 5. But a little bit larger concept graphs might already diminish the readability seriously. Then, Sowa's conventions for drawing conceptual graphs [So84] would be better for the graphical representation of concept graphs; for our example, such representation is given by Figure 6 in [PW99] showing the flight connections of a commuter between Innsbruck and Vienna.

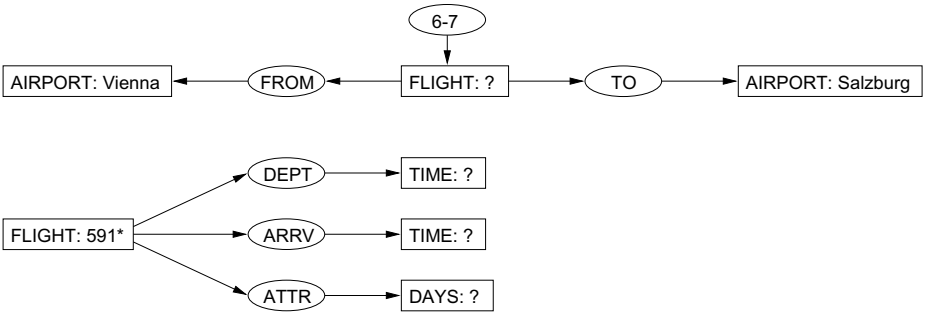


Fig. 6. Query graphs for retrieving flight information

Sowa's graph representations may also be used for formulating *queries* within a TOSCANA-system. This can be performed by choosing from presented line diagrams some conceptual and relational instances and by combining those instances to a graphically represented concept graph in which one or more references are replaced by question marks. The graphical representation of the *query graphs* is needed for checking them syntactically and semantically. Figure 6 shows some examples of querying flight information using concept graphs. In [GE99], it is discussed how those query graphs may be algorithmically tranformed into SQL statements for querying the database of the TOSCANA-system.

Since the intention for developing TOSCANA was always to offer the user rich information in a transparent way so that she can make her decisions based on the presented information which is restricted as little as possible, a TOSCANA-system should allow general queries having as answer a wide "landscape" of detailed facts. For instance, a user, living in Vienna, might ask for the best flight connections at the weekend between Saturday 7 a.m. and Sunday 8 p.m. for visiting Graz, Innsbruck, and Salzburg where she has to meet colleagues to discuss important documents. The full information, under the given restrictions, is represented in the concept graph shown in Figure 7. Obviously, such a representation would be too complex for the customary user. Therefore, we propose another graphical representation, shown in Figure 8, which yields the same flight information as the graph in Figure 7. The easily understood conventions of the shown *information map* are the following: A straight line connecting two towns indicates that there are direct flights between those towns and an arrowhead on such a line points toward their destination; a small table linked to an arrowhead gives the information about the relevant flights, their departure and arrival times, and the week-days they operate. The used graphical means can be described by PAL terms as follows: the arrows represent to the objects of the extent of $FromTo$, the link between an arrowhead and a flight number in the relevant table is characterized by the compound attribute $\xi^{13}(\chi(From, To))$, and the columns in the small tables correspond to the objects in the extent of the compound attribute $\eta(\neg\xi^{12}(\neg Sa, \neg Su), \xi^{14}(\chi(\xi^{13}(\chi(Dept, Arrv)), FlDays))$. The descriptions by PAL terms allow a contextual-logic management of the information maps so that they can be integrated in a TOSCANA-system.

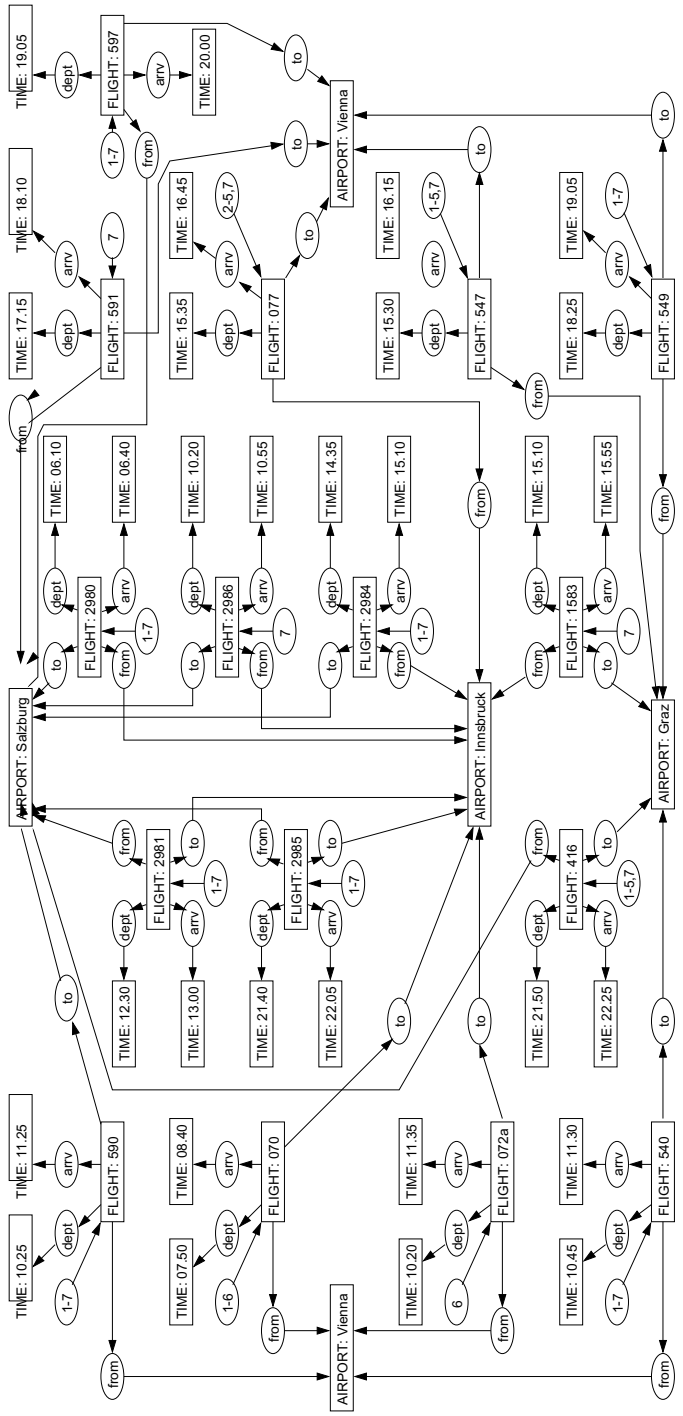


Fig. 7. Conceptual graph representing the weekend flights between Vienna, Graz, Innsbruck, and Salzburg

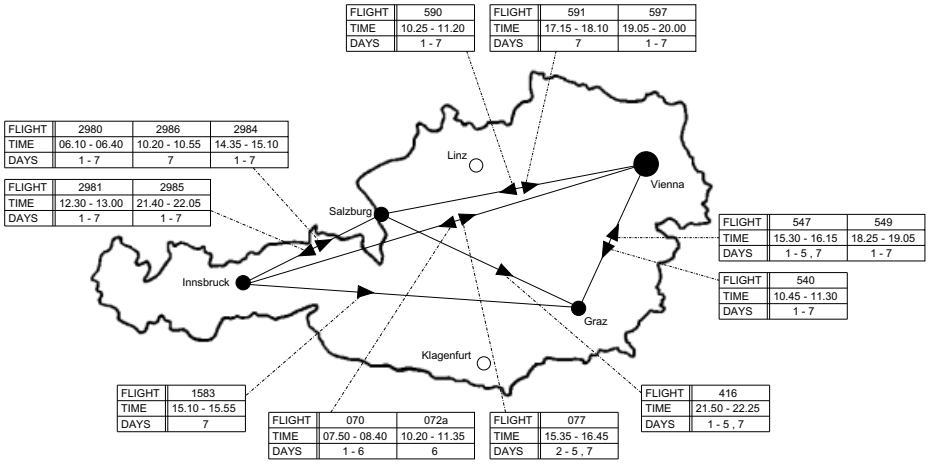


Fig. 8. Information map with the same flight information as the graph in Figure 7

An inspection of the information map immediately shows that tours via Graz to Innsbruck or Klagenfurt are impossible. Therefore, flights to Innsbruck and Salzburg with further connections have to be considered. Since there is only one flight from Salzburg to Graz late on Sunday, the best choices, giving reasonable periods of time for meeting colleagues, seem to be the flights 590 (Vienna-Salzburg), 2985 (Salzburg-Innsbruck), 1583 (Innsbruck-Graz), and 549 (Graz-Vienna); but there is also another solution, namely the flights 070 (Vienna-Innsbruck), 2984 (Innsbruck-Salzburg), 597 (Salzburg-Vienna), 540 (Vienna-Graz), and 549 (Graz-Vienna). Now, the user has to decide, using further information and preferences, for instance, about staying over night in a hotel or at home and about the best possible times for the meetings.

The example teaches us that the development of a TOSCANA-system has to use as much as possible the background knowledge of the potential users, especially, for obtaining a satisfactory human-machine interface. The tremendous increase of readability by changing from Figure 7 to 8 has its explanation in the common knowledge in our present culture, in particular, concerning the use of geographical maps and spatial reasoning. For instance, straight lines between towns with arrowheads on them may be intuitively understood as flight routes, and rows in a table, linked to a flight route, which have two numbers after the word "Time" are identified as the corresponding departure and arrival time without any difficulty (notice that the activated background knowledge has to be made explicit in Figure 7). Of course, the labelled line diagrams of concept lattices are less easily read and require some practice, but our experience is that customers, who are familar with the application domain and are interested in the presented information, understand labelled line diagram astonishingly well. Therefore, also in the future, TOSCANA-systems will use labelled line diagrams for presenting information, but will combine them with information maps and other graphical representations.

Acknowledgement. This paper has substantially benefited from the financial support given by the Australian Research Council (IREX-Programme) which allowed the fourth author to work for two months in 1999 at the School of Information Technology of the Griffith University (Gold Coast Campus).

References

Bu91. R. W. Burch: *A Peircean reduction thesis*. Texas Tech University Press, Lubbock 1991.

GW99. B. Ganter, R. Wille: *Formal Concept Analysis: Mathematical Foundations*. Springer, Heidelberg 1999. (Translation of: *Formale Begriffsanalyse: Mathematische Grundlagen*. Springer, Heidelberg 1996.)

GE99. B. Groh, P. Eklund: Algorithms for creating relational power context families from conceptual graphs. In: W. Tepfenhart, W. Cyre (eds.): *Conceptual Structures: Standards and Practices*. LNAI **1640**. Springer, Heidelberg 1999, 389–400.

GSW98. B. Groh, S. Strahringer, R. Wille: TOSCANA-systems based on thesauri. In: M.-L. Mugnier, M. Chein (eds.): *Conceptual Structures: Theory, Tools and Applications*. LNAI **1453**. Springer, Heidelberg 1998, 127–138.

KSVW94. W. Kollewe, M. Skorsky, F. Vogt, R. Wille: TOSCANA – ein Werkzeug zur begrifflichen Analyse und Erkundung von Daten. In: R. Wille, M. Zickwolff (eds.): *Begriffliche Wissensverarbeitung – Grundfragen und Aufgaben*. B. I.-Wissenschaftsverlag, Mannheim 1994, 267–288.

ME99. Ph. Martin, P. Eklund: Embedding knowledge in Web documents. *Computer Networks* **31** (1999), 1403–1409.

Na99. NaviCon: TOSCANA 2.1 – Benutzerhandbuch. NaviCon GmbH, Frankfurt 1999.

Pr97. S. Prediger: Logical scaling in Formal Concept Analysis. In: D. Lukose, H. Delugach, M. Keeler, L. Searle, J. F. Sowa (eds.): *Conceptual structures: fulfilling Peirce's dream*. LNAI **1257**. Springer, Heidelberg 1997, 332–341.

Pr98. S. Prediger: *Kontextuelle Urteilslogik mit Begriffsgraphen. Ein Beitrag zur Restrukturierung der mathematischen Logik*. Dissertation, TU Darmstadt. Shaker Verlag, Aachen 1998.

PW99. S. Prediger, R. Wille: The lattice of concept graphs of a relationally scaled context. In: W. Tepfenhart, W. Cyre (eds.): *Conceptual Structures: Standards and Practices*. LNAI **1640**. Springer, Heidelberg 1999, 401–414.

So84. J. F. Sowa: *Conceptual structures: Information processing in mind and machine*. Adison-Wesley, Reading 1984.

SW00. G. Stumme, R. Wille (eds.): *Begriffliche Wissensverarbeitung: Methoden und Anwendugen*. Springer, Heidelberg 2000.

Vo96. F. Vogt: *Datenstrukturen und Algorithmen zur Formalen Begriffsanalyse: Eine C++–Klassenbibliothek*. Springer, Heidelberg 1996.

VW95. F. Vogt, R. Wille: TOSCANA – A graphical tool for analyzing and exploring data. In: R. Tamassia, I. G. Tollis (eds.): *GraphDrawing '94*. LNCS **894**. Springer, Heidelberg 1995, 226–233.

Wi97. R. Wille: Conceptual Graphs and Formal Concept Analysis. In: D. Lukose, H. Delugach, M. Keeler, L. Searle, J. F. Sowa (eds.): *Conceptual Structures: Fulfilling Peirce's Dream*. LNAI **1257**. Springer, Heidelberg 1997, 290–303.

Wi00. R. Wille: *Contextual logic summary*. FB4-Preprint, TU Darmstadt 2000.

A Conceptual Graph Model for W3C Resource Description Framework

Olivier Corby, Rose Dieng, and Cédric Hébert

INRIA Sophia Antipolis
{Olivier.Corby,Rose.Dieng}@inria.fr,
WWW home page: http://www.inria.fr/acacia
Acacia Project
2004 route des Lucioles, BP 93
06902 Sophia Antipolis Cedex
France

Abstract. With the aim of building a "Semantic Web", the content of the documents must be explicitly represented through metadata in order to enable contents-guided search. Our approach is to exploit a standard language (RDF, recommended by W3C) for expressing such metadata and to interpret these metadata in conceptual graphs (CG) in order to exploit querying and inferencing capabilities enabled by CG formalism. The paper presents our mapping of RDF into CG and its interest in the context of the semantic Web.

1 Introduction

The Web is recognized as a fabulous information repository, with millions of heterogeneous information sources available throughout the world. But the existing keyword-based search engines do not take into account the semantics of the documents accessible through the Web. The user can be easily overwhelmed by the huge number of answers (not always relevant) to a query. Therefore, the need of a "Semantic Web" is more and more emphasized [2,3]. The semantics of the documents must be explicitly represented through semantic metadata in order to enable semantic-contents-guided search [1]. Several proposals have been offered to this end, for example Ontobroker [8] and Shoe [12], that rely on extensions of HTML and exploitation of ontologies. In [9], the authors analyse several languages that may be used for representing metadata. They notice the following problems to be solved when dealing with large amounts of semi-structured information: searching information, extracting information, maintaining weakly structured sources, generating documents. They emphasize the importance of relying on standards that are widely accepted by the Web community. [9] stresses that knowledge representation languages offered in artificial intelligence (AI) seem attractive for this aim, but suffer from the lack of wide acceptance. We are convinced of the interest of AI representation languages that enable not only the representation of metadata but also support inferences on them. Among such AI knowledge representation formalisms, [13,14] stress the advantages of

B. Ganter and G.W. Mineau (Eds.): ICCS 2000, LNAI 1867, pp. 468–482, 2000.
© Springer-Verlag Berlin Heidelberg 2000

conceptual graph (CG) formalism for expressing metadata. Another approach is to exploit a standard language for expressing metadata and to be able to interpret these metadata in conceptual graphs in order to exploit querying and inferencing capabilities enabled by conceptual graph formalism.

RDF (Resource Description Framework) has been introduced and recommended by the World Wide Web Consortium (W3C) to enable descriptions of semantic metadata for the Semantic Web. RDF enables the addition of semantic information to a Web document, without making any assumptions about the structure of this document. The future will reveal whether RDF will be accepted as a standard for content descriptions of Web resources and whether it will be widely used by the authors of Web documents. In the event RDF is broadly accepted, the approach of automatically interpreting metadata expressed in RDF into conceptual graphs seems interesting: it will enable us both to rely on a standard and draw benefit from the advantages of conceptual graphs.

The purpose of the paper is to show that conceptual graphs can be used as a means to exploit RDF metadata to handle metadata-based search queries. After a description of the principles of RDF, we will detail the mapping of RDF and CG. Then, we discuss the interest of this approach in comparison to related work.

2 RDF

RDF is based on an underlying model with triples made of resource, property, and value.

- A resource is an entity accessible by an URI on the Web (e.g. an HTML or XML document). Resources are the elements described by RDF statements.
- A property defines a binary relation between resources and/or atomic values. A property enables us to attach information to resources, and provide descriptions for resources.
- A value can be either a simple character string or a resource. Reification enables one to transform a triple into a resource. The notion of collections permits us to define groups to which some properties are applied.

An RDF statement specifies a value for a property of a resource.

RDF has an XML syntax and can be seen as an object-oriented formalism for metadata statements. These metadata can rely on common ontologies represented using RDF Schema (RDFS).

RDF statements can be considered as triples (resource, property, value). The vocabulary used in these triples can be defined using RDFS, by a hierarchy of classes and a hierarchy of properties.

Contrary to object-oriented or frame-based representations, RDF relies on a property-centric approach. Anyone can define properties about Web resources, in order to offer descriptions for these resources. In RDFS, properties are defined globally and not encapsulated in class definitions. They can be specialized using the subPropertyOf relationship. RDF/S offers three core classes:

– Resource (i.e. class of all objects),
– Property (i.e. class of all properties),
– Class (i.e. class of all classes).

Two core properties are provided: type and subClassOf. The classes can be specialized through the subClassOf relationship. A resource is said to be an instance of a given class by means of the type property. The range and domain core properties are used to define the range (resp. domain) of properties.

3 Mapping RDF to CG

The model of CG formalism [17,6] is based on (1) a support made of a concept type lattice and of a relation type set possibly organized in hierarchy, a set of individual markers enabling the designation of instances, a conformity relation between markers and types, and (2) a base of conceptual graphs built on this support.

It therefore seems natural to translate a) the RDF statements into a base of CG-facts b) the hierarchy of classes appearing in an RDF schema into a concept type hierarchy in CG, and c) the hierarchy of properties appearing in a RDF schema into a relation type hierarchy in CG. Therefore we will rely on a CG model enabling us to build a relation type hierarchy.

3.1 Mapping of Basic RDF

A basic RDF statement says something like : 'the author of the resource found at `http://www.bookstore.org/id1971` is `John Rawls`'. It can be stated as a triple by this way :

```
author(http://www.bookstore.org/id1971, 'John Rawls')
```

Several statements can be written about the same resource, for example :

```
title(http://www.bookstore.org/id1971, 'A Theory of Justice')
date(http://www.bookstore.org/id1971, '1971')
```

Written with the RDF/XML syntax :

```
<rdf:Description about='http://www.bookstore.org/id1971'>
    <author>John Rawls</author>
    <title>A theory of Justice</title>
    <date>1971</date>
</rdf:Description>
```

This can be interpreted in CG as :

```
[Resource : http://www.bookstore.org/id1971] - {
    -> (author) -> [Literal : John Rawls]
    -> (title) -> [Literal : A theory of Justice]
    -> (date) -> [Literal : 1971]}
```

The principle of the mapping relies on considering an RDF description as an instance of a Resource CG concept type and the associated properties as relations of this concept. The designator of a Resource concept is the URI of the resource itself.

3.2 Mapping of Nested RDF Descriptions

The value of a basic RDF triple can also be an RDF description. For example, we can express that the value of the `bio` property is itself a description of another resource :

```
<rdf:Description about='http://www.bookstore.org/id1971'>
  <title>A theory of Justice</title>
  <author>John Rawls</author>
  <bio>
     <rdf:Description about='http://www.bookstore.org/John.Rawls'>
          <position>Philosopher</position>
     </rdf:Description>
  </bio>
</rdf:Description>
```

This RDF description will be translated into the following CG:

```
[Resource : http://www.bookstore.org/id1971] - {
   -> (title) -> [Literal : A theory of Justice]
   -> (author) -> [Literal : John Rawls]
   -> (bio) -> [Resource : http://www.bookstore.org/John.Rawls] ->
                    (position) -> [Literal : Philosopher]}
```

In case of nested RDF descriptions, the mapping consists of creating a Resource-typed concept for each nested resource description. Each nested resource concept is then linked to its embedded resource via a relation. For example, in the previous example, the nested resource `http://www.bookstore.org/John.Rawls` is linked by means of a `bio` relation to the embedding resource `http://www.bookstore.org/id1971`.

3.3 RDF Schema

RDF descriptions can be typed according to a predefined ontology called a RDF schema. Thus RDFS formalism enables provision of a vocabulary used for the RDF annotations. For example, the previous description can be typed as a description of a book, the class Book being itself defined in an RDF schema.

```
<rdf:RDF xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
         xmlns:ns='http://www.inria.fr/acacia/iccs#'>
    <ns:Book rdf:about='http://www.bookstore.org/id1971'>
        <ns:author>John Rawls</ns:author>
```

```
        <ns:title>A theory of Justice</ns:title>
    </ns:Book>
</rdf:RDF>
```

The leading RDF markup with a `xmlns:rdf` attribute defines the RDF namespace `rdf` as a shortcut for the RDF URI, namely : `http://www.w3.org/-1999/02/22-rdf-syntax-ns#`. Each RDF markup must be prefixed with this namespace in order to identify what is RDF-related and what is application-specific. An XML namespace is also associated with the application schema, (say `ns`), in order to identify the markup correctly.

The corresponding CG is the following:

```
[Book : http://www.bookstore.org/id1971] - {
    -> (author) -> [Literal : John Rawls]
    -> (title) -> [Literal : A theory of Justice]}
```

Accordingly, when the RDF description is typed by a specific RDF Schema class, a concept of the corresponding type is created. In the example, the Resource is of type Book, hence a Book concept is created.

Concept and relation type names must be prefixed in a unique way, with a URI, to prevent name clashes from different schemas. For example, the concept type `Book` should be named : `http://www.inria.fr/acacia/iccs#Book` and the relation type `author` : `http://www.inria.fr/acacia/iccs#author`. For the sake of readability, we skip these prefixes for CG in the paper, but they are mandatory in the implementation.

4 Mapping RDF Schema

The RDF Schema, according to the current W3C Candidate Recommendation from March 2000 [5], allows the definition of classes and properties. Classes and properties can be refined in subclasses and subproperties. In RDF, properties are first class objects which exist by themselves. Hence, new properties can be added to existing classes in order to enable reuse of these classes.

4.1 Mapping of Classes

RDFS classes can be modelled as CG concept types. In order to map the Resource core class of RDFS, we introduce a Resource concept type at the top level of the CG concept type hierarchy.

```
concept type Resource
```

A RDFS class without any superclass explicitly indicated will be modelled by a subtype of Resource in the CG concept type hierarchy.

```
<rdfs:Class rdf:ID='Cl'/>
```

This can be translated into,

```
concept type Cl < Resource
```

For example :

```
<rdfs:Class rdf:ID='Book'/>
```

can be modelled as :

```
concept type Book < Resource
```

4.2 Mapping Subclasses

The subClassOf relation between classes in RDFS corresponds to the subtype relation (denoted ¡) between concept types in CG formalism. If an RDFS class Cl2 is defined as a subclass of Cl1, it will be modelled by a Cl2 concept type, subtype of the Cl1 concept type in the CG concept type hierarchy.

```
<rdfs:Class rdf:ID='Cl2'>
     <rdfs:subClassOf rdf:resource='#Cl1'/>
</rdfs:Class>
```

This can be translated into :

```
concept type Cl2 < Cl1
```

The Novel subclass of Book :

```
<rdfs:Class rdf:ID='Novel'>
     <rdfs:subClassOf rdf:resource='#Book'/>
</rdfs:Class>
```

can be modelled as a subtype of Book:

```
concept type Novel < Book
```

4.3 Mapping of Properties

A property is defined according to a domain (i.e. a class) and has an associated range that can be a literal or a class. For example, the title property can be defined with 'Book' domain and 'Literal' range. 'Literal' is prefixed with the RDFS namespace.

```
<rdf:Property ID='title'>
     <rdfs:domain rdf:resource='#Book'/>
     <rdfs:range rdf:resource='http://www.w3.org/TR/1999/PR-rdf-
         schema-19990303#Literal'/>
</rdf:Property>
```

A property definition can be modelled as a CG binary relation type with an associated signature that maps the domain and range to the related concept types. For instance, the previous example is translated into:

```
relation type title (Book, Literal)
```

4.4 Mapping of SubProperties

In RDFS, a subproperty can refine an existing property, in the same way as a subclass refines a class. It means that if p2 is a subproperty of p1, then :

```
p2(URI, v) => p1(URI, v)
```

that is, if a URI has value v for property p2, then it also has v as value for property p1.

For example, we define here the author property and its jointAuthor subproperty :

```
<rdf:Property ID='author'>
    <rdfs:domain rdf:resource='#Book'/>
    <rdfs:range rdf:resource='http://www.w3.org/TR/1999/PR-rdf-
          schema-19990303#Literal'/>
</rdf:Property>

<rdf:Property ID='jointAuthor'>
        <rdfs:subPropertyOf rdf:resource='#author'/>
</rdf:Property>
```

In terms of CGs, an RDF subproperty is translated into a relation type that is a subtype of the relation type translating the superproperty. In the example above, the jointAuthor relation type will be defined as a subtype of the author relation type.

```
relation type author (Book, Literal)
relation type jointAuthor < author
```

4.5 Remark: Limits of this Mapping

Let us notice a limitation in the mapping from RDF to CG :

In the RDF Schema, a property can be defined with several classes as domain. However, this is not possible directly in CG because a relation type has only one signature. If this happens, three possibilities are offered to solve the problem :

1. If the domain classes have a common RDF Schema class ancestor, set the relation domain to that ancestor,
2. or else, define an abstract common superclass to all domain classes of the property and assign this new class as the domain of the relation signature. This is only possible if one masters the RDF Schema(s).
3. otherwise, set the domain of the property to Resource, the top level CG concept type that matches all resources. The property will then be allowed for all resources.

For example :

```
<rdf:Property ID='title'>
    <rdfs:domain rdf:resource='#Book'/>
    <rdfs:domain rdf:resource='#Show'/>
    <rdfs:range rdf:resource='http://www.w3.org/TR/1999/PR-rdf-
         schema-19990303#Literal'/>
</rdf:Property>
```

Solution 2 would consist of defining an abstract class, called 'Work' for example, having 'Book' and 'Show' as subclasses and, then, define the title relation type with the following signature :

```
concept type Work
concept type Book < Work
concept type Show < Work

relation type title (Work, Literal)
```

Otherwise, solution 3 would consist of defining title as :

```
relation type title (Resource, Literal)
```

4.6 Implementing the RDF Schema Metamodel

The RDF metamodel is itself described in RDF, thus enabling the extension of the metamodel, by refining the predefined classes and properties.

For example, in RDF Schema, it is possible to define a 'Concept' metaclass that refines the 'Class' metaclass, and then define a schema in terms of 'Concept' instead of 'Class' :

```
<rdfs:Class rdf:ID="Concept">
    <rdfs:subClassOf="http://www.w3.org/TR/1999/PR-rdf-
          schema-19990303#Class/>
</rdfs:Class>

<ns:Concept rdf:ID="Paper"/>
```

We have implemented the whole RDF and RDF Schema metamodel in the CG support and enable metamodel extension.

5 More Advanced Features

5.1 Reification

A reified statement is an RDF statement about an RDF statement, as for example :

John K. Galbraith says :
'The author of the resource http://www.bookstore.org/id1971 is John Rawls'.

In RDF, the treatment of such a reified statement is somehow cumbersome and introduces specialized properties as shown below :

```
<rdf:Description>
      <rdf:subject resource="http://www.bookstore.org/id1971"/>
      <rdf:predicate resource="#author"/>
      <rdf:object>John Rawls</rdf:object>
      <rdf:type resource="http://www.w3.org/1999/02/22-rdf-
           syntax-ns#Statement"/>
      <attributedTo>John K. Galbraith</attributedTo>
</rdf:Description>
```

In CG, this can be done simply with a context named Statement :

```
[Statement: [Book: http://www.bookstore.org/id1971] -> (author) ->
                  [Literal:John Rawls]
] -> (attributedTo) -> [Literal : John K. Galbraith]
```

Reified RDF statements, i.e. : descriptions that are instances of the Statement class, are translated into CG by means of a predefined context named Statement. The RDF description itself is translated into CG following the standard mapping.

5.2 Mapping of Containers

The model of basic RDF relies on triples with single values. In the case where the value of a property is a set of values, the W3C recommendation defines containers such as bags, sequences and alternatives in order to hold such values. For example here, the authors are given as a set of names :

```
<rdf:Paper rdf:about='http://www.inria.fr/acacia/iccs2000'>
    <ns:authors>
        <rdf:Bag>
            <rdf:li>C\'edric H\'ebert</rdf:li>
            <rdf:li>Olivier Corby</rdf:li>
            <rdf:li>Rose Dieng</rdf:li>
        </rdf:Bag>
    </ns:authors>
</rdf:Paper>
```

Containers can be handled by an adequate Bag concept which is related to its members by means of a member relation, called `rdf:li` :

```
[Paper:http://www.inria.fr/acacia/iccs2000]
   -> (authors) -> [Bag]-{
```

```
    -> (rdf:li) -> [Literal: C\'edric H\'ebert]
    -> (rdf:li) -> [Literal: Olivier Corby]
    -> (rdf:li) -> [Literal: Rose Dieng]}
```

We introduce in the CG concept type hierarchy an abstract 'Container' concept type and three subtypes for the concrete containers (see the appendix on types at the end of the paper). We also introduce a `rdf:li` relation type :

```
relation type rdf:li (Container, Resource)
```

5.3 Mapping Containers Having aboutEach Statements

RDF offers a means to factorize statements that apply to all members of a container. This is done with an aboutEach statement. In the example below, the bag is given the `auth` ID, and it is said that all members of the `auth` bag have INRIA as institute :

```
<rdf:Paper rdf:about='http://www.inria.fr/acacia/iccs2000'>
    <ns:authors>
        <rdf:Bag ID='auth'>
            <rdf:li>C\'edric H\'ebert</rdf:li>
            <rdf:li>Olivier Corby</rdf:li>
            <rdf:li>Rose Dieng</rdf:li>
        </rdf:Bag>
    </ns:authors>
</rdf:Paper>

<rdf:Description aboutEach='#auth'>
        <ns:inst>INRIA</ns:inst>
</rdf:Description>
```

We can translate on the fly to distribute the factorized value to all bag members :

```
[Paper: http://www.inria.fr/acacia/iccs2000]
 -> (authors) -> [Bag]-{
   -> (rdf:li) -> [Literal:C\'edric H\'ebert]
      -> (inst) -> [Literal:INRIA]
   -> (rdf:li) -> [Literal:Olivier Corby]
      -> (inst) -> [Literal:INRIA]
   -> (rdf:li) -> [Literal:Rose Dieng]
      -> (inst) -> [Literal:INRIA]}
```

6 Querying

The main interest of mapping RDF to CG is the adequacy between the two models, i.e. concepts and relations smoothly map onto classes and properties

that are defined independently in CG as well as in RDF. Furthermore, it enables us to use RDF without any knowledge of Conceptual Graphs.

The second reason is the relevance of the CG projection operation to querying a RDF/CG base. Querying RDF metadata consists of retrieving RDF triples belonging to classes, taking specialization into account. This can be done through the projection operation.

Furthermore, thanks to the implementation platform that we have chosen, namely F. Southey's NOTIO [16], it is possible to parametrize precisely the graph matching process. Hence, it is possible to tune concept matching, including type and instance matching. For example, concepts may match according to (at least) one of the four conditions on concept types :

- first type is a supertype of the second
- first type is a subtype of the second
- first type is either a subtype or a supertype of the second
- concepts have same type.

Relation matching can also be parametrized, as well as other aspects of the graph matching. This functionality is well adapted to metadata information retrieval as it authorizes approximate matching along specialization and generalization and on relations and concepts.

In the current prototype, the query language is RDF itself. The user describes a partial RDF statement that he is looking for. The RDF query may hold variables, prefixed by '?', to indicate the parts that are unknown, the value of which should be returned by the query processor.

For example, let's look for books the author of which is John Rawls, and return their title :

```
<ns:Book rdf:about="?1">
    <ns:author>John Rawls</ns:author>
    <ns:title>?2</ns:title>
</ns:Book>
```

The RDF query is translated into the graph shown below :

```
[Book] - {
    -> (author) -> [Literal : John Rawls]
    -> (title) -> [Literal]}
```

The query processor projects the query graph on the CG base. The resulting (sub)graphs are translated back into RDF in order to be presented to the user in a uniform way.

The prototype also implements approximate search on literal values, thanks to the NOTIO matching scheme that enables us to attach customized match comparators to markers. We implemented such an approximate comparator that tests whether the query literal value is included into the graph literal value. Approximate query values are prefixed by the '∼' character.

It is then possible to send a query that searches an author, the value of which contains 'Rawls', as shown below.

```
<ns:Book rdf:about="?1">
    <ns:author>~Rawls</ns:author>
    <ns:title>?2</ns:title>
</ns:Book>
```

If several properties implement the *author* relationship, e.g. author, joint authors, etc. the problem may arise of taking all of them into account for query processing. In fact, RDF enables the association of a common external label to all these properties, say `author`. An advanced query GUI would be able to propose the abstract `author` property to the user and translate it to several CGs according to the signatures of the target author relation signatures (container, literal, etc.). In any case, this problem is relative to RDF, not to the CG translation model.

7 Implementation

We implemented a prototype using the Notio CG platform [16] and the VRP RDF parser from ICS Forth [19]. We exploited the possibility offered by Notio for parametrizing the projection (cf. generalization, specialization). Our prototype can translate classes and properties of RDF Schema and RDF statements, except the aboutEachPrefix RDF statement that is not presented here.

The translation of the RDF metadata into a base of CG-facts can be done automatically thanks to our prototype. Several ways of integrating such an automatic translator RDF-¿CG in a Web search engine can be thought out :

- A robot could access the Web documents and build a base of CG-facts corresponding to the translation into CG of their RDF metadata. The link between the CG-facts associated to a document and the document would be kept by the robot. This would be done before any requests from a user. Then, when a user makes a request to search a given document, this request would be translated into a CG-query. The parameters of the user's request (in particular, if the user wants to obtain approximate answers by enabling generalization or specialization) can also be exploited to parametrize the projection to be used. The results of the projection of the CG-query on the base of CG-facts constitute the answers to the user's request.
- Another possibility would be to let the search engine use the translator to build the CG dynamically, only after a request by the user.

The prototype currently runs as a Java servlet, accessed by means of a standard Internet navigator at a given URI. The user can type a query and send it to the system which performs the projection and sends back the answer in RDF. The resulting RDF statements are displayed by means of an XSLT stylesheet, thanks to James Clark's XT engine [7]. The URI contained in the resulting RDF statements are transformed into active HTML links on which the user may click. Hence, the whole process implements a conceptual search engine.

8 Conclusion and Discussion

Our approach delivers a more powerful and relevant search with the CG projection. In particular, the parametrization of the projection enables several levels of search.

Our approach takes advantage of the CG formalism, as [13,14], but without requiring the author of the document to know CG. The interest of our approach is that if RDF, recommended by W3C, is widely adopted as a standard by the Web community, then a Web document author can both continue to use RDF annotations and draw benefit from the CG formalism, even without knowing himself the CG formalism.

The exploitation of RDF schemas by means of Conceptual Graphs seems more relevant in the context of a company or of a given community: this company or community can agree on the conceptual vocabulary used for expressing the metadata about their documents.

In the future, we plan to study a query language for RDF statements and the mapping to appropriate CG projections.

Acknowledgements

We thank Ahmed Amerkad who implemented the CG to RDF translator, Peter Eklund, Philippe Martin and Denis Kuntz for proof-reading the paper, the remaining errors being ours. We also thank the Escrire research action for fruitful discussions and partial funding of this project.

References

1. T. Berners-Lee. Metadata Architecture. `http://www.w3.org/DesignIssues/Metadata.html`
2. T. Berners-Lee. Semantic Web Road map. `http://www.w3.org/DesignIssues/Semantic.html`
3. T. Berners-Lee, D. Connolly, R. R. Swick. Semantic Web. `http://www.w3.org/1999/04/WebData.html`
4. T. Bray, J. Paoli and C. M. Sperber-McQueen, Extensible Markup Language (XML) 1.0, W3C Recommendation, `http://www.w3.org/TR/REC-xml`, 1998
5. D. Brickley and R.V. Guha eds. Resource Description Framework (RDF) Schema Specification 1.0, W3C Candidate Recommendation 27 March 2000, `http://www.w3.org/TR/rdf-schema`
6. M. Chein, M. L. Mugnier. Conceptual Graphs: Fundamental Notions. Revue d'Intelligence Artificielle, vol. 6, n. 4, p. 365-406, 1992.
7. J. Clark. The XT Extensible Stylesheet Language (XSLT) engine. `http://www.jclark.com`.
8. D. Fensel, S. Decker, M. Erdmann, R. Studer. Ontobroker: Or How to Enable Intelligent Access to the WWW. In B. Gaines, M. Musen eds, Proc of the 11th Workshop on Knowledge Acquisition, Modeling and Management (KAW'98), Banff, Canada, April 18-23. `http://ksi.cpsc.ucalgary.ca/KAW/KAW98/fensel1`

9. F. Van Harmelen, D.Fensel. Practical Knowledge Representation for the Web. Proc. of IJCAI'99 Workshop on Intelligent Information Integration, Stockholm, Sweden, 1999. http://www.cs.vu.nl/~frankh/postscript/IJCAI99-III.html
10. O. Lassila 1998, Web Metadata: A Matter of Semantics. IEEE Internet Computing, July-August 1998, Vol. 2, No. 4.
11. O. Lassila and R. R. Swick eds. Resource Description Framework (RDF) Model and Syntax Specification, W3C Recommendation 22 February 1999, http://www.w3.org/TR/REC-rdf-syntax.
12. S. Luke, L. Spector, D. Rager, J. Hendler. Ontology-based Web Agents. In Proc. of the First Int. Conference on Autonomous Agents, 1997.
13. P. Martin and P. Eklund. Embedding knowledge in web documents: CGs versus XML metadata languages. Proc. of the 7th Int. Conf. on Conceptual Graphs (ICCS'99), Springer-Verlag, August 1999.
14. P. Martin and P. Eklund. Embedding knowledge in web documents. Proc. of the 8th Int. World Wide Web Conf. (WWW8), p. 324-341, Elsevier, 1999.
15. A. Michard. XML : Langage et applications. Eyrolles, 1999.
16. F. Southey, Univ. of Guelf, Canada, http://dorian.cis.uoguelph.ca/CG/projects/notio, 1999
17. J. F. Sowa. Conceptual Structures: Information Processing in Mind and Machine. Addison-Wesley, Reading, 1984.
18. J. F. Sowa. Conceptual Graph Standard, http://www.bestweb.net/~sowa/cg/cgdpans.htm, 1999.
19. K. Tolle. The VRP RDF Parser, ICS Forth, Greece, http://www.ics.forth.gr/proj/isst/RDF

Appendix : The Type System

This appendix presents (part of) the type system as it is implemented in the prototype. We also present (part of) the metamodel as it is implemented. These meta types are not intended to be instantiated in conceptual graphs, but only to be specialized in meta schema.

```
rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
rdfs='http://www.w3.org/TR/1999/PR-rdf-schema-19990303#'
concept type rdf:Thing
concept type rdfs:Literal   < rdf:Thing
concept type rdfs:Resource  < rdf:Thing
concept type rdfs:Container < rdfs:Resource
concept type rdf:Bag        < rdfs:Container
concept type rdf:Alt        < rdfs:Container
concept type rdf:Seq        < rdfs:Container
concept type rdf:Statement  < rdfs:Resource
relation type rdf:Property(rdfs:Resource, rdf:Thing)
relation type rdf:li(rdfs:Container, rdf:Thing) < rdf:Property

concept type rdfs:Class   < rdfs:Resource
concept type rdf:Property < rdfs:Resource
rel.type rdf:type(rdfs:Resource,rdfs:Class)     < rdf:Property
```

```
rel.type rdfs:subClassOf(rdfs:Class,rdfs:Class) < rdf:Property
rel.type rdfs:subPropertyOf(rdf:Property,
                                  rdf:Property)<rdf:Property
rel.type rdfs:ConstProperty(rdfs:Resource,rdf:Thing)<rdf:Property
rel.type rdfs:domain(rdf:Property,rdfs:Class)< rdfs:ConstProperty
rel.type rdfs:range(rdf:Property,rdf:Thing) < rdfs:ConstProperty
rel.type rdf:object(rdf:Statement,rdf:Thing)       < rdf:Property
rel.type rdf:subject(rdf:Statement,rdfs:Resource)  < rdf:Property
rel.type rdf:predicate(rdf:Statement,rdf:Property) < rdf:Property
```

Computing with Conceptual Structures

Bernhard Ganter

Dresden University of Technology
ganter@math.tu-dresden.de

Abstract. We give an overview over the computational tools for conceptional structures that have emerged from the theory of Formal Concept Analysis, with emphasis on basic ideas rather than technical details. We describe what we mean by conceptual computations, and try to convice the reader that an elaborate formalization is a necessary precondition. Claiming that Formal Concept Analysis provides such a formal background, we present as examples two well known algorithms in very simple pseudo code. These can be used for navigating in a lattice, thereby supporting some prototypical tasks of conceptual computation. We refer to some of the many more advanced methods, discuss how to compute with limited precision and explain why in the case of incomplete knowledge the conceptual approach is more efficient than a combinatorial one. Utilizing this efficiency requires skillful use of the formalism. We present two results that lead in this direction.

1 Introduction: Conceptual vs. Computational

The basic meaning of "to compute" is to perform arithmetic operations on numbers in order to obtain a numerical result. To solve a difficult problem, highly complex computations may be necessary, but fortunately, computations can be done efficiently and with high precision. They lead to reliable results, at least when they are carried out by a *computer*, i.e., by computing machinery. Human beings tend to count computations as their less desirable activities. As units of (human) thought we prefer *concepts* over numbers, grouping objects according to their common attributes. Typical operations of conceptual knowledge processing are generalization and subsumption, judgement and conclusion, rather than addition and multiplication.

Numerical computations are useful for certain purposes, in particular for describing the physical word in terms of counting and measuring. When they are appropriate, their power often goes far beyond that of merely verbal argumentation. It is common belief that computations are objective, and that conceptual conclusions are not. Most objects of our thought, however, cannot naturally be expressed by numbers, and therefore numerical computations are of very limited use. A good indication that it is not straightforward to use numerical methods for conceptual purposes is the example of computers themselves: their computational power and accuracy stands in sharp contrast to the incomprehensibility and untrustworthiness of their operating instructions and manuals.

B. Ganter and G.W. Mineau (Eds.): ICCS 2000, LNAI 1867, pp. 483–495, 2000.
© Springer-Verlag Berlin Heidelberg 2000

There were many attempts to combine the advantages of computation with those of other knowledge processing techniques, from Aristotelian logic to Logic Programming. Logical calculi and algorithms were invented that support automated reasoning, but also numerical descriptions of concepts, based on similarity and dimensionality, are widely used. Yet these approaches have not led to a satisfactory "mathematics of concepts". In fact, there are warnings that such attempts may be in vain:

> *Mathematicians wish to treat matters of perception mathematically, and make themselves ridiculous . . . the mind . . . does it tacitly, naturally, and without technical rules.*
>
> Blaise Pascal (1670), cited from [3].

In fact, mental processes cannot be reduced to simple operations similar the the operations of arithmetic. The power of modern computing machinery has not been sufficient to fully emulate the mental abilities even of a snail or a butterfly. We have learnt to be modest when exploring the "technical rules" of the brain.

Nevertheless, there has been some progress since B. Pascal. We have a better understanding of the principles of mental operations, and we can formulate methods that perform tasks similar to mental information processing. Modern mathematical logic offers sophisticated and efficient algorithms. But its notions are too far from everyday language. There certainly still is demand for conceptual computation. John Sowa has proposed to use Conceptual Graphs as *"a system of logic based on the existential graphs of Charles Sanders Peirce and the semantic networks of artificial intelligence. They express meaning in a form that is logically precise, humanly readable, and computationally tractable"*. To work out the mathematical foundations that are necessary to make such computations work is, however, quite a task. It is advisable to look for ties to existing mathematical theories that could be utilized.

About a century ago, mathematicians became aware that computation is not restricted to numbers: computing with functions, matrices, polynomials, symmetries and the like had become a matter of course. A field of mathematical research devoted to studying the meta rules of computation was established in the 1930s under the name of Universal Algebra (Birkhoff [1], see also Grätzer [11]). It provides a vast repertoire of methods. A general algebraic structure in the sense of Universal Algebra is made up from arbitrary elements (replacing numbers as units of computation) and of operations (that take these elements as input and produce as output other elements, only depending on the input). In the setting of conceptual knowledge processing, the elements may be the concepts and the operations can be, as indicated above, common generalization and specialization, but possibly also others like negation or opposition. An *algebra of concepts* along these lines was worked out by Rudolf Wille and his group under the name of Formal Concept Analysis [9]. Based on mathematical Lattice Theory, it meanwhile has become a rather compact and stable theory itself. It can, as Wille [21] has demonstrated, be linked with Conceptual Graphs, thereby leading to a multitude of computational possibilities for these.

2 Formalization Precedes Computation

Do we need a mathematical theory in order to compute? Children can do without, can't they? Above, we have mentioned some natural operations for computations with concepts, such as common generalization and specialization. One could be tempted to use these operations right away and find out later, by doing, about the rules that govern such computations.

We predict that such an approach will not be successful. Consider as a simple example the sentence

The least common superconcept of cat and dog is pet.

This seems do be an instance of a conceptual operation as described earlier. Input are the concepts *cat* and *dog*, output is the concept *pet*. But such a statement cannot remain undisputed, and therefore it does not have the necessary objectivity. It is crucial for an algebraic operation that its results are independent of the computational process. As soon as there is the slightest doubt, computations loose their value.

We might argue that the cat-dog statement is *false* (because e.g. *carnivore* denotes a superconcept of *cat* and *dog*, but not of *pet*, and because tigers are cats, but no pets). That would even be too optimistic because it suggests that there might be a correct result. But this is not the case. There is no objective least superconcept of *cat* and *dog*, moreover, there is no objective interpretation of *cat* and *dog* either[1]. The meanings of these words depend on context, and so do conceptual operations.

This is a serious obstacle for computation. We are forced to consider context when computing, and, in particular, must precisely disclose the contextual influence in a manner that can be algorithmically encoded. An undispensable precondition for successful conceptual computation therefore is a suitable formalization of context and context-dependent concept operations.

One approach to do this is to use so-called *conceptual ontologies* and *type hierarchies*. A much more general and mathematically simpler way is gone in Formal Concept Analysis. There, a *formal context* is mathematically defined to be a triple (G, M, I) consisting of two sets G, M and a binary relation $I \subseteq G \times M$. The elements of G and M are usually interpreted as *objects* and *attributes*, with the relation I expressing the incidence between them, but note that the sets G and M may be arbitrary. All conceptual operations are performed with reference to such formal contexts. It may seem that such a data type is too simple, and in fact, more complex context representations are also used in Formal Concept Analysis. But it can be demonstrated that all these can be expressed in and represented by one or several such formal contexts (G, M, I), by means of a translation process called *scaling*.

[1] This seems to be just splitting hairs. A cat is a cat, isn't it? Let me reformulate: It is not obvious at all how to make the notion of *the set of all cats* sufficiently precise. For example, does it depend on time? We can easily agree on one definition, but the word "cat" by itself does not give a precise definition. In order to apply formal methods effectively, this level of precision however is indispensable.

It was again Rudolf Wille [21] who has demonstrated that this machinery can also be used to introduce a mathematical representation of conceptual graphs which connects these with the broad algebraic theory behind Formal Concept Analysis. This provides many possibilities for new algorithms. Such results will be presented elsewhere, here we shall give some impressions of the computational methods used by Formal Concept Analysis itself.

3 Formal Concepts

The basic mathematical theory of Formal Concept Analysis has been presented on many occasions, see [9] for a monography. Let us shortly recall that for any given formal context (G, M, I) we can define two operators that map subsets of G to subsets of M and conversely, for $A \subseteq G$, $B \subseteq M$ given by

$$A \mapsto \mathbf{int}(A) := \{m \in M \mid gIm \text{ for all } g \in A\},$$
$$B \mapsto \mathbf{ext}(B) := \{g \in G \mid gIm \text{ for all } m \in B\},$$

and that a *formal concept* of (G, M, I) is defined as a pair (A, B) of sets satisfying $A \subseteq G$, $B \subseteq M$, $\mathbf{int}(A) = B$, and $A = \mathbf{ext}(B)$. (We usually write A', B' instead of $\mathbf{int}(A)$, $\mathbf{ext}(B)$, but such a notation could be confusing in the program code below). The set $A = \mathbf{ext}\,\mathbf{int}(B)$ is called the *extent* of the concept (A, B), while $B = \mathbf{int}\,\mathbf{ext}(A)$ is the concept's *intent*. Ordered by the "subconcept–superconcept relation" $(A_1, B_1) \leq (A_2, B_2) : \Longleftrightarrow\ A_1 \subseteq A_2$, these formal concepts form a complete lattice, called the *concept lattice* of the formal context (G, M, I). This concept lattice is in fact a lattice in the sense of mathematical Lattice Theory (Birkhoff [1], see also Gr"atzer [10] for a recent monography). Moreover, every complete lattice is isomorphic to some concept lattice, which means that the mathematical methods of Formal Concept Analysis apply to each theory using lattices. The lattice operations *join* and *meet* (or, synonymously, *supremum* and *infimum*), can be interpreted on formal concepts as *least common superconcept* and *greatest common subconcept*.

Using lattices is not the only natural algebraization of conceptual operations. For example, Lehmann and Wille [16] have presented a family of algebraic structures, called *trilattices*, that can be used to describe the triadic approach by C.S. Peirce to concepts. Wille has shown that these structures are useful for the algebraic description of nested conceptual graphs. But trilattices are much less understood today than lattices.

Lattices are not only well investigated in their algebraic structure, they also admit a simple and appealing graphical representation (the *lattice diagram*). Interpreted as concept lattices, these diagrams have proven to support communication about conceptual data. The commercial TOSCANA-software[2] uses such diagrams for a better access to relational data bases.

Mentioning TOSCANA, would should point out that there are several extensive implementations of lattice algorithms, both related to Formal Concept Analysis (e.g., Burmeister [2], Vogt [20], Lindig [17]) and independent (e.g., Freese [6],

[2] `www.navicon.de`

Doignon and Falmagne [4]). Many of these programs can generate lattices from data, prepare or compute graphical representation including labelings, test for structural properties like distributivity and such. Others can translate between data structures. Several implementations are devoted to exploration procedures as described below.

4 Computing in a Concept Lattice

It is a common misconception that in order to perform lattice computations, one should first generate and store the lattice itself. There are simple and fast algorithms for generating a concept lattice from a given formal context, and we shall give such an algorithm below, but usually it is not necessary to generate all concepts. This is similar as with numeric computations: to perform addition or multiplication, it would be absurd to first generate all numbers that are potentially needed. It suffices to know that any number has a bit representation that describes its behaviour in algebraic operations. Similarly, a concept lattice that is used in practice may have billions of elements, or even be infinite, and yet every single algebraic operation may be performed without much effort. We shortly describe how this is done (in the finite case):

The usual computer representation of a set S is a bit vector, each bit representing an element of S. In fact, bit vectors of size $|S|$ then can be used to represent all subsets of S. A 1 in position s indicates that the element $s \in S$ belongs to the respective subset, a 0 indicates that it does not. The set operations $\cap$ and $\cup$ then correspond to the bitwise AND and OR, and thus are at least as simple and fast as the elementary arithmetical operations. We shall use them in the sequel without further mention. A formal context (G, M, I) can be respresented by a $G \times M$-bit matrix with an 1 in position (g, m) indicating that g and m stand in relation I. The gth row of this matrix, considered as a bit vector of size $|M|$, represents the set

$$g' := \{m \in M \mid gIm\},$$

and similarly is the mth column a bit vector representation for $m' := \{g \in G \mid gIm\}$. Formal concepts were introduced as pairs (A, B) of sets, with $A \subseteq G$ and $B \subseteq M$; they can thus be encoded as a pair of bit vectors of size $|G|$ and $|M|$, respectively. Writing **gcs** for the greatest common subconcept and **lcs** for the least common superconcept, we find that these are easy to compute:

$$\begin{aligned}\mathbf{gcs}\,((A_1, B_1), (A_2, B_2)) &= (A_1 \cap A_2, \mathbf{int}\,\mathbf{ext}(B_1 \cup B_2)),\\ \mathbf{lcs}\,((A_1, B_1), (A_2, B_2)) &= (\mathbf{ext}\,\mathbf{int}(A_1 \cup A_2), B_1 \cap B_2).\end{aligned}$$

The only non-obvious operations here are **int ext** and **ext int**. These are again easy. The program in Figure 1 computes, for a given set $T \subseteq M$, its closure $T_{ie} := \mathbf{int}\,\mathbf{ext}(T)$:

We have mentioned that there are simple algorithms to compute all concepts of a formal context. One that has first been implemented some twenty years

```
BEGIN
    T_ie := M;
    for  g ∈ G  do  if  T ⊆ g'  then  T_ie := T_ie ∩ g';
END
```

Fig. 1. Computing the closure $T'' := \mathbf{int}\,\mathbf{ext}(T)$ of an arbitrary set $T \subseteq M$. For computing $\mathbf{ext}\,\mathbf{int}(T)$, the rôles of G and M are interchanged.

ago is so simple that it can be reprinted here, see Figure 2 (a more detailed description is given in [9]). It relies on the fact that the operator $T \mapsto \mathbf{int}\,\mathbf{ext}(T)$ mathematically is a *closure operator*. The program computes, for a given closure operator $T \mapsto \textsc{Clo}(T)$ on a finite set M all *closures*, i.e., sets that are of the form $\textsc{Clo}(X)$ for some X.

```
procedure next_closure(var A : set; var found : boolean);
BEGIN
    X := A; found := false ; i := n;
    while (not found) and (i > 0) do begin
        if i ∉ X then begin
            A := Clo(X ∪ {i});
            found := (X ∩ {1,...,i − 1} = A ∩ {1,...,i − 1});
        end ;
        X := X \ {i}; i := pred (i);
    end ;
END
```

Fig. 2. Recursive algorithm to generate closed sets of a given closure operator Clo on the set $\{1, 2, \ldots, n\}$. To generate all closed sets, start with the closure of the empty set, $A := \textsc{Clo}(\emptyset)$, and repeatedly call the procedure next_closure as long as found=true .

The complexity of such algorithms has been studied[3], and it seems that this one still competes. But perhaps more interesting is its behaviour in practice. In a recent study, Lindig [17] has randomly generated formal contexts with 10,000–20,000 concepts. He found that his implementation of the algorithm needed about one second per thousand concepts; this is fast enough for most applications. Even the covering relation (see [8] for an algorithm) can be computed for lattices of this size in less than a minute. Since the algorithm is so simple, it can be modified in

[3] See Kuznetsov and Objedkov [15] for a recent comparison.

many ways with little effort: it can easily be made incremental, can be restricted to sublattices and other subsets of interest, can be used "under automorphisms", et cetera [9]. Kuznetsov [12], [13], [14] has shown that to determine the number of concepts of a given formal context is $\#\mathcal{P}$-complete. We should point out that there are many recent developments of new lattice generating algorithms, see e.g. Stumme et al. [19], and the literature cited there.

We have already mentioned that computing a full concept lattices is rarely necessary and often not desirable. A concept lattice of empirical data may be very large, may be incompletely known, may vary with changing views (scalings) of the data. Computing with formal concepts is easy using their bit representation. This leads to another key question: *Why* compute with concepts? What are the goals of conceptual computations? Again, a comparison with arithmetic is instructive: addition and multiplication of numbers are important because they enable us to solve tasks of practical value. To achieve solutions, certain mathematical techniques or compound operations are used, like solving a system of equations.

A list of ten prototypical *tasks of conceptual knowledge processing* was compiled by Wille [22]. They are

Exploring Searching Recognizing
Identifying Analyzing Investigating Deciding
Improving Restructuring Memorizing.

In Wille's paper, these ten tasks are described in detail and examples are given of how they can be attacked in conceptual structures. Other authors have suggested to use concept lattices for browsing, retrieval and for clustering, tasks that may be subsumed under those of Wille's list. Many applications are inspired by the analogy to *trees*: lattices can be used, like trees, as search- and decision structures, but with additional features. One of the disadvantages of decision trees, for example, is that the decisions must be made in a fixed order. A single missing information is likely to make the decision procedure unsuccessful, even if the available information would suffice to determine a result. Lattices are more flexible. In fact, one way to introduce concept lattices is as decision structures where the decisions can be made in arbitrary order. An elementary technique that supports these purposes allows to move about in a lattice, from elements to neighbouring elements. This compares to the **change directory**-command, used to move through tree-structured file systems, except that the branching in a lattice is both upwards and downwards: a concept may have several subconcepts, but also several (immediate) superconcepts. It is easy to find all immediate subconcepts of a given concept. In Figure 3 we present an algorithm that was mentioned in [8] (but has been implemented much earlier). When applied to the closure operator **int ext** of a formal context (G, M, I), it determines all (intents of) lower neighbours of a given concept (B', B). The same algorithm, applied to the **ext int**-operator on the set G, can determine the upper neighbours of any given concept. There is also a version that serves the same purpose as the latter, but operates on the set M, as the one in Figure 3.

```
BEGIN
    n_l := 0;
    for  m ∈ M \ B  do  MAXGEN[m] := true ;
    for  m ∈ M \ B  do
        begin
            C := CLO(B ∪ {m}) \ B;
            if  MAXGEN[n]= false for all n ∈ C \ {m}  then
                begin
                    n_l := n_l + 1;
                    LN[n_l] := C;
                end else  MAXGEN[m] := false ;
        end ;
END
```

Fig. 3. Program to compute those closures of a given closure operator CLO on the set M that cover a given closure B. n_l is the number and LN[1..n_l] is a list of all such neighbours.

The algorithms in Figures 2 and 3 are extraordinarily simple (yet useful). Most of the mentioned program implementations are, of course, much more advanced and cannot be described in detail here.

5 "Floating Point" Lattice Operations

One of the successful features of arithmetic is to allow a precise amount of imprecision, realized by truncating the decimal or binary representation of numbers. "Floating point" computations usually lead to results that differ only unessentially from the true values. They can, if necessary, be repeated with higher precision. Moreover, truncating often reflects the fact that the data are not given with higher precision. A similar data simplification is used for high dimensional numerical data: the analysis is carried out only for the "dimensions of highest charge".

The use of bit representations for formal concepts makes it easy to define operations of restricted precision, "floating point" operations, for concept lattices, too. In fact, since the **int ext**-operator is local in the sense that there is no bit carry-over, there is no natural choice of most significant bits. It is a matter of the intended interpretation purpose how such a coarsening is made. Wille [22] has suggested to understand and use conceptual structures such as concept lattices as *conceptual landscapes of knowledge*. Scouring about such landscapes can be done with lattice operations of restricted precision with varying choices of significant bits. The most advanced implementation of this idea is the already mentioned TOSCANA-software.

To give an idea of how such a coarsening may be useful in practice, let us allusively discuss a realistic scenario: Suppose we investigate data on the effects of a certain diet. The parameters given for each subject in the study may be numerical (age, height, initial and final weight, ...) and non-numerical (health status, sports activities, ...). For a rough conceptual analysis, the numerical values may be too precise and can be replaced by coarser qualitative ones (young, tall, overweight, ...). These new attributes must be defined in terms of the given parameters, but are not required to be mutually exclusive: They may have their own "conceptual ontology". With such a coarsening, we may discover regularities of the data ("healthy normalweight subjects did not lose much weight"). Then these observations can be made more and more precise by focussing and refining the qualitative attributes, based on the given data.

This is a general strategy, and it can be extended in many ways. It is, for example, possible to cover functional dependencies as well, see [9] for details. Notions of dimension can also be utilized for concept lattices, again in several ways. There is order-dimension and ordinal dimension, there are embeddings into direct products and tensor products, there are decompositions into overlapping "maps" like for an atlas. The mathematical and algorithmical preconditions for these notions are well understood (see [9]), the practical experiences are growing (see e.g. [17]).

6 Reliable Computation with Incomplete Data

For many applications it is unrealistic to assume that data are completely given. We have to make do with incomplete information. There are different approaches to this phenomenon, including the use of probabilistic methods and fuzzy set theory. It has been demonstrated (see e.g. Pollandt[18]) that these can be combined with concept lattice computations, but this will not be treated here. Instead, we discuss how to deal with incomplete but reliable data[4].

Assume that we study a data context (G, M, I) which is only very partially known. We concentrate on the attribute combinations that occur in (G, M, I). To get an idea of what is meant, think of a context where G is the set of all natural numbers, M is a set of properties like "prime", "sum of two squares", "sum of two cubes" (of natural numbers), that these numbers may or may not have, and where I is the natural incidence. A typical question that could be asked is

> *Is there a prime that is the sum of two cubes, but not the sum of two squares?*

In general, such an elementary query can be described by a pair (S, T) of sets S and T of attributes representing the question

> *Is there an object having all the attributes from S, but none of the attributes in T?*

[4] See also the article by Burmeister and Holzer in this volume.

From observations or from other sources we may learn for some of such pairs (S, T) whether the answer is yes or no, and thereby collect partial information about the data context (G, M, I).

Algorithmically, such information is difficult to handle. Even the most elementary question if the answer to a query (S, T) can be inferred from earlier observations turns out to be an $\mathcal{NP}$-complete problem. This becomes apparent when translating the setting into that of Propositional Calculus (with the set M as the set of propositional variables). Each query (S, T) then corresponds to propositional clause $\bigwedge S \to \bigvee T$, and the inference problem for such queries translates to the inference problem for clauses, a problem well known to be algorithmically intractable. Thus, reconstructing the combinatorial structure of a large formal context from elementary observations of this kind is a very time consuming task.

The computational conditions improve drastically if, instead of asking for the combinatorial structure, we explore the *conceptual* structure and ask for the possible concept intents rather than the attribute combinations of single objects (concept intents are attribute combinations in common to sets of objects). It turns out that then atomic queries of the form $(S, \{t\})$ suffice to determine the full structure. Such queries correspond to propositional formulae $\bigwedge S \to t$, known as propositional definite Horn formulae. In Formal Concept Analysis, *implications* are used instead, corresponding to more general propositional formulae of the form $\bigwedge S \to \bigwedge T$. Again, the complexity of the inference problem for such formulae is well known: it is essentially linear. Working with implications to explore the conceptual structure of a context is therefore much easier than the combinatorial approach.

Skillful use of the mathematical theory in fact allows to combine combinatorial and conceptual information without abandoning computational efficiency. To give an impression of how such more advanced methods are introduced, we shall take, in the last section of this article, a glimpse at a recent theoretical result. It is the starting point of a practical conceptual knowledge acquistion method.

This will, however, require a pinch of mathematical framework.

7 Cumulated Clauses

As was mentioned above, attribute combinations may be described using propositional *clauses*, i.e., expressions of the form $\bigwedge A \to \bigvee B$, where A and B are subsets of the attribute set M. We say that an arbitrary attribute set $F \subseteq M$ is a *model* of $\bigwedge A \to \bigvee B$ if and only if $A \not\subseteq F$ or $B \cap F \neq \emptyset$. F is a model of a set $\mathcal{C}$ of clauses if it is a model of each clause in $\mathcal{C}$. It is well known that clauses suffice to describe all set families. More precisely, for each set $\mathcal{F}$ of subsets of M there ist some set $\mathcal{C}$ of clauses such that the elements of $\mathcal{F}$ are precisely the models of $\mathcal{C}$.

A *cumulated clause* is an expression of the form

$$\bigwedge A \to \bigvee_{t \in T} \bigwedge B_t,$$

where T is some index set and A and each $B_t, t \in T$, are subsets of M. A set $F \subseteq M$ is a model of $\bigwedge A \to \bigvee_{t \in T} \bigwedge B_t$ if and only if $A \not\subseteq F$ or $B_t \subseteq F$ for some $t \in T$. Cumulated clauses generalize both ordinary clauses (where $|B_t| = 1$ for each $t \in T$) and implications (where $|T| = 1$). We use them because they have the expressiveness of clauses and share some algorithmic properties of implications. In particular, every set $\mathcal{F}$ of subsets of M can be described by cumulated clauses. For example we may use for each subset $X \subseteq M$ the cumulated clause α_X defined as

$$\alpha_X \equiv \quad \bigwedge X \to \bigvee \left\{ \bigwedge F \mid F \in \mathcal{F} \text{ minimal w.r.t. } X \subseteq F \right\}.$$

Then $\mathcal{C} := \{\alpha_X \mid X \subseteq M\}$ is a set of clauses that describes $\mathcal{F}$. Unfortunately, this set is of exponential size and therefore of little practical value.

An interesting result now is that for each family of sets there is a natural irredundant description with cumulated clauses. This is the key to a useful conceptual knowledge acquistion method. We present two recent mathematical results that make this method work.

First we need the notion of a pseudo model. The definition is elementary, but somewhat bewildering, because it is a recursive definition that seems to lack a base case. A closer look tells that this is not so.

Definition 1. *Let $\mathcal{F}$ be a set of subsets of the finite set M. A set $P \subseteq M$ is called a* pseudo model *of $\mathcal{F}$ if it satisfies the following two conditions:*

1. *$P \notin \mathcal{F}$ and*
2. *for each pseudo model Q of $\mathcal{F}$ which is a proper subset of P there is some $F \in \mathcal{F}$ with $Q \subseteq F \subseteq P$.*

Theorem 1. *Let M be a finite set and let $\mathcal{F}$ be a set of subsets of M. Then the* stem base

$$\{\alpha_P \mid P \text{ a pseudo model of } \mathcal{F}\}$$

is an irredundant set of cumulated clauses, the models of which are precisely the elements of $\mathcal{F}$.

A proof of this result can be found in [7]. It generalizes a result of Duquenne and Guigues [5] which applies to the case that $\mathcal{F}$ is closed under intersections. In that case, the stem base consists of implications and can be handled efficiently because of the linear inference algorithm for implications. This is frequently used for algorithmic purposes. For practical applications, the implicational approach is almost sufficient, except that one would like to include some non-implicational meta knowledge, mostly about the structure of many-valued attributes. But, as already said, non-implicational propositional knowledge cannot be efficiently handeled. But there is way around this: Using cumulated clauses we can show that mixing in a few non-implications does not ruin the efficiency of conceptual computation:

Theorem 2. *Let $\mathcal{N}$ be a set of cumulated clauses. The number of set operations necessary to decide if a given implication can be inferred from $\mathcal{N}$ together with a given set $\mathcal{H}$ of implications is (for fixed $\mathcal{N}$) linear in the size of $\mathcal{H}$.*

This theorem allows to extend the exploration techniques of Formal Concept Analysis to many valued contexts and, perhaps, to more complex conceptual structures that can be represented in context language. First implementations have already yielded promising results.

References

1. G. Birkhoff. *Lattice Theory.* American Math. Soc. Colloquium Publications 25. Providence, R.I., 3rd edition 1973.
2. P. Burmeister. ConImp*: Ein Programm zur Formalen Begriffsanalyse.* In: G.Stumme, R.Wille (Hrsg.). *Begriffliche Wissensverarbeitung: Methoden und Anwendungen.* Springer-Verlag, 1999.
3. Keith Devlin. *Goodbye Descartes. The end of logic and the search for a new cosmology of the mind.* John Wiley & Sons, New York 1997.
4. J.-P. Doignon, Falmagne. *Knowledge Spaces.* Springer-Verlag 1999.
5. J.-L. Guigues, V. Duquenne. *Familles minimales d'implications informatives resultant d'un tableau de données binaires.* Math. Sci. Humaines 95 (1986), 5-18.
6. R. Freese, J.Ježek, and J.B. Nation. *Free lattices.* Mathematical Surveys and Monographs, Vol. 42, American Mathematical Society, Providence, RI, 1995.
7. B. Ganter, R. Krau"se. *Pseudo models and propositional Horn inference.* Preprint, TU Dresden, 1999.
8. B. Ganter, S. Kuznetsov. *Stepwise construction of the Dedekind-MacNeille completion.* In: M.-L. Mugnier, M. Chein (eds.). *Conceptual structures: theory, tools, and applications.* LNAI 1453, Springer-Verlag 1998, 295–302.
9. B. Ganter, R. Wille. *Formal Concept Analysis.* Springer-Verlag 1998.
10. G. Gr"atzer. *General Lattice Theory.* Birkh"auser, Basel. 2nd edition 1998.
11. G. Gr"atzer. *Universal Algebra.* Springer-Verlag, 2nd edition 1979.
12. S.O. Kuznetsov. *Mathematical aspects of concept analysis.* Journal of Mathematical Science, Series Contemporary Mathematics and its Applications 18, 1654-1698, 1996.
13. S.O. Kuznetsov. *Interpretation on graphs and algorithmic complexity characteristics of a search for specific patterns.* Automatic Documentation and Mathematical Linguistics, **23**, no. 1, 37-45, 1989.
14. S.O. Kuznetsov. *Some Counting and Decision Problems in Formal Concept Analysis.* Preprint of the Technische Universität Dresden, MATH-Al-14-1999, September 1999.
15. S.O. Kuznetsov, Sergei A. Objedkov. *Algorithms for the Construction of the Set of all Concepts and their line diagram.* To appear.
16. F. Lehmann, R. Wille. *A triadic approach to Formal Concept Analysis.* In: G. Ellis, R. Levinson, W. Rich, J.F. Sowa (eds.). *Conceptual structures: applications, implementation and theory.* LNAI 954, Springer-Verlag 1995, 32–43.
17. C. Lindig. *Algorithmen zur Begriffsanalyse und ihre Anwendung in Softwarebibliotheken.* Ph.D.-Thesis, University of Braunschweig, 1999.
18. S. Pollandt. *Datenanalyse mit Fuzzy-Begriffen.* In: G.Stumme, R.Wille (Hrsg.). *Begriffliche Wissensverarbeitung: Methoden und Anwendungen.* Springer-Verlag 1999.

19. G. Stumme, R. Taouil, Y. Bastide, N. Pasquier, L. Lakhal. *Fast Computation of Concept Lattices Using Data Mining Techniques.* To appear.
20. F. Vogt. *Formale Begriffsanalyse mit C++: Datenstrukturen und Algorithmen.* Springer-Verlag 1996.
21. R. Wille. *Conceptual graphs and formal concept analysis.* In: D. Lukose, H. Delugach, M. Keeler, L. Searle, J.F. Sowa (eds.): *Conceptual structures: Fulfilling Peirce's dream.* LNAI 1257. Springer-Verlag 1997, 290-303.
22. R. Wille. *Conceptual Landscapes of Knowledge: A Pragmatic Paradigm for Knowledge Processing.* In: G. Mineau, A. Falls (eds.): Proc. 2nd Int. Symposium on Knowledge Retrieval, Use and Storage for Efficiency, Vancouver 1997.

Symmetry and the Computation of Conceptual Structures

Robert Levinson

Department of Computer Science, University of California, Santa Cruz, CA 95064

Abstract. The discovery and exploitation of symmetry plays a major role in sciences such as crystallography, quantum theory, condensed-matter physics, thermodynamics, chemistry, biology and others. It then should not be surprising then, since Conceptual Structures are proposed as a universal knowledge representation scheme, that symmetry should play a role in their interpretation and their application. In this tutorial style paper, we illustrate the role of symmetry in Conceptual Structures and how algorithms may be constructed that exploit this symmetry in order to achieve computational efficiency.

1 Introduction

1.1 Background and Motivation

CGs have a great promise and many applications [2], natural language processing and understanding, knowledge acquisition, systems modeling, data mining to name a few . Development and research in this domain, however, has suffered from the belief that this promising technology will not scale, even on multi processor computers. Our research has given comfort to the skeptics. The first attempt (early 1996) at converting 100,000 English sentences to CGs and matching the CG database against a small set of queries took about 6.5 hours on a Sun Sparc Ultra enterprise 4000. Using a variety of techniques, including the progressive application of our associative database methods III through VI, we managed to reduce this to 19 seconds (of which 16 seconds is the overhead ontology loading and 3 seconds is the actual processing time) on the same Sparc station (early 1997). Our goal is to further speed the process up so that this technology can actually be useful on real world size problems.

In order to achieve the current level of efficiency, symmetry has been exploited in multiple ways. In this paper, we first outline the many ways symmetry arises in Conceptual Structures and their application and then we explain how such symmetry can be exploited in CG implementations. This paper is meant to increase deeper understanding of our previous published methods as opposed to presenting new methods.

1.2 Symmetry and Diversity

Our research group has found it to be a useful pedagogical and research tool to divide information into two classes: diversity and symmetry. Symmetry is

B. Ganter and G.W. Mineau (Eds.): ICCS 2000, LNAI 1867, pp. 496–509, 2000.
© Springer-Verlag Berlin Heidelberg 2000

normally defined as "invariance with respect to transformation". *In this paper we will give a much looser definition of symmetry as "shared information between objects or systems and/or their representations".* The relationship between the traditional definition and our definition can be seen if , for example, one considers the bit level representation of two objects: 1001, 0111 where each bit represents a boolean feature "hair color is green.", "is male", "is living", and "can fly", respectively. In this representation we see that both sample objects share the feature that they can fly and, when plotted in a four dimensional Cartesian coordinate space, would be the same magnitude in the fourth dimension Further, this correspondence would not change if their values were changed (transformed) along the other dimensions.

Our approach to symmetry may not be easy to accept or get used to at first, we hope that you, the reader, will let us humor you in the meantime. Also, some of what follows is speculative, but has proved valuable as a paradigm for developing, understanding and explaining our research and that of others. We argue that each of the following terms are virtually synonymous: symmetry, similarity, relationship, mutual information, redundancy, regularity, structure, constraint. In the case of mutual information, for example, knowing something about one information object tells us something about another as well.

The theme of this paper is consistent with and support for the model of "computing as information compression, alignment and search" proposed by Wolff. [16] We consider different issues, however. Specifically, we explore how symmetry (potential compression) can be exploited cost effectively in graph operations and also what is the relationship between symmetry and the representation of knowledge in conceptual structures.

1.3 Diversity Can Often Only Be Resolved Through Search

Diversity or "potential surprise" - alternatively defined as "that which is left over in a system after symmetry has been removed" corresponds exactly with the notion of complexity (amount of computation required) used in algorithmic complexity analysis in Computer Science. Formal attempts to further characterize the information complexity of Algorithms have been developed such as Kolmogorov complexity [15]. Our goal is not to replace this work but to provide a deeper intuitive understanding of the interplay between computational complexity and symmetry.

We can view most algorithm design as the problem of constructing minimal cost functions from inputs to outputs. Practically speaking, one never encounters a problem with "unbounded input size" and knowledge of symmetries can be used to reduce the size of the set of possibilities that needs to be explored. But once all such symmetry is exploited all that remains to be done in traditional AI systems is to "try stuff". As possibilities are explored, further knowledge of relationships (symmetry) may be invoked leading to further savings (through search space pruning). Thus, part of the art of search algorithm design is to explore things in an order (such as in binary search) that is expected to reduce the amount of required future computation (diversity).

2 Exploiting Symmetry or Redundancy

Note that we are not saying that "symmetry" by itself is useful. What we are saying is that the recognition of symmetry and subsequent exploitation of this knowledge can be useful when the cost of recognition and exploitation pays for the reduced complexity (diversity reduction) of future computations. In a sense, unrecognized or unexploited symmetry, is "false diversity".

As an example of exploiting symmetry, consider Peirce's published proof procedure for Existential Graphs Inference [11]: [1] To prove an arbitrarily complex graph T:

1. Start with an empty sheet.
2. Using various rules of inference (Peirce gave 5 "alpha" rules which are sufficient to do the job) add, delete and change graphs on the sheet.
3. The theorem T is proven if the graph T ever appears alone or alongside others on the sheet.

At an ICCS95 evening session we proposed and proved the soundness and completeness of the following alternative "proof by contradiction" method, similar to that used in resolution refutation [12]:

1. Start with an empty sheet.
2. Add the negated theorem to be proved, T, to the sheet. (Negation is done by drawing a circle around T.)
3. Using the alpha rules add, delete and change graphs on the sheet.
4. The theorem T is proven if an empty circle (standing for "false" or contradiction) ever appears on the sheet alone or alongside others.

Although these two proof procedures accomplish the same thing, we argue that the second is superior for the following reasons:

- Since we are moving from complexity (circled T, where T is arbitrarily complex) towards simplicity (empty circle), quite often, simplifying rules (deletion and erasure) are correct to apply.
- Now all search attempts have the same goal! By exploiting this redundancy a human or mechanical theorem prover can more easily learn and transfer experience from proof to proof.
- In particular, **any intermediate sheet** that occurs in a derivation of circle is a valid goal for future proofs, since all such sheets embody contradictions.

Here, we have exploited "symmetry across computations", exactly the motivation behind the traditional Computer Science techniques of caching, memoization, and dynamic programming.

[1] Existential graphs (EGs) are a precursor of Sowa's [14] conceptual structures. The following discussion does not require knowledge of the details of EGs.

3 Symmetry and Conceptual Structures

In conceptual structures themselves, symmetry arises at the node level (due to types), graph level (due to analogy and existential graph inference rules) and database level (due to graph matching and storage levels). We will discuss each of these occurrences in the following sections.

3.1 Concept Types and Type Hierarchy

In Conceptual Structures, multiple concept labels may replace concept nodes in a conceptual structure and multiple relation labels may replace relation nodes in a conceptual structure. Such valid substitutions are normally stored in the ontology or type hierarchy that is associated with a Conceptual Structures database. In particular, words Y more specific than a word X share all of X's constraints and may be substituted for the word X in an answer to a question. In fact, the word X may be viewed as an abstraction or name for its successors like Y. For example, when asked for an example of "long-tailed" animals, squirrels may be a valid answer. Thus, all squirrels are expected to be symmetric with respect to tail type. In our view, any two objects (represented as conceptual structures) or simple entries in a type hierarchy are symmetric with respect to the attribute or subgraphs they share.

3.2 Relations as Symmetry

Consider the statement, "X is the mother of Y". Then all pairs, for example (Hillary Clinton, Chelsea Clinton), and (Mrs. Brady, Marcia Brady), share symmetry as they are analogous with respect to satisfying the Mother-Of relation. It is this ability for relations to refer to equivalent pairs (or more than 2 - for non-binary relations) that gives them first class status in Conceptual Structures.

3.3 Classification and Symmetry

The above examples are meant to show the tight correspondence between classification and symmetry: *Two objects that are classified as being the same on one or more dimensions, will be treated analogously in decisions based on just those dimensions.* Here "dimensions" are meant to be any attributes or features of the represented objects and need not be orthogonal or mutually independent.

In a CG database, two objects represented by the same conceptual structure,will be treated identically - regardless of which facts have been left out. What is not represented can not be exploited! Thus, classification is a very powerful tool that can be properly applied for efficient reasoning (as in scientific generalization) but also misapplied as in stereotypes of people by the color of their skin, sex, religion or birthplace, etc.

3.4 Graph-Level Symmetry

Matching of graph-level structure occurs in a CG database during:

- Query processing: An answer to a query Q is a supergraph of Q where node matching may invoke the type hierarchy.
- Graph insertion: To insert a graph G into a CG hierarchy requires finding the immediate subgraphs and immediate supergraphs of G in the hierarchy.
- Graph deletion: To delete a graph we must find it in the database.
- Logical inference: In Peirce's Alpha rule of deiteration [11] we are allowed to find and remove graphs (in a nested CG structure) that are the same as an outer graph.
- Analogical Reasoning: Analogies occur when conceptual structures have isomorphic or similar relational structure and concept nodes are related analogously as well. Such analogical reasoning is one of the main benefits of the use of CGs compared to first order logic representations (such as PROLOG) that suppress or ignore node and relation topology.

4 Supporting Operations

The two main mathematical operations required to implement the graph matching operations above are *graph isomorphism* and *subgraph-isomorphism*. Informally, in subgraph isomorphism, one is looking for a mapping between nodes in one graph G1 and nodes in a second graph G2 such that every edge-adjacency in G1 has a corresponding edge-adjacency in G2. In CGs, this matching also allows for some label relaxation due to the type hierarchy. In graph-isomorphism, the mapping must go both ways. Subgraph-isomorphism is known to be NP-complete, even for the special class of "bipartite graphs" that CGs fall into. It is an open problem whether graph-isomorphism is in class Polynomial or not.

Hence, the required operations required in CG computation could be expensive in the worst case. But the good news is that with a little cleverness (symmetry exploitation) their cost can be dramatically reduced for the average practical case. Furthermore, these costs can be shared (via symmetry) across the multiple graph matching operations that are required in a CG database. Specifically, efficiency is improved by using the set of design guidelines outlined in the next section.

Note that one could also be interested in finding symmetries between nodes in a single graph G. This information could, possibly, be used to compact the representation of G and speedup further operations on G. In general, we do not exploit this type of symmetry in what follows. In fact we make use of the lack of symmetry in most natural language graphs to process them more efficiently since it is easy to distinguish their nodes without search. (i.e. most analogies between nodes fail immediately at the node label level or their immediate adjacencies).

4.1 Conceptual Graph Implementation Guidelines [9]

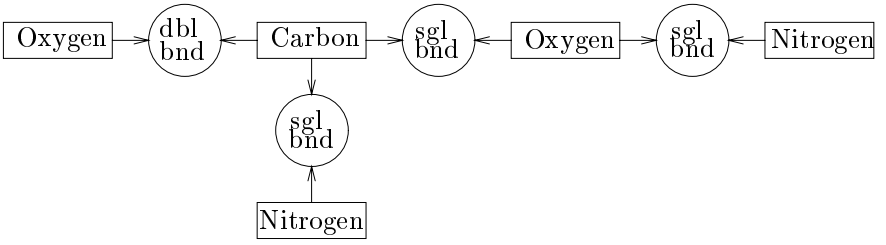
The following are proposed as useful guidelines not absolute principles.

– REDUNDANCY REMOVAL: Every primitive data object, label or symbol should be stored only once with pointers used to denote the actual uses of the object.
MINIMAL COMPOSITION:
– Every compound object should be stored with the minimum information required to represent the combination of its parts.
PROPER GRAIN SIZE:
– Given no loss of accuracy, objects should be processed at the highest level of abstraction possible.
A CONCEPTUAL GRAPH AS A SET OF TUPLES.
– If one were to implement a conceptual graph based on the diagrammatic representation, the costs associated with storage and matching would be much higher than they need to be. For example, the chemical shown in Figure 1, if represented as diagrammed, would require 9 nodes and 7 edges. Using the hypergraph-based tuple notation at the bottom of the figure we only need 4 nodes and 3 edges.
In fact, most natural language CGs can be rendered into tuple notation without using any edges. Figure 2 shows the *Display Form* of the Conceptual Graph which reads: *a monkey is eating a walnut with a spoon made out of the walnut's shell*. This CG has a cycle in it, so transcribing it into a *Linear Form* that can be typed into a computer requires some finessing. Some concept node needs to be picked as the *head*. Usually, the concept node with most arcs linked to it makes for the best choice for the head that produces the simplest CG. Picking [EAT] in our example for the head, yields the following *Linear Form*.

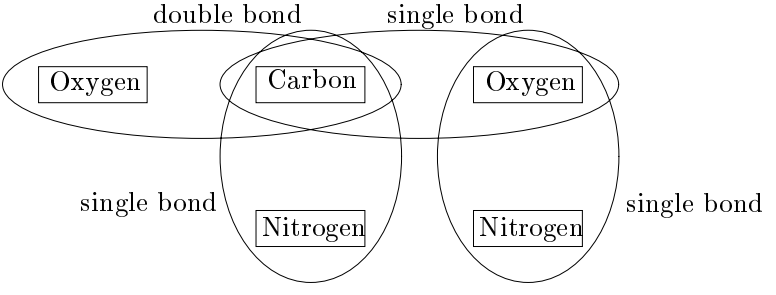
```
[EAT] -
  (AGNT) --> [MONKEY]
  (OBJ)  --> [WALNUT : *x]
  (INST) --> [SPOON] --> (MATR) --> [SHELL] <-- (PART)
                              <-- [WALNUT : *x].
```

Note the usage of the symbol $*x$. It is used as a *variable* to denote an unspecified individual of type [WALNUT]. Both instances must be the same one. So, it is a binding variable. In the *Tuple Notation*, we have developed, this binding variable is not used. In our convention, *a rose is a rose is a rose*. In this case, *a walnut is a walnut is a walnut*. All occurrences of a concept node are the same, unless differentiated. In the tuple notation, one walnut is differentiated from another with a number designator. The following CG would be read as: *a monkey is eating a walnut, with a spoon made from a shell of another walnut*:

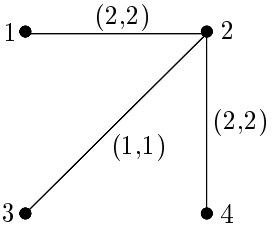
```
[EAT] -
  (AGNT) --> [MONKEY]
  (OBJ)  --> [WALNUT.1]
  (INST) --> [SPOON] --> (MATR) --> [SHELL] <-- (PART)
                              <-- [WALNUT.2].
```



CG Diagramatic Representation



CG Hypergraph Representation



Relation Table

1 double bond [Oxygen,Carbon]
2 single bond [Oxygen,Carbon]
3 single bond [Oxygen,Nitrogen]
4 single bond [Nitrogen,Carbon]

Relation-Based CG Representation

Fig. 1. CG Compaction

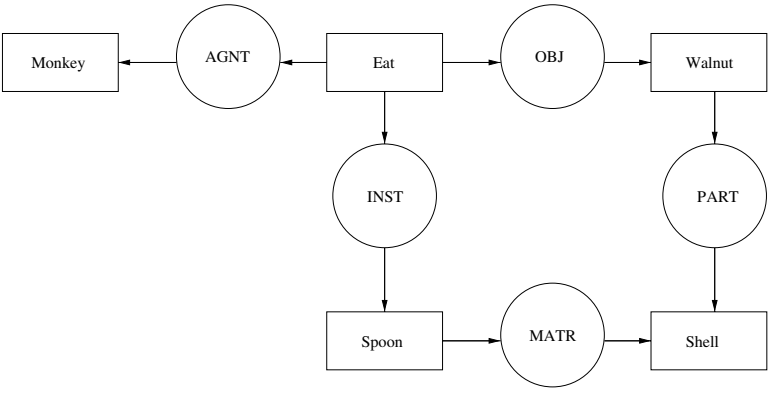


Fig. 2. A conceptual graph with a cycle

Alternatively, instead of the concept node [EAT], the concept node [SPOON] could be the head, which would produce the following *Linear Notation.*

```
[SPOON] -
  (INST) <-- [EAT] -
              (OBJ)  --> [WALNUT] --> (PART)
                --> [SHELL : *y].
              (AGNT) --> [MONKEY],
  (MATR) --> [SHELL : *y] .
```

In the *Tuple Based Notation*, which we developed, the above would be rendered, as follows :

```
@CG1 : {
   AGNT (EAT,    MONKEY),
   OBJ  (EAT,    WALNUT),
   INST (EAT,    SPOON),
   MATR (SPOON,  SHELL),
   PART (WALNUT, SHELL)
}.
```

It is as if each conceptual relation is simultaneously the *head* of the Conceptual Graph without taking favorites and making it less accessible for a searching agent. Any subset of the total CG, can be isolated, and used as a means for retrieval, or a join against another utterance. At the same time, no strange variables have been introduced. This is consistent with the modern view of language as sets (as opposed to sequences) of conceptual constructs.
RECURSIVE COMPOSITION:

– The same abstraction mechanism that goes from labels to graphs can be taken one step further to facilitate the storage and retrieval of nested graphs.

TOPOLOGICAL INFORMATION:

– A graph is itself the best descriptor of its nodes. A graph is the most compact representation of all the adjacencies and shortest paths between its nodes.

5 Six Organizational Methods

The design guidelines have been used to develop six database retrieval methods for Conceptual Graphs that have been discussed in previous papers by our group [7] and others. Each method exploits a larger amount of symmetry and hence (due to diversity reduction) achieves a deeper level of storage and retrieval efficiency. Complexity counts are rough estimates where each graph comparison is considered O(1).

- METHOD I: Arbitrary Flat Ordering. This is the naive system (used as a reference point to compare the other systems to) which involves no symmetry exploitation. Graphs are placed in a file in a random ordering. Space costs and retrieval costs are maximal.
- METHOD II: Two-level Ordering. Common features (subgraphs) of database graphs are used to create a set of "screens" that are used to prune the number of graphs that need to be accessed at the higher level. The addition of these screens increases storage costs over Method I but reduces retrieval costs by some constant factor K. Many fewer graph comparisons are required at the highlevel but all graphs must still be compared at the screen level.
- METHOD III: Multi-Level Partial Order. The full partial order relation "subgraph-of" is imposed on the database graphs and the relation is stored by storing its transitive kernel (transitive arcs have been removed) in a hierarchy. [6]. Further graphs may be added to the database to further balance and integrate the partial order. This is the main method currently in use in most semantic network and conceptual graphs implementations. [13,1] We see that storage costs have further increased, as we are still storing more pointers, but retrieval costs have dropped dramatically. Empirically such databases require log**2(N) graph comparison on a database with N graphs and provide access times that are respectable for many real-time applications. Gerard Ellis and Steven Barnes have provided implementation enhancements to the original Method III.
- METHOD IV: Hierarchical Node Descriptor Method. This method, developed with Gerard Ellis[8], exploits further symmetry between the graphs by discovering that neighborhoods of nodes of a given label (such as Carbon in chemistry) often are the same or similar as node neighborhoods in other graphs. Furthermore, these neighborhoods may be broken up into "descriptor units" which involve two nodes and their path distances. Descriptor units are shared across many node descriptors. Thus, storing graphs as sets of node descriptors, and node descriptors as sets of descriptor units provided substantial savings at the storage and graph matching levels.
  However, this method proved rather complicated to implement and in the 18 month process of doing so, Methods V and VI were discovered! Fortunately, by further exploiting symmetry and special characteristics of conceptual graphs, we are now able to get the power of Method IV but with a much simpler implementation.
- METHOD V:Shared (symmetric) Node Mappings.

Method V was described in one paragraph in the UDS Paper [9]. In Method III Phase I, a node X is not queried unless each of its predecessors has successfully matched as a subgraph (generalization) of the query graph. The idea of Method V is to exploit the work that has been done to find proper node bindings between the predecessor CGs and the query when finding proper node bindings between X and the query. In particular, each set of bindings must be consistent with (and hence only augments) all previous sets of bindings found.

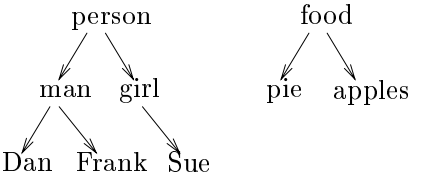
In order to bootstrap in this way, node binding information is added to all the predecessor links in the traditional Method III hierarchy. For each link $A->B$ we store (as a boolean matrix) for each node in A, which nodes in B it could possibly bind to in a correct subgraph-isomorphism match.

– METHOD VI: The Universal Data Structure.
Method VI is described in detail in the UDS paper. It exploits every design principle outlined above. By maximally exploiting symmetry both storage costs and retrieval times are reduced! In fact, it uses 1/100th of the storage of Method III while producing a 100th fold speedup. The reason it is called a Universal Data structure, is that we propose that it can be used as an implementation for any of the following traditional AI methods: conceptual graphs, semantic networks, relational databases, RETE rule matching, neural network learning and automatic theorem proving. [2]
In Method VI (see Figure 3), primitive node labels are stored (in a hash table) at the bottom of the hierarchy, tuples are stored as sequences of pointers to the node table, and graphs are stored as sets of pointers to the tuple table. Nested graphs point to the graph table, etc. Tuple tables essentially take the structure of relational databases. With just an elaborate set of pointers being stored, you might legitimately ask "but where is the beef?" the truth is that when on a diet high in symmetry, beef is not necessary: two structures (nodes, tuples, graphs) match if they have exactly the same pointers or, if a comparison is needed, their set of pointers is a subset of the other set (modulo the type and graph hierarchies). This definition of matching is sufficient for more than 99 percent of the graphs that we have seen arising from CG representation of natural language sentences. Difficult graphs such as those that occur in VLSI logic and organic chemistry will require further processing - due precisely to the fact that large amounts of apparent symmetry in the graph nodes (carbon atoms, for example, as in Figure 1) need to be distinguished based on higher level graph structures.

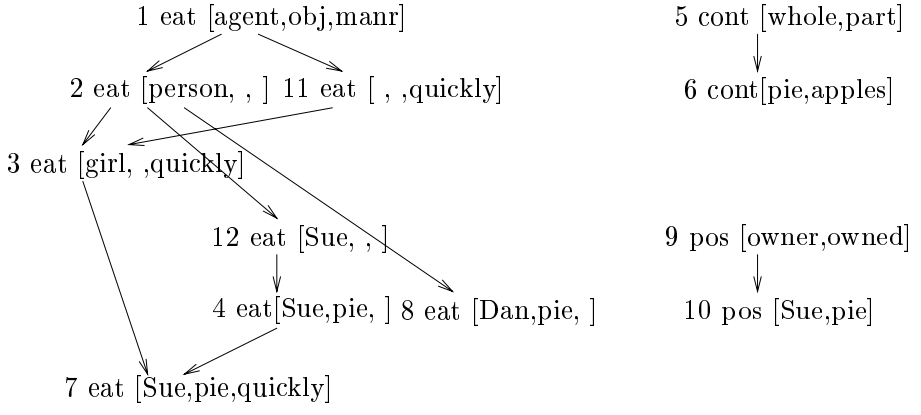
6 Further Enhancing the Methods: Parallelism

The idea of parallelism, that several computations may take place at the same time given adequate hardware, is yet another method of exploiting symmetry. By adding the redundancy of multiple processors speedup up on the order of the

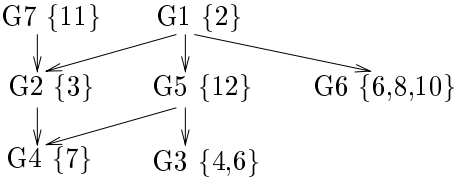
[2] Work on applying Method III hierarchies to automatic first order logic theorem proving has been done with John Esch [3].



Node Hierarchy



Relation Hierarchy



Graph Hierarchy

Fig. 3. Universal Data Representation

number of added processors is possible. Interestingly it is the "independence" (lack of symmetry or relationship) between the results and partial results of the computations that allows us to parallelize them.

James D. Roberts has designed parallel hardware and algorithms for enhancing CG computation. He recognized that many of the graph comparisons required by Methods III-VI can be scheduled independently of each other. He also

recognized that when comparing two graphs for subgraph-isomorphism using the technique of "refinement", that efficiency may be enhanced by parallelizing underlying boolean matrix operations. His considerations lead to an expected graph retrieval time of O(n**2/logN) using only O(logN**2/b) processors. [5]. As stated above Method III-VI systems require O(log(N)**2) graph comparisons.

7 The CG Mars Lander System

The CG Mars Lander[4] is a retrieval system designed for the purpose of answering factual questions about a body of documents. The results reported here are for a Method III style implementation - a Method VI style implementation is underway that will also integrate with Protege[10]. English discourse, coupled with ontology and queries are inputs to the system, and the result is an English answer with appropriate references to where in the English discourse the query gets answered.

7.1 Summary of Results and Timings

Our current program can accept a large set of CGs (tens of thousands), an ontology of some few hundred thousand words, and a set of queries. The program stores the CGs in a database which can be saved and restored, and answer queries by returning relevant CGs from the previously saved DB. The timing statistics on a Sun Ultra Enterprise 4000 (with 4 UltraSPARC 167Mhz and 512KB External Cache CPU and 256BM of main memory) are as follows: Read, process, and store an 18,000 CG input file in 1 hour and 46 minutes. Reloading of above DB takes on the order of seconds. A 150,000 word ontology is processed in 16 seconds. Each query is handled in 5.5 seconds. For smaller databases (hundreds of CGs only), the time to handle a single query can be as low as 0.2 seconds. A typical CG consists of some ten tuples, each of which has two to five arguments. Some large CGs can, however, reach up to 30 tuples (with no effective limit in the program).

7.2 Cost Benefit Analysis

Suppose you only plan to do a small number of queries Q, over a database of N CGs. The question arises whether it is worth the overhead of creating a Method III database, in which retrievals and insertions take approximately $\log_{10}^2(N)$ comparisons for a database of size N. The alternative is simply to compare each query to every CG, ignoring symmetry across queries.

Here we do the necessary math to aid in the decision: No pre-compilation of database: cost is $N * Q$ graph comparisons. (call this Method I)

Creating method III database: average cost per insertion is approximately $\log_{10}^2(\frac{N}{2})$, giving a cost to create the database of $N \log_{10}^2(\frac{N}{2})$ and a cost to answer

Table 1. Comparison table between costs of respective methods

N	Q	Method I Cost	Method III Cost
10	1	10	5.00
10	10	100	14.90
10	100	1,000	104.80
100	1	100	296.60
100	10	1,000	328.64
100	100	10,000	688.60
1,000	1	1,000	7,293.43
1,000	10	10,000	7,374.44
1,000	100	100,000	8,184.40
1,000	1,000	1,000,000	16,284.44
10,000	1,000	10,000,000	152,823.78
10,000	10,000	100,000,000	296,823.78

the queries of $Q \log_{10}^2(N)$. So, the question is when NQ is $< (N \log_{10}^2(\frac{N}{2}) + Q \log_{10}^2(N))$. Table 1 contains the results.

From this it can clearly be seen that Method III is most cost effective except for very small ratios of queries to database size, and for very high ratios the savings grow exponentially. Methods IV-VI provide further order of magnitude improvements over these numbers.

Acknowledgments

The work on the CG Mars Lander project was done in collaboration with the research staff of TextWise, Syracuse, New York. TextWise and Manning and Napier Information Systems provided some financial support for the project.

References

1. F. Baader, B. Hollunder, B. Nebel, H.-J. Profitlich, and E. Franconi, "An empirical analysis of optimization techniques for terminological representation systems," in *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning*, (Cambridge, MA), October 1992.
2. G. Ellis, R. A. Levinson, and P. Robinson, "Managing complex objects in PEIRCE," *Special Issue on Object-Oriented Approaches in Artificial Intelligence and Human-Computer Interaction (IJMMS)*, vol. 41, pp. 109–148, 1994.
3. J. Esch and R.A.Levinson, "Propogating truth and detecting contradictions in conceptual graphs databases," in *Conceptual Structures: Knowledge Representation as Interlingua* (P. Eklund, G. Ellis, and G. Mann, eds.), vol. 1527, pp. 229–247, Springer-Verlag, 1997.
4. G. Fuchs and R.A.Levinson, "The cg mars lander," in *Conceptual Structures: Fulfilling Peirce's Dream* (D. Lukose, H.S.Delegach, M. Keeler, L. Searle, and J. Sowa, eds.), vol. 1527, pp. 611–616, 1997.
5. R. Hughey, R. Levinson, and J. D. Roberts, "Issues in parallel hardware for graph retrieval," in *Proceedings of the 1st International Conference on Conceptual Structures*, pp. 62–71, 1993.

6. R. Levinson, "A self-organizing retrieval system for graphs," in *AAAI-84*, pp. 203–206, Morgan Kaufman, 1984.
7. R. Levinson, "Pattern associativity and the retrieval of semantic networks," *Computers and Mathematics with Applications*, vol. 23, no. 6-9, pp. 573–600, 1992. Part 2 of Special Issue on Semantic Networks in Artificial Intelligence, Fritz Lehmann, editor. Also reprinted on pages 573–600 of the book, Semantic Networks in Artificial Intelligence, Fritz Lehmann, editor, Pergammon Press, 1992.
8. R. Levinson and G. Ellis, "Multilevel hierarchical retrieval," *Knowledge-Based Systems*, vol. 5, pp. 233–244, September 1992. Special Issue on Conceptual Graphs.
9. R. A. Levinson, "Uds: A universal data structure," in *Proc. 2nd International Conference on Conceptual Structures*, (College Park, Maryland USA), pp. 230–250, 1991.
10. N. Noy and M. Musen, "An algorithm for merging and aligning ontologies," in *Proceedings of Workshop on Ontology Management*, vol. 1527, (Orlando, Florida), pp. 611–616, AAAI Press, 1999.
11. D. Roberts, "The existential graphs," in *Semantic Networks in Artificial Intelligence*, pp. 639–664, Roberts, 1992.
12. J. Robinson, "A machine oriented logic based on the resolution pronciple," *Journal of the ACM*, pp. 23–41, 1995.
13. J. B. Schmolze and T. A. Lipkis, "Classification in the KL-ONE knowledge representation system," in *IJCAI-83*, pp. 330–332, 1983.
14. J. F. Sowa, *Conceptual Structures.* Addison-Wesley, 1983.
15. P. Vitanyi and M. Li, *An Introduction to Kolmogorov Complexity and its Applications.* Springer-Verlag, 1993.
16. J. Wolff, *Towards a Theory of Cognition and Computing.* Chichester,UK: Ellis Horwood, 1991.

An Introduction to SNePS 3

Stuart C. Shapiro

Department of Computer Science and Engineering
and Center for Cognitive Science
State University of New York at Buffalo
226 Bell Hall
Buffalo, NY 14260-2000
shapiro@cse.buffalo.edu

Abstract. This paper provides an introduction to SNePS 3, the latest entry in the SNePS family of knowledge representation and reasoning (KRR) systems. The emphasis is on SNePS 3 as an example of a logic-based network KRR system.

1 Introduction

SNePS 3 is the latest entry in the SNePS family [1] of knowledge representation and reasoning (KRR) systems.[1] It is based on SNePS 2.5 [2] and ANALOG [3,4], and is currently being implemented in CLOS, the Common Lisp Object System [5]. This paper provides an introduction to SNePS 3 as an example of a logic-based network KRR system. In the rest of this paper, I will use the term "SNePS" when discussing features that are generally true of formalisms in the SNePS family, and "SNePS 3" when discussing features that distinguish SNePS 3 from other family members.

Due to space considerations, I will only be able to provide an informal and incomplete introduction to SNePS 3. Section 6 will be a more formal, but still incomplete, introduction to the syntax of the SNePS 3 language.

Informally, information is represented in SNePS as a network of nodes and labeled directed arcs. An arc label may be used on arbitrarily many different arcs, but each node has a unique identifier. For example, Fig. 1, based on a project in which Cassie, a SNePS-based agent [6], is embodied as a Foveal Extra-Vehicular Activity Helper-Retriever (FEVAHR) robot [7], represents the following information:[2]

> *Cassie is talking to and looking at Stu. Cassie is a FEVAHR. FEVAHRs are robots. Stu and Bill are people. People and robots are agents.*

How the network of Fig. 1 represents this information, *i.e.,* the syntax and semantics of SNePS networks, will be explained below.

[1] "SNePS" originally stood for "Semantic Network Processing System." Now, I would prefer it be thought of as a name in its own right.

[2] Fig. 1 shows only a small part of the FEVAHR knowledge base, chosen to illustrate the issues discussed in this paper.

B. Ganter and G.W. Mineau (Eds.): ICCS 2000, LNAI 1867, pp. 510–524, 2000.
© Springer-Verlag Berlin Heidelberg 2000

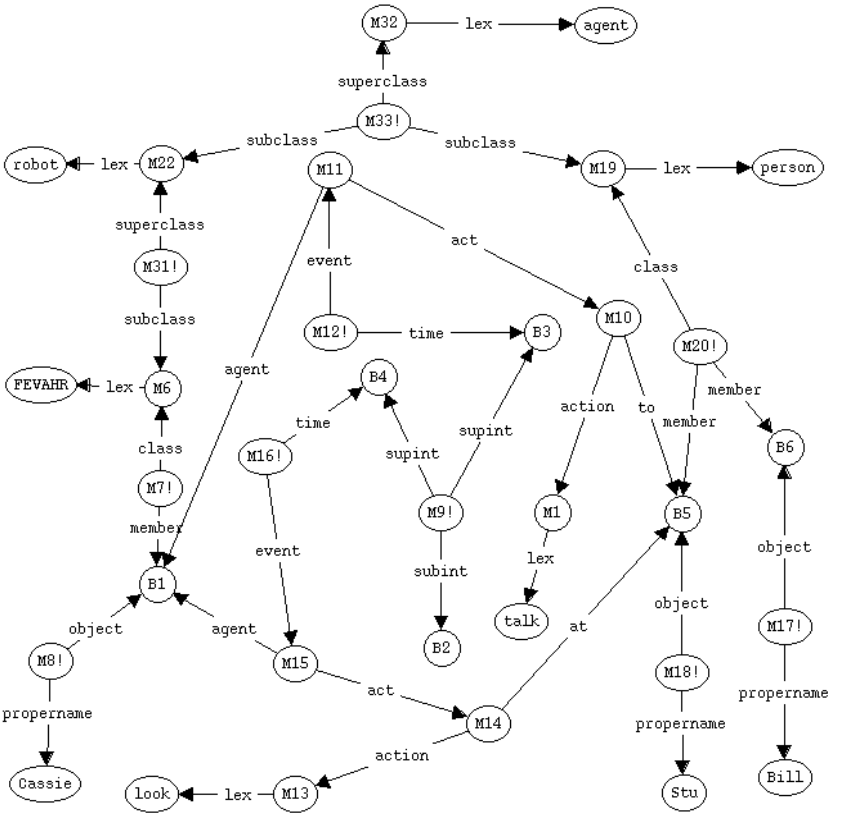


Fig. 1. A SNePS network representing the information that: Cassie is talking to Stu (`M12!`); Cassie is looking at Stu (`M16!`); Cassie is a FEVAHR (`M7!`); FEVAHRs are robots (`M31!`); Stu and Bill are people (`M20!`); people and robots are agents (`M33!`). Tables 1–4 give the intended semantics of each node.

2 KRR Systems as Logics

Every KRR system is a logic, in the sense that it has, or should have, a well-defined syntax, compositional semantics, and inference procedure. SNePS has been designed specifically as a logic to support natural language (NL) competent agents [3,4,6,8,9]. SNePS constitutes the "language of thought" of a SNePS-based agent. The domain of discourse of that language, the set of entities that are denoted by well-formed SNePS expressions, is the domain of all mental entities conceivable by the agent. Adding a new expression to a SNePS network implements the agent's conceiving of the entity represented by that expression.

The basic SNePS principles are

Propositional Semantic Network: The only well-formed SNePS expressions are nodes.
Term Logic: Every well-formed SNePS expression is a term.
Intensional Representation: Every SNePS term represents (denotes) an intensional (mental) entity.
Uniqueness Principle: No two SNePS terms denote the same entity.
Paraconsistent Logic: A contradiction does not imply anything whatsoever.

The significance of SNePS being a *propositional* semantic network is that only nodes are well-formed expressions having semantics, whereas in many other network-based KRR systems, including many semantic networks, arcs denote propositions—relations that are asserted to hold between entities. (See [10] for a discussion of assertional *vs.* structural information.)

The significance of SNePS being a term logic is that proposition-denoting terms may be arguments of other terms without leaving first-order logic [11].

The significance of SNePS using intensional representation is that cognitively distinct mental entities are denoted by distinct terms even if they are co-extensional—the entire network forms an opaque context in which there is no substitution of equals, because no two terms are fully equal [12,13,14,15]. This principle can also be read as "*Every* SNePS term denotes a mental entity," which means that no terms are created for purely technical reasons, and that even the analogue of logical variables, "variable nodes", have consistent compositional semantics denoting mental entities. Variable nodes did not have such semantics in earlier versions of SNePS [16]. Supplying that was a main motivator of ANALOG [3,4] and SNePS 3.

The Uniqueness Principle supports intensional representation, and imposes a requirement of structure-sharing on the implementation—no two distinct but structurally equal terms exist in a network.

SNePS being a paraconsistent logic means that a contradiction in one area of the knowledge base would not "corrupt" the information in another area of the knowledge base.

3 Levels

To a large extent, SNePS is a logical-level semantic network [17]. Just as a Prolog programmer must choose the predicates to be used in a Prolog program, a SNePS user must choose the arc labels and other representational techniques to use in a SNePS-based application. This person is called a "knowledge engineer", and the job of a knowledge engineer has been referred to as "conceptualization" [18, p. 9]. For example, Figures 2 and 3 show two possible ways of representing *Cassie is a FEVAHR*, and another is shown in Fig. 1.

However, just as the syntax, semantics, and inference mechanism of Horn clauses is built into Prolog, there is a level at which syntax, semantics, and an inference mechanism is built into SNePS. These will be discussed below.

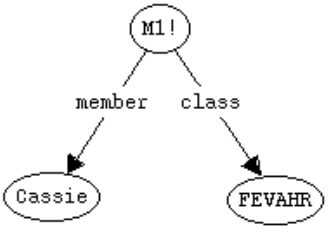


Fig. 2. A possible representation of *Cassie is a FEVAHR.*

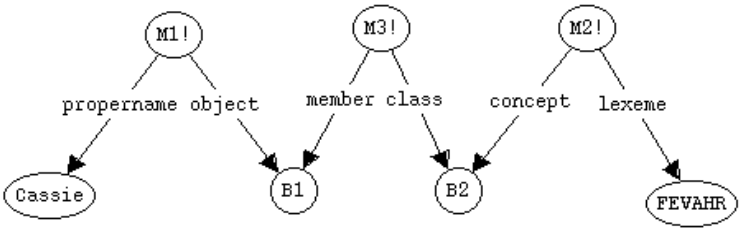


Fig. 3. Another possible representation of *Cassie is a FEVAHR.*

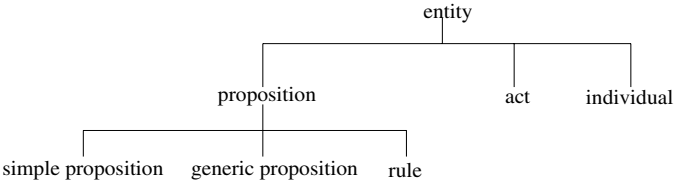


Fig. 4. The initial taxonomy of SNePS semantic classes.

4 Semantics

Every SNePS 3 node must be given a semantic class recognized by the system. The initial semantic taxonomy, which may be extended by the knowledge engineer, is shown in Fig 4.

Proposition nodes have an assertional status that is recognized by the inference mechanism (see Sect. 5 below). Rule nodes are proposition nodes that can be used for node-based inference. An example is shown in Fig. 5. Generic proposition nodes, such as `M89!` in Fig. 6, can be used for subsumption-based inference. Simple proposition nodes are proposition nodes that are neither generic propositions nor rules. Act nodes may be performed by the SNeRE acting executive [19,20], [2, Chap. 4].[3] Individual nodes are nodes that are neither proposition nodes nor act nodes.

[3] Due to space limitations, act nodes and the acting executive will not be discussed in this paper.

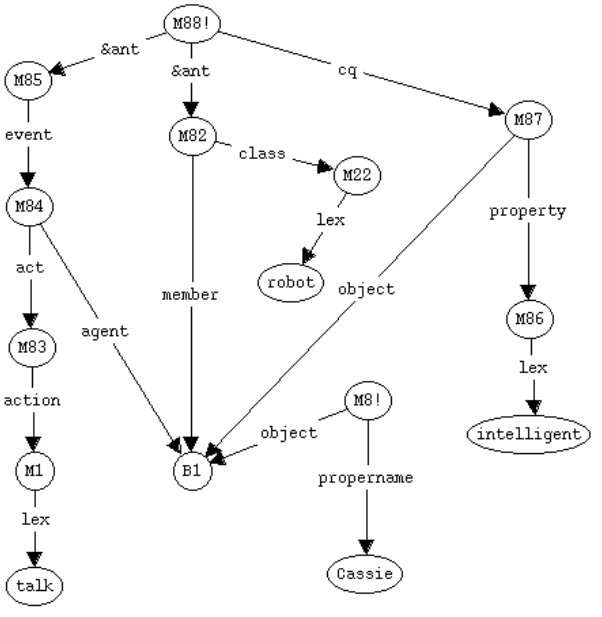


Fig. 5. M88! is a rule node denoting the proposition, *If Cassie is a robot that talks, then Cassie is intelligent.*

The SNePS 3 node classes and the categories of entities represented by those nodes as expressed in the SNePS formalism may be redundant, but this is because we, as SNePS 3 designers, did not want to specify *a priori* how the knowledge engineers could choose to represent categories. For example, node M6 of Fig. 1, node FEVAHR of Fig. 2, and node B2 of Fig. 3 might each represent the category of FEVAHR robots in different conceptualizations. When necessary to distinguish these two levels of semantics, I shall use the terms "SNePS semantics" and "domain semantics."

The intended domain semantics (intended by me, the knowledge engineer of this project) of the nodes in Fig. 1 are shown in Tables 1–4.[4] There, $[\![n]\!]$ is used to refer to the denotation of the SNePS term n. Lexemes, agents, times, categories, actions, acts, events, and propositions are all assumed to be entities in the domain of discourse, which, in this case, is the universe of Cassie's mental entities. Act-denoting terms are nodes in the SNePS act class, proposition-denoting terms are nodes in the SNePS proposition class, and all other terms are nodes in the SNePS individual class.

[4] Table 2 gives the semantics of B2 as "an interval of time." Cassie/FEVAHR contains a variable, NOW, whose value is the term denoting the current time, and which changes as time moves [21]. At the time of the snapshot of Fig. 1, B2 is the value of NOW, which is why Cassie's talking and looking is expressed in the present tense.

Table 1. The semantics of lexeme-denoting terms from Fig. 1

`agent`:	The English lexeme "agent".
`Bill`:	The English lexeme "Bill".
`Cassie`:	The English lexeme "Cassie".
`FEVAHR`:	The English lexeme "FEVAHR".
`look`:	The English lexeme "look".
`person`:	The English lexeme "person".
`robot`:	The English lexeme "robot".
`Stu`:	The English lexeme "Stu".
`talk`:	The English lexeme "talk".

Table 2. The semantics of agent-, time-, and category-denoting terms from Fig. 1

`B1`: Cassie.	`B2`: An interval of time.	`M6`: The category of FEVAHRs.
`B5`: Stu.	`B3`: An interval of time.	`M19`: The category of people.
`B6`: Bill.	`B4`: An interval of time.	`M22`: The category of robots.
		`M32`: The category of agents.

Table 3. The semantics of action-, act-, and event-denoting terms from Fig. 1

`M1`:	The action of talking.
`M13`:	The action of looking.
`M10`:	The act of talking to Stu.
`M14`:	The act of looking at Stu.
`M11`:	The event of Cassie's talking to Stu.
`M15`:	The event of Cassie's looking at Stu.

Table 4. The semantics of proposition-denoting terms from Fig. 1

`M7!`:	The proposition that Cassie is a FEVAHR.
`M8!`:	The proposition that Cassie's name is "Cassie".
`M9!`:	The proposition that ⟦`B2`⟧ is a subinterval of ⟦`B3`⟧ and ⟦`B4`⟧.
`M12!`:	The proposition that Cassie is talking to Stu throughout ⟦`B3`⟧.
`M16!`:	The proposition that Cassie is looking at Stu throughout ⟦`B4`⟧.
`M17!`:	The proposition that Bill's name is "Bill".
`M18!`:	The proposition that Stu's name is "Stu".
`M20!`:	The proposition that Stu and Bill are people.
`M31!`:	The proposition that FEVAHRs are robots.
`M33!`:	The proposition that robots and people are agents.

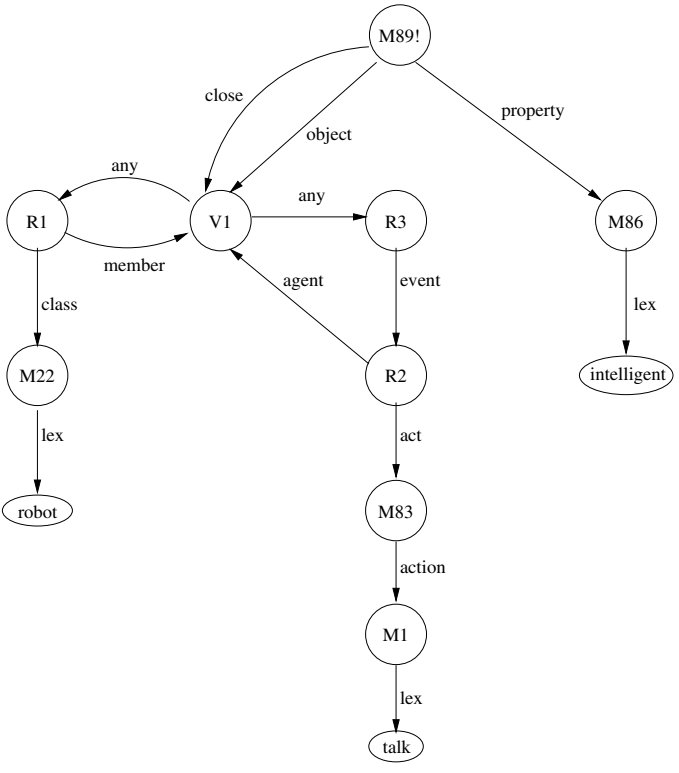


Fig. 6. `M89!` is a generic proposition node denoting the proposition that *Any robot that talks, is intelligent.*

5 Inference Methods

A SNePS-based agent might believe only some of the propositions represented in the network. For example, Cassie might believe that Bill believes that Stu is tall, without herself believing it. We notate believed propositions by appending an exclamation mark (`!`) to the identifiers of nodes that denote believed propositions, and we refer to such nodes as *asserted*. All the proposition nodes of Figures 1–3 are asserted. In Fig. 5, nodes `M82`, `M85`, and `M87` are unasserted proposition nodes, and node `M88!` is an asserted proposition node.

Inference in SNePS is a method of computing newly asserted nodes from previously asserted nodes. Thus, inference causes the agent to have new beliefs. The newly asserted nodes might previously have been in the network, denoting then unbelieved propositions, or they might be created by the inference engine.

There are four inference methods in SNePS, the first two of which distinguish SNePS from logic-based but non-network-based KRR systems. Briefly, and informally, these are:

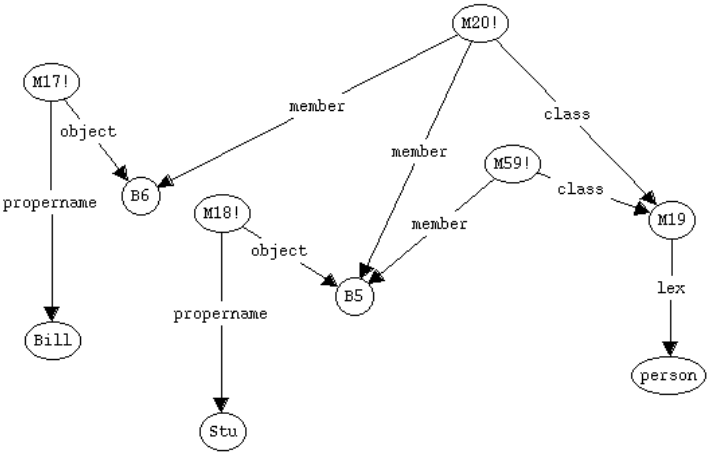


Fig. 7. Node `M20!`, denoting the proposition that Stu and Bill are people, implies `M59!`, denoting the proposition that Stu is a person, by wire-based inference.

Wire-based inference, referred to as "reduction inference" in previous versions of SNePS (notably in [22]), is an inference method whereby a proposition node *m1* implies a proposition node *m2* with a subset or a superset of *m1*'s arcs. (See Secs. 6.1 and 6.2.) For example, node `M20!` in Fig. 7, repeated from Fig. 1, implies node `M59!` by wire-based inference.

Path-based inference [22,23,24] is an inference method whereby a path of arcs from a proposition node *m1* to a node *m2* (of any class) may imply a proposition node *m3* with all of *m1*'s arcs plus an additional one to *m2*. Path-based inference must be sanctioned by path-based inference rules that are given to the SNePS system, but not represented in the SNePS object language. (See [2, Sect. 2.5.2] for the syntax and semantics of paths.) One possible path-based inference rule is

```
(define-path class
   (compose class
            (kstar (compose subclass- ! superclass))))
```

Given this rule, node `M60!` of Fig. 8, denoting the proposition that Stu is an agent, may be inferred from `M59!`, denoting the proposition that Stu is a person, and the path of arcs from `M59!` to `M32` by path-based inference followed by wire-based inference so that there is only one `class` arc emanating from `M60!`.

Node-based inference uses nodes that are the analogue of non-atomic formulas in first-order predicate logic to represent domain rules that SNePS can use to infer newly asserted proposition nodes from previ-

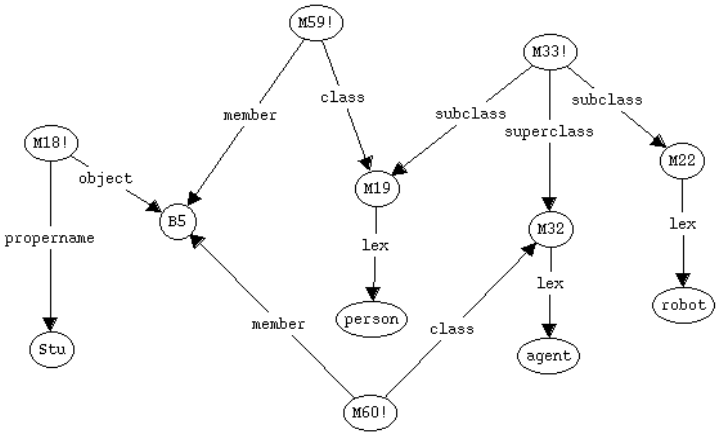


Fig. 8. Node `M60!`, denoting the proposition that Stu is an agent, may be inferred from `M59!`, denoting the proposition that Stu is a person, and the path of arcs from `M59!` to `M32` by path-based inference followed by wire-based inference.

ously asserted proposition nodes. For example, in Fig. 5, node `M88!` is an "and-entailment"—a kind of entailment with `M82` and `M85` as conjoined antecedents and `M87` as a consequent. `M88!` denotes the domain rule, *If Cassie is a robot that talks, then Cassie is intelligent.* Since `M82` follows from Fig. 1's `M7!` by path-based inference, and `M85` follows from Fig. 1's `M12!` by wire-based inference, `M87` may be inferred by node-based inference.

Subsumption inference is an inference method in SNePS 3 that is not available in earlier versions of SNePS, and uses the ANALOG [3,4] technique of representing variables. For example, Fig. 6 shows a SNePS 3 representation of the generic proposition, *Any robot that talks, is intelligent.* `V1` is a structured variable that denotes the arbitrary talking robot [25], `R1` and `R3` are restrictions on `V1`, and `M89!` denotes the asserted generic proposition. Since `R1` subsumes Fig. 5's `M82` and `R3` subsumes Fig. 5's `M85` and those nodes follow from Fig. 1 as explained above, therefore `V1` subsumes Fig. 1's (and Fig. 5's) `B1`, `M89!` subsumes Fig. 5's `M87`, and `M87` follows from `M89!` by subsumption inference.

6 Syntax

The syntax of the SNePS 3 language is defined in terms of *nodes*, *relations*, *wires*, *cables*, and *cablesets* (see also [3,4,22]). Informally, a node is what I have been calling a node heretofore, and a relation is an arc label. A wire is a pair, $\langle r, n \rangle$,

of a relation r and a node n. A cable is a pair, $\langle r, ns\rangle$ of a relation r and a set of nodes (or "nodeset") ns. A cableset is a set of cables, $\{\langle r_1, ns_1\rangle, \ldots, \langle r_k, ns_k\rangle\}$, such that no two r_i are the same. A set of wires all of which have the same relation forms a cable. A set of wires with different relations forms a cableset. Nodes that I have been informally presenting as labeled circles or ovals with arcs emanating from them formally *are* cablesets.

With the previous paragraph as introductory, and the foregoing sections as motivational, I can now give a more formal presentation of SNePS 3 syntax.

6.1 Relations

A SNePS 3 *relation* is a four-tuple, $\langle name, type, adjust, limit\rangle$, where the four constituents are:

***name*:** A symbolic name of the relation. No two relations may have the same name. Relation names were used in Figures 1–8 as labels on directed arcs.

***type*:** The SNePS semantic class of the nodes in the range of this relation, *i.e.,* of the nodes pointed to by arcs labeled with this relation.

***adjust*:** Either `expand`, `reduce`, or `none`. This specifies how wire-based inference treats instances of this relation. For example, Fig. 7 illustrates the reducibility of the relation named `member`. However, the relation named `&ant`, shown in Fig. 5 is expandable. A value of `none` means that this relation is not subject to wire-based inference.

***limit*:** The minimal size of a nodeset that can be paired with this relation in a cable. If *adjust* is `reduce`, then *limit* is the minimal allowed reduction. For example, to prevent Fig. 7's node `M20!` from implying a node with only a `class` arc and no `member` arcs, the *limit* of the relation named `member` is 1. If *adjust* is `expand`, then *limit* is the minimal nodeset-size of a cable that allows more wires to be added. For example, it does not make sense to add `&ant` arcs to a node that is not a kind of entailment.

Assuming that the knowledge engineer has declared the classes `category`, `event`, and `time` to be subclasses of the SNePS class of individual nodes, some example relations would be:

$\langle$`member`, `entity`, `reduce`, `1`$\rangle$ $\langle$`class`, `category`, `reduce`, `1`$\rangle$
$\langle$`event`, `event`, `reduce`, `1`$\rangle$ $\langle$`time`, `time`, `reduce`, `0`$\rangle$
$\langle$`&ant`, `proposition`, `expand`, `1`$\rangle$ $\langle$`cq`, `proposition`, `reduce`, `1`$\rangle$

6.2 Case Frames

A *case frame* is a pair, $\langle c, \rho\rangle$, where c is a SNePS semantic class and ρ is set of relations. Informally, c is the class of all nodes that have the given set of arcs emanating from them. A more formal explanation is given below.

Case frames are partially ordered by the relation *adjustable* ($\sqsupseteq_{adj}$). Case frame $\langle c_1, \rho_1\rangle$ is adjustable to case frame $\langle c_2, \rho_2\rangle$ ($\langle c_1, \rho_1\rangle \sqsupseteq_{adj} \langle c_2, \rho_2\rangle$) iff

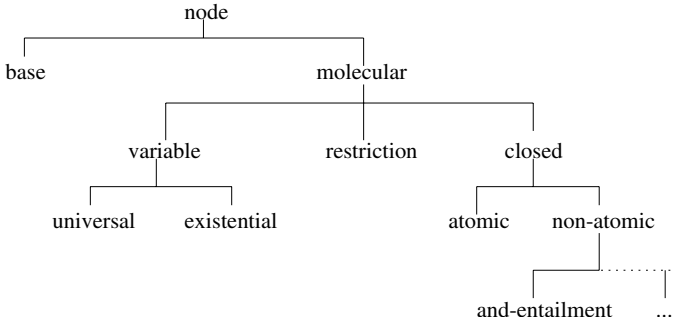


Fig. 9. Part of the hierarchy of the syntactic classes of SNePS nodes.

1. $c_1 = c_2$;
2. every relation in $\rho_1 - \rho_2$ is reducible and has *limit*=0;
3. every relation in $\rho_2 - \rho_1$ is expandable and has *limit*=0.

For example, the fact that Fig. 5's `M85` follows from Fig. 1's `M12!` by wire-based inference is partially justified by the fact that
$\langle$`proposition`, $\{\langle$`event`, `event`, `reduce`, `1`$\rangle$, $\langle$`time`, `time`, `reduce`, `0`$\rangle\}\rangle$
$\sqsupseteq_{adj}$
$\langle$`proposition`, $\{\langle$`event`, `event`, `reduce`, `1`$\rangle\}\rangle$

6.3 Nodes

Before entering nodes into a SNePS 3 network, the knowledge engineer must declare the case frames to be used. The system will check that relations are being used consistently. The knowledge engineer may also add additional semantic classes to the SNePS 3 class hierarchy, and use these new classes in the case frames.

A valid SNePS 3 node must be in one of the syntactic classes in the SNePS 3 syntactic hierarchy, part of which is shown in Fig. 9.

A base node consists only of an identifier. A base node identifier may be created by a SNePS 3 user. Some examples of these in Fig. 1 are `Cassie`, `robot`, and `look`. Other base nodes may be created and named by the system. Examples of these in Fig. 1 are `B1` and `B2`. When a base node is created, it must be given a SNePS 3 semantic class.

An atomic closed molecular node is a cableset $\{\langle r_1, ns_1\rangle, \ldots, \langle r_k, ns_k\rangle\}$ such that:

1. the set $\{r_1, \ldots r_k\}$ is the relation set of some declared case frame $\langle c, \rho\rangle$;
2. for each i, each node in ns_i is of the semantic class which is the *type* of r_i;
3. for each i, the cardinality of ns_i is at least the *limit* of r_i.

If a cableset satisfies these three conditions, it is considered to be an *instance* of the case frame $\langle c, \rho\rangle$, and is entered into the network as a node in the semantic class c. Closed molecular nodes are given identifiers of the form `M`n by the system.

We can refer to a cableset either by using the cableset notation, using just its identifier, or using both. For example, node `M20!` of Figures 1 and 7 may be referred to as $\texttt{M20!} : \{\langle \texttt{member}, \{\texttt{B5}, \texttt{B6}\}\rangle, \langle \texttt{class}, \{\texttt{M19} : \{\langle \texttt{lex}, \{\texttt{person}\}\rangle\}\}\rangle\}$.

SNePS 3 knowledge engineers may create any case frames they desire to form atomic closed molecular nodes. The syntax of non-atomic, variable, and restriction nodes, however, are fixed.

Non-atomic molecular nodes are used to represent rules used for node-based inference. For example, node `M88!` of Figure 5 is an and-entailment, the case frame for which is

$$\langle \texttt{rule}, \{\langle \&\texttt{ant}, \texttt{proposition}, \texttt{expand}, \texttt{1}\rangle, \langle \texttt{cq}, \texttt{proposition}, \texttt{reduce}, \texttt{1}\rangle\}\rangle$$

See [2, Chap. 3] for the syntax of the other non-atomic nodes.

A universal variable node is an instance of one of the case-frame schemata

$$\langle c, \{\langle \texttt{any}, \textit{restriction}, \texttt{expand}, \texttt{1}\rangle\}\rangle$$

and an existential variable node is an instance of one of the case frame schemata

$$\langle c, \{\langle \texttt{some}, \textit{restriction}, \texttt{none}, \texttt{1}\rangle, \langle \texttt{depends}, \textit{universal}, \texttt{none}, \texttt{0}\rangle\}\rangle$$

where c is any semantic class, *restriction* is any restriction node, and *universal* is any universal variable node. Variable nodes are given identifiers of the form `Vn` by the system.

A restriction node is an instance of any case frame declared by the knowledge engineer, except that every restriction node must dominate at least one variable node. Thus, variable nodes and restriction nodes form cycles in the SNePS 3 network. Two such cycles are illustrated in Fig. 6: `V1-R1-V1` and `V1-R3-R2-V1`. Restriction nodes are given identifiers of the form `Rn` by the system.

One more relation is needed for the SNePS 3 syntax of generic propositions: $\langle \texttt{close}, \textit{universal}, \texttt{none}, \texttt{0}\rangle$. Nodes from which `close` arcs emanate are taken to form the scope of universal variable nodes that the `close` arcs point to, as well as all existential variable nodes dependent on them. An example of the need for this is to distinguish *Any talking robot is not intelligent.* from *It is not the case that any talking robot is intelligent.* A `close` arc is shown in Fig. 6.

7 Summary

SNePS 3, the latest version of the SNePS family of propositional semantic networks, is a logic- and network- based knowledge representation, reasoning, and acting system, based on a paraconsistent, first-order term logic, with compositional intensional semantics.

SNePS 3 differs from earlier versions of SNePS by having the following features:

1. formal SNePS semantic classes of nodes
2. formal definition of relations

3. formal definition of case frames
4. structured variables
5. wire-based inference, including expansion as well as reduction, and limits on adjustment
6. subsumption inference

8 Availability and Use

SNePS 3 is currently being implemented. SNePS 2.5 has been implemented in ANSI Common Lisp and runs on any platform where that language is installed. SNePS 2.5 is useable for research and experimentation, and can currently handle knowledge bases on the order of about 1,000 SNePS nodes. It and other versions of SNePS are currently in use at various sites around the world, including the U.S., Portugal, Italy, and Japan.

The SNePS 2.5 source code and manual may be freely downloaded from the SNePS Research Group web pages at `http://www.cse.buffalo.edu/sneps/`, along with a tutorial, sample demonstration runs, and a bibliography.

9 Benchmarks and Comparisons

SNePS comes with a suite of demonstration problems and applications that can be used to familiarize oneself with how to use it, and may be used for comparison with other systems. Demonstrations that were taken from other sources include The Jobs Puzzle from [26, Chapter 3.2], Schubert's steamroller problem (see [27]), and a database management system example from [28].

Schubert's steamroller problem was run on a 1993 version of SNePS [29], and the results compared with those reported in [27]. The SNePS version produced fewer unifications and was faster than most unsorted logic solutions, but was outperformed by sorted logic solutions. The current version of SNePS is much faster on this problem than the 1993 version, partially due to improvements in SNePS, and partially due to faster, bigger computers.

The SNePS representation of the Jobs Puzzle is much simpler and closer to the English version of the puzzle than the clause form representation presented in [26, p. 58ff].

Acknowledgments

The design and implementation of SNePS 3 is a joint effort of the author and Syed S. Ali, Debra T. Burhans, Alistair E. Campbell, Alan M. Hunt, Haythem O. Ismail, Geoffrey D. Koplas, and other members of the SNePS Research Group, Department of Computer Science and Engineering, University at Buffalo. The author is grateful an anonymous reviewer, and to William J. Rapaport, Haythem O. Ismail, and other members of the SNePS Research Group for their comments on earlier versions of this paper.

References

1. Stuart C. Shapiro and William J. Rapaport. The SNePS family. *Computers & Mathematics with Applications*, 23(2–5):243–275, January–March 1992. Reprinted in [30, pp. 243–275].
2. Stuart C. Shapiro and The SNePS Implementation Group. *SNePS 2.5 User's Manual*. Department of Computer Science and Engineering, State University of New York at Buffalo, Buffalo, NY, 1999.
3. Syed S. Ali and Stuart C. Shapiro. Natural language processing using a propositional semantic network with structured variables. *Minds and Machines*, 3(4):421–451, November 1993.
4. Syed S. Ali. *A "Natural Logic" for Natural Language Processing and Knowledge Representation*. PhD thesis, Technical Report 94-01, Department of Computer Science, State University of New York at Buffalo, Buffalo, NY, January 1994.
5. Guy L. Steele, Jr. *Common Lisp the Language*. Digital Press, second edition, 1990.
6. Stuart C. Shapiro. The CASSIE projects: An approach to natural language competence. In J. P. Martins and E. M. Morgado, editors, *EPIA 89: 4th Portugese Conference on Artificial Intelligence Proceedings, Lecture Notes in Artificial Intelligence 390*, pages 362–380. Springer-Verlag, Berlin, 1989.
7. Stuart C. Shapiro. Embodied Cassie. In *Cognitive Robotics: Papers from the 1998 AAAI Fall Symposium, Technical Report FS-98-02*, pages 136–143. AAAI Press, Menlo Park, California, October 1998.
8. Stuart C. Shapiro and William J. Rapaport. Models and minds: Knowledge representation for natural-language competence. In Robert Cummins and John Pollock, editors, *Philosophy and AI: Essays at the Interface*, pages 215–259. MIT Press, Cambridge, MA, 1991.
9. Stuart C. Shapiro. SNePS: A logic for natural language understanding and commonsense reasoning. In Łucja Iwańska and Stuart C. Shapiro, editors, *Natural Language Processing and Knowledge Representation: Language for Knowledge and Knowledge for Language*. AAAI Press/The MIT Press, Menlo Park, CA, 2000.
10. William A. Woods. What's in a link. In Daniel G. Bobrow and Allan Collins, editors, *Representation and Understanding*, pages 35–82. Academic Press, New York, 1975. Reprinted in [31, pp. 218–241].
11. Stuart C. Shapiro. Belief spaces as sets of propositions. *Journal of Experimental and Theoretical Artificial Intelligence (JETAI)*, 5(2&3):225–235, April–September 1993.
12. Anthony S. Maida and Stuart C. Shapiro. Intensional concepts in propositional semantic networks. *Cognitive Science*, 6(4):291–330, October–December 1982. Reprinted in [31, pp. 170–189].
13. Stuart C. Shapiro. Symmetric relations, intensional individuals, and variable binding. *Proceedings of the IEEE*, 74(10):1354–1363, October 1986.
14. Stuart C. Shapiro and William J. Rapaport. SNePS considered as a fully intensional propositional semantic network. In Nick Cercone and Gordon McCalla, editors, *The Knowledge Frontier*, pages 263–315. Springer-Verlag, New York, 1987.
15. William J. Rapaport, Stuart C. Shapiro, and Janyce M. Wiebe. Quasi-indexicals and knowledge reports. *Cognitive Science*, 21(1):63–107, January–March 1997. Reprinted in [32, pp. 235–294].
16. David J. Israel. Interpreting network formalisms. In N. Cercone, editor, *Computational Linguistics*, pages 1–13. Pergammon Press, Oxford, 1983.

17. Ronald J. Brachman. On the epistemological status of semantic networks. In Nicholas V. Findler, editor, *Associative Networks: The Representation and Use of Knowledge by Computers*, pages 3–50. Academic Press, New York, 1979.
18. Michael R. Genesereth and Nils J. Nilsson. *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann, Los Altos, CA, 1987.
19. Deepak Kumar. The SNePS BDI architecture. *Decision Support Systems*, 16(1):3–19, January 1996.
20. Deepak Kumar and Stuart C. Shapiro. Acting in service of inference (and *vice versa*). In Douglas D. Dankel II, editor, *Proceedings of The Seventh Florida AI Research Symposium (FLAIRS 94)*, pages 207–211. The Florida AI Research Society, May 1994.
21. Haythem O. Ismail and Stuart C. Shapiro. Two problems with reasoning and acting in time. In Anthony G. Cohn, Fausto Giunchiglia, and Bart Selman, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Seventh International Conference (KR 2000)*, pages 355–365, San Francisco, 2000. Morgan Kaufmann.
22. Stuart C. Shapiro. Cables, paths and "subconscious" reasoning in propositional semantic networks. In John Sowa, editor, *Principles of Semantic Networks: Explorations in the Representation of Knowledge*, pages 137–156. Morgan Kaufmann, Los Altos, CA, 1991.
23. Stuart C. Shapiro. Path-based and node-based inference in semantic networks. In David L. Waltz, editor, *Tinlap-2: Theoretical Issues in Natural Languages Processing*, pages 219–225. ACM, New York, 1978.
24. Rohini Srihari. Combining path-based and node-based reasoning in SNePS. Technical Report 183, Department of Computer Science, SUNY at Buffalo, Buffalo, NY, 1981.
25. Kit Fine. A defence of arbitrary objects. *Proceedings of the Aristotelian Society*, Supp. Vol. 58:55–77, 1983.
26. L. Wos, R. Overbeek, E. Lusk, and J. Boyle. *Automated Reasoning: Introduction and Applications*. Prentice-Hall, Englewood Cliffs, NJ, 1984.
27. Mark E. Stickel. Automated deduction by theory resolution. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence (IJCAI'85)*, volume 2, pages 1181–1186, Los Altos, CA, 1985. Morgan Kaufmann.
28. C. J. Date. *An Introduction to Database Systems*, volume 1. Addison-Wesley, Reading, MA, 3 edition, 1981.
29. Joongmin Choi. *Experience-Based Learning in Deductive Reasoning Systems*. Technical report 93-20, Department of Computer Science, State University of New York at Buffalo, Buffalo, NY, 1993.
30. Fritz Lehmann, editor. *Semantic Networks in Artificial Intelligence*. Pergamon Press, Oxford, 1992.
31. Ronald J. Brachman and Hector J. Levesque, editors. *Readings in Knowledge Representation*. Morgan Kaufmann, San Mateo, CA, 1985.
32. Francesco Orilia and William J. Rapaport, editors. *Thought, Language, and Ontology: Essays in Memory of Hector-Neri Castañeda*. Kluwer Academic Publishers, Dordrecht, 1998.

Composition Norm Dynamics Calculation with Conceptual Graphs

Aldo de Moor

Infolab, Tilburg University, The Netherlands
ademoor@kub.nl

Abstract. Network information system specification in virtual professional communities requires a legitimate user-driven approach. In such an approach, only specification changes are produced that are not only meaningful but also acceptable to all users. To do so, for each requested change, a relevant user group needs to be selected to work out the required knowledge definition changes. This paper describes the mechanism through which such a relevant user group can be calculated. The dynamics of the composition norms that guide the required specification behaviour are explained. The conceptual graph notation for four categories of specification knowledge is given. The Peirce conceptual graph workbench is used to demonstrate the composition norm dynamics calculation.

1 Introduction

Collaborative work is increasingly being done in a distributed fashion, supported by commonly available Internet-based information tools such as mailing lists or the web. We define the *virtual professional communities* in which such collaboration is to take place as communities or networks of professionals whose collaboration on activities required to realize shared goals is mostly or completely computer-enabled. The workflows of these communities are often supported by network information systems consisting of linked and configured standard information tools. The communal requirements and systems typically evolve strongly, with the users having an important role both as sources and as modellers of the system specifications. Active user participation in the specification process of such continuously evolving network information systems is very important, since community members have the most detailed knowledge about when breakdowns in work arise and how they can be resolved. One significant weakness of the traditional methods supporting specification processes is that they do not sufficiently involve the users (see [3] for a detailed study). They tend to rely on external analysts controlling the specification process, leaving the users only the rather passive role of being interviewed by them. Other methods, in particular socio-technical specification methods (such as Soft Systems Methodology), often overinvolve users in the sense of letting them participate in every conceivable change process. To increase the efficiency and willingness of users to participate in change processes, it must therefore be exactly known for each specification

B. Ganter and G.W. Mineau (Eds.): ICCS 2000, LNAI 1867, pp. 525–539, 2000.
© Springer-Verlag Berlin Heidelberg 2000

process what subset of all users is to take part, and in which capacity these are to be involved. To adequately determine the *relevant user group*, a *legitimate user-driven specification approach* is required. First, in such an approach, the community members are not just to provide specification knowledge, but also to control the process in which this knowledge is produced, thus making the specification proces truly *user-driven*. Second, the members of virtual professional communities, like their counterparts in traditional communities, are guided in their work by shared *social norms*. These norms should govern both the operations of a network and the specification processes in which the network and its information system is being defined. As these networks are egalitarian in nature, such norms cannot be imposed from above, but should originate from the community as a whole. Thus, the user-driven specification process needs to be *legitimate* as well, in the sense that specification changes are not only meaningful, but also *acceptable* to all members of the community. We call the norms that regulate the acceptable specification behaviour of the members of a virtual professional community *composition norms* (while we refer to the norms that regulate operational workflow behaviour as *action norms*).

The RENISYS (**RE**search **N**etwork **I**nformation **SY**stem **S**pecification) method [3] supports such a legitimate user-driven approach. It allows users facing a *breakdown* in their work to identify *problematic knowledge definitions* which they feel should be changed. For each of these definitions, RENISYS calculates the relevant user group, which it provides with the appropriate related knowledge definitions and the discussion environment needed for the group to work out the acceptable definition changes.

In [2], we explained how ontological and normative knowledge can be represented in conceptual graphs, and how these knowledge categories can be used to produce legitimate knowledge definition changes. In [4], we explored how the context lattices proposed in [8] can be applied to to efficiently structure, query, and update composition norms. Based on this work, we now show how the RENISYS method uses conceptual graph theory to determine the exact relevant user group required for a particular required specification change. To do so, the *composition norm dynamics* need to be calculated. In this way, a set of applicable norms can be calculated for each user and composition (part of the specification process necessary to resolve the breakdown). By then calculating what is the resultant effect of such a norm set, the method can determine whether a particular user is permitted, required, or prohibited to take part in certain stages of the specification process. This calculation, however, falls outside the scope of the current paper (see [3] for a detailed description).

Sect. 2 describes the semantics of composition norm dynamics. The conceptual graph notation of the various categories of knowledge definitions that are the output of specification processes are explained in Sect. 3. In Sect. 4, it is shown how the norm dynamics can be calculated using a standard conceptual graph workbench. The final section contains some discussion and conclusions.

2 Composition Norm Dynamics

The structure of composition norms has been extensively discussed elsewhere [3]. We therefore restrict ourselves here to briefly outline their main elements:

• *deontic effect*: the intended effect of a norm on the person who is to make a composition. A composition is either *permitted*, *required*, or *forbidden*.

• *actor*: the role, for example that of editor or reviewer, that a person is to play in order to be affected by the norm.

• *control process*. This concerns either an *initiation*, *execution*, or *evaluation* of the specification process at hand.

• *specification process*. In a specification process, a knowledge definition is changed. These change processes are either *creations*, *modifications*, or *terminations* of such definitions. The knowledge definitions themselves can be of four different categories: type definitions, state definitions, action norms, and composition norms. The role of the norms was already described in the previous section, while type definitions describe ontological knowledge, and state definitions represent states-of-affairs. For each knowledge definition category, a separate specification process has been defined. Thus, there are twelve customized specification processes, such as 'Create_Type' or 'Terminate_State'. The characteristics of these knowledge definitions and their specification processes have been discussed in detail in [4] and [3]. Their conceptual graph notation is presented in Sect. 3.

The formal notation for a composition norm d_{cn} is the following:
$d_{cn} = \langle id, de, a, cp, sp \rangle$, where id is the identifier of the norm, de is the deontic effect, a the actor, cp the control process and sp the specification process. An example of such a norm could be: $d_{cn_1} =$
$\langle \#12, Perm, List_Owner, Exec, \underline{Modify_Type(Mailing_List)} \rangle$[1]. This norm says that a list owner is permitted to carry out changes in the (functionality) definitions of a mailing list, for example, by declaring the list to have an open instead of closed subscription procedure.

Example Assume that the set of legitimate composition norms $\mathcal{D}_{CN}$ consists of the following definitions:

- $\langle \#58, Perm, Publ_Coord, Init, \underline{Terminate_State(Reviewer)} \rangle$
- $\langle \#59, Req, Editor, Exec, \underline{Modify_Type(Review)} \rangle$
- $\langle \#60, Perm, Actor, Control, \underline{Specify(Definition)} \rangle$
- $\langle \#61, Req, Editor, Control, \underline{Create_Type(Edit)} \rangle$
- $\langle \#62, Forb, Journal_Editor, Eval, \underline{Create_Type(Edit)} \rangle$
- $\langle \#63, Req, Reviewer, Init, \underline{Create_Type(Edit)} \rangle$

[1] Note that, for simplicity, in the examples we represent the actor and control process entities by their types labels, instead of giving the full entity definition that would also include an identifier and a referent. Furthermore, the (nested) structure of the specification process is not yet formally defined, this we will do in Sect. 3. The predicate stands for the type of specification process, its argument for the knowledge definition being changed.

- $\langle$#64, $Perm$, $Reviewer$, $Control$, $\underline{Create_Type(Edit_Report)}\rangle$
- $\langle$#65, $Forb$, $Reviewer$, $Eval$, $\underline{Create_Type(Edit_Report)}\rangle$

Composition norm #58 indicates that a publication coordinator may start the removal of a particular reviewer of a journal. Norm #59 expresses that an editor must revise the review process, if prompted. Norm #60 is a very generic norm, saying that any actor may control any specification process. Such a generic norm is typically defined at the initiation of a network, when the information system is still small in scope and only few users and actor roles have been defined. Norm #61 says that an editor must control the creation of new types of edit processes. However, according to norm #62 a journal editor is not allowed to evaluate such newly created process types. This norm could be introduced to ensure that such an editor cannot manipulate the results of his own work processes. Norm #63 says that a reviewer is responsible for starting the creation of a new edit process, for example when he is no longer satisfied with the way reviews are being handled. Norm #64 permits a reviewer to fully control the creation of report edit process types. Finally, norm #65 says that a reviewer is not allowed to evaluate a newly created report edit process definition. Such a privilege could instead be granted, for instance, only to the editorial board.

The types of the various norm elements are ordered using the following type hierarchy [2]:

T >
- Definition
- Entity >
  - PD_Actor >
    - *Editor* >
      - *Journal_Editor*
    - *Publ_Coord*
    - *Reviewer*
  - Control >
    - Init
    - Exec
    - Eval
  - Activity >
    - *Edit* >
      - *Edit_Report*
    - *Review*
  - Specify >
    - Create_Type
    - Modify_Type
    - Terminate_State

Composition norms play different roles depending on the users and specification processes they apply to at a particular moment in time. We refer to the way

[2] This hierarchy is formed by the relevant parts of the ontological framework introduced in [3], combined with some new, example-based types (in italics). Note that not all intermediate types are presented here to conserve space.

in which the status of norms can change as *norm dynamics*. These dynamics for composition norms can be summarized as follows: at any time, a composition norm base contains the set of *legitimate* norms $\mathcal{D}_{CN}$. A legitimate composition norm is *invoked* when there is at least one user with whom the norm matches. Invoked norms become *active* if they match with the *active specification process*, which is the process in which a problematic definition is currently to be changed. Each specification process consists of three parts: its initiation, execution, and evaluation. These parts are called the *specification process compositions*, which in the case of the active specification process we also refer to as *active compositions*. For each combination of user and active composition, a set of *applicable norms* exists, which determines what is the acceptable specification behaviour for that user and composition.

These norm dynamics need to be known, because they restrict the sets on which norm calculations need to be carried out. This is especially important in case of large numbers of norm definitions. In order to model the norm dynamics, two matching processes need to be defined. A *user match* is defined as a match between a user and an actor component of some composition norm. This means that at least one of the actor roles that the user plays is a subtype of the norm actor component. A *composition match* is defined as a match between a *specification process composition* and the composition part of some composition norm, also called the *norm composition part*. Such a match implies that the specification process composition must be a specialization of the norm composition part, as we support the view that generic norms are stronger than more specific norms. Thus, a specification process composition matches with, i.e. is governed by some composition norm, if the norm composition part is more generic than the specification process composition. In Sect. 4 we show how to calculate this match using conceptual graphs.

Definition 1

Let there be a composition norm $d_{cn} = \langle id, de, a, cp, sp \rangle \in \mathcal{D}_{CN}$. **comp** is a function on the argument of d_{cn} that produces the norm composition part: $\mathbf{comp}(d_{cn}) = \langle cp, sp \rangle$. Furthermore, $\mathcal{U}$ is the set of users, $\mathcal{E}$ is the set of entities ($\mathcal{U} \subset \mathcal{E}$). Function $\mathbf{type}(e)$ returns the type, and $\mathbf{ref}(e)$ returns the referent of entity e.

Let there be a user $u \in \mathcal{U}$ and an actor a of some composition norm $d_{cn} = \langle id, de, a, cp, sp \rangle \in \mathcal{D}_{CN}$. There is a *user match* between u and a, denoted as $u\ \theta_u\ a$, if $\exists\, e \in \mathcal{E} \mid \mathbf{ref}(e) = \mathbf{ref}(u) \wedge \mathbf{type}(e) \leq \mathbf{type}(a)$.

Let there be some specification process composition $comp \in \mathcal{C}omp$ (the set of all possible compositions, which is the Cartesian product of all control processes and all specification processes), and a norm composition part $\mathbf{comp}(d_{cn})$. There is a *composition match* between *comp* and $\mathbf{comp}(d_{cn})$, denoted as $comp\ \theta_n\ \mathbf{comp}(d_{cn})$, if *comp* is a specialization of $\mathbf{comp}(d_{cn})$. There is such a specialization if both the control process part and the specification process of *comp* are specializations of their counterparts in $\mathbf{comp}(d_{cn})$. A specification process is a specialization of another specification process if both the

type and the embedded knowledge definition of the first are specializations of the second.

Example The set of legitimate norms $\mathcal{D}_{CN} = \{\#58, \ldots, \#65\}$ The active specification process is the creation of a report edit process type, informally labeled $\underline{Create_Type(Edit_Report)}$. We further assume that the network contains two users: John, a journal editor, and Jack, a reviewer.

For composition norm #60, for instance, the norm composition part **comp** (#60) is $\langle Control, \overline{Specify(Definition)}\rangle$. Given were a user John (more precisely: $u_1 \in \mathcal{U}$ with $\overline{\mathbf{type}(u_1) = User}$ and $\mathbf{ref}(u_1) = John$)) and an entity e_1 (with $\mathbf{type}(e_1) = Journal_Editor$ and $\mathbf{ref}(e_1) = John$)). Say there is an active composition
$comp_1 = \langle Init, \underline{Create_Type(Edit_Report)}\rangle$:

$u_1\ \theta_u a$ holds, because $\overline{\mathbf{ref}(e_1) = \mathbf{ref}(u_1) \wedge \mathbf{type}(e_1) < Actor}$ (the norm actor).

$comp_1\ \theta_n$ **comp**(#60) holds, because the various parts of *comp* are specializations of their counterparts in **comp**(#60).

We say that a (legitimate) composition norm becomes an *invoked composition norm* if there is a user match between some user and the norm actor.

Definition 2
D_{CN_I} is the set of invoked composition norms.

The function $\mathbf{D_{CN_I}}$: $\mathcal{P}(\mathcal{U}) \rightarrow \mathcal{P}(\mathcal{D}_{CN})$ determines which legitimate composition norms are in D_{CN_I} (where $\mathcal{P}$ denotes the powerset):
$\mathbf{D_{CN_I}}(\mathcal{U}) = \{d_{cn} = \langle id, de, a, cp, sp\rangle \in \mathcal{D}_{CN} \mid \exists\, u \in \mathcal{U} : u\ \theta_u\ a\}$

Example The set of invoked norms $D_{CN_I} = \{\#59, \ldots, \#65\}$. Legitimate composition norm #58 is not invoked, because there are no users playing roles that are subtypes of the $Publ_Coord$ actor role. All the other norms are invoked, because there is at least one user playing some role that is a subtype of these norms.

Whereas the invocation of legitimate norms depends on which users are participating in the community, the actual activation of the invoked composition norms depends on the currently *active specification process*. An invoked composition norm is an *active composition norm* if at least one of the active compositions (i.e. the initiation, execution, or evaluation of the active specification process) matches with the composition part of the invoked norm.

Definition 3
D_{CN_A} is the set of active composition norms.

sp_a is the active specification process. The set of active compositions $Comp_A$ $= \{\langle Init, sp_a\rangle, \langle Exec, sp_a\rangle, \langle Eval, sp_a\rangle\}$.

The function $\mathbf{D_{CN_A}} : \mathcal{SP} \rightarrow \mathcal{P}(D_{CN_I})$ determines which invoked norms are in D_{CN_A}:

$\mathbf{D_{CN_A}}(sp_a) = \{d_{cn_i} \in D_{CN_I} \mid \exists\, comp_a \in Comp_A : comp_a\ \theta_n\ \mathbf{comp}(d_{cn_i})\}$

Example The set of active norms $D_{CN_A} = \{\#60, \ldots, \#65\}$ For instance, none of the active compositions of the active specification process $Create_Type(Edit_Report)$ is a specialization of the norm composition part of invoked norm #59, which is therefore not an active composition norm. For all other invoked norms, there is at least one active composition of sp_a which is a specialization of the norm composition part.

Active norms do not have an effect on all users. We call an active composition norm *applicable* to a particular user for a particular active composition if (1) the user matches with the actor part of the norm and (2) the active composition matches with the norm composition part.

Definition 4
$D_{CN_APPL(u,comp_a)}$ is the set of applicable composition norms for user u and active composition $comp_a$. The function $\mathbf{D_{CN_APPL}} : \mathcal{U} \times Comp_A \rightarrow \mathcal{P}(D_{CN_A})$ determines which active norms are in $D_{CN_APPL(u,comp_a)}$ for u and $comp_a$:
$\mathbf{D_{CN_APPL}}(u, comp_a) =$
$\{d_{cn_a} = \langle id, de, a, cp, sp\rangle \in D_{CN_A} \mid u\ \theta_u\ a\ \wedge\ comp_a\ \theta_n\ \mathbf{comp}(d_{cn_a})\}$

Example The applicable norm sets are:
- $D_{CN_APPL(John,Init_Create_Type(Edit_Report))} = \{\#60, \#61, \#62\}$
- $D_{CN_APPL(John,Exec_Create_Type(Edit_Report))} = \{\#60, \#61\}$
- $D_{CN_APPL(John,Eval_Create_Type(Edit_Report))} = \{\#60, \#61, \#62\}$
- $D_{CN_APPL(Jack,Init_Create_Type(Edit_Report))} = \{\#60, \#63, \#64\}$
- $D_{CN_APPL(Jack,Exec_Create_Type(Edit_Report))} = \{\#60, \#64\}$
- $D_{CN_APPL(Jack,Eval_Create_Type(Edit_Report))} = \{\#60, \#64, \#65\}$

For example, $D_{CN_APPL(John,Init_Create_Type(Edit_Report))}$ contains active composition norm #60, because (1) there is a user match between John and the actor component of this norm (the journal editor role that John plays is a subtype of *Actor*) and (2) there is a composition match between active composition
$\langle Init, \underline{Create_Type(Edit_Report)}\rangle$ and norm composition part
$\langle Control, \underline{Specify(Definition)}\rangle$.

The different norm sets are depicted in Fig. 1. The various subsets depicted within the set of active norms represent the different sets of applicable norms determined above. In the next section, the informal notation of the knowledge definitions that are the object of specification processes is made formal using conceptual graph notation.

3 Specification Knowledge Definition Representation

Four different categories of specification knowledge definitions are distinguished in RENISYS: type definitions, state definitions, action norms and composition

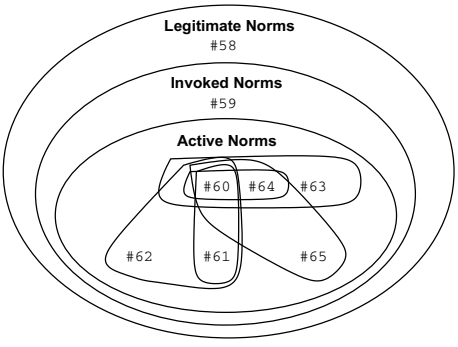


Fig. 1. Norm Dynamics Example

norms. Type definitions are used to represent ontological knowledge, state definitions represent states-of-affairs, action norms regulate operational workflow behaviour, while composition norms govern the meta-level specification behaviour. Since the role of these definitions in the specification process has already been discussed quite extensively in [2] and [4], we here only show how they are represented in conceptual graphs, along with a simple example. One main advantage of using conceptual graphs over, for example, SQL tables and operations, is that in this way generalization hierarchies of specification knowledge can be taken into account.

The general graph structure of a specification knowledge definition is:

$[k : def]$.

Here, k is a knowledge category, whereas def represents the definition core in graph format. As the specific graph representation format of the definitions varies for the various knowledge categories, their formats are discussed separately.

3.1 Type Definitions

Definition 5

A type definition $d_t = \langle id, t_d, t_g, E, R \rangle \in \mathcal{D}_T$, with t_d the defined type, t_g the genus type, E a set of entities, and R the set of relations connecting them, is defined as:

$[\texttt{Type} : [t_d : *\texttt{x}] \rightarrow (\texttt{Def}) \rightarrow [t_g : *\texttt{x}]\ \mathbf{dif}(d_t)]$.

$\mathbf{dif}(\texttt{d}_\texttt{t})$, the differentia of the type definition, is connected to the genus concept $[\texttt{t}_\texttt{g} : *\texttt{x}]$ by its relations that have the genus placeholder X in their source or destination concepts. Thus, the differentia forms a subgraph that specializes the genus to the defined type. This representation of the type definition is different from the one used by Sowa (1984,p.106). The source entity of the (Def)-relation denotes the defined type, the destination entity the genus type.

Example A (partial) type definition of the report-editing process could be:

[Type : [Edit_Report : *x] → (Def) → [Edit : *x]−
(Matr) → [Draft_Report]
(Rslt) → [Edited_Report]].

3.2 State Definitions

Definition 6
A state definition $d_s = \langle id, E, R \rangle \in \mathcal{D}_S$ is defined as:
[State : def].
def is the conceptual graph formed by the concepts in E linked by the relations in R. To be meaningful, state definitions need to be circumscribed by the available type definitions.

Example The state definition that says that Harry is the list owner of the CG-mailing list is represented as:
[State : [List_Owner : #Harry] → (Poss) → [Mailing_List : #CG]].

3.3 Action Norms

Definition 7
An action norm $d_{an} = \langle id, de, a, cp, w \rangle \in \mathcal{D}_{AN}$ is represented as follows:
[$an : a \leftarrow$ (Agnt) $\leftarrow cp \rightarrow$ (Obj) $\rightarrow w$].
de is the deontic effect of the norm. a, cp, w stand for some actor[3], control process, and workflow, respectively. The norm category label an is:

$$an = \begin{cases} \texttt{Perm_Act} & \text{if } de = Perm \\ \texttt{Req_Act} & \text{if } de = Req \\ \texttt{Forb_Act} & \text{if } de = Forb \end{cases}$$

Example The graph representation of the action norm that says that an editor may carry out the editing process is:
[Perm_Act : [Editor] ← (Agnt) ← [Exec] → (Obj) → [Edit]].

3.4 Composition Norms

Definition 8 A composition norm $d_{cn} = \langle id, de, a, cp, sp \rangle \in \mathcal{D}_{CN}$ is represented as:
[$cn : a \leftarrow$ (Agnt) $\leftarrow cp \rightarrow$ (Obj) $\rightarrow dp \rightarrow$ (Rslt) $\rightarrow def$].

Here, de, a, cp, sp mean the deontic effect, actor, control process, and specification process. $dp =$ [**type**(sp)] and $def =$ **ref**(sp). The composition category

[3] The meaning of the term *actor* is different from its interpretation in CGT, where it refers to a node in a dataflow graph that can perform computations on the declarative graph knowledge [9, p.188]

label is:

$$cn = \begin{cases} \texttt{Perm_Comp} & \text{if } de = Perm \\ \texttt{Req_Comp} & \text{if } de = Req \\ \texttt{Forb_Comp} & \text{if } de = Forb \end{cases}$$

Example Composition norm #58 is represented as:

[Perm_Comp : [Publ_Coord] ← (Agnt) ← [Init] → (Obj)−
[Terminate_State] → (rslt) → [State : [Reviewer]]].

4 Norm Dynamics Calculation with Conceptual Graphs

This section illustrates how conceptual graph theory can be used to calculate the norm dynamics discussed in Sect. 2. To this purpose we use the Peirce conceptual graphs workbench[4], which was developed by Gerard Ellis. Among other things, the Peirce tool allows for the handling of nested graphs, which are needed to represent composition norms[5].

Type Hierarchy

The knowledge base has been loaded with the type hierarchy described in Sect. 2. These type definitions are represented as follows:

```
Editor < PD_Actor.
  Journal_Editor < Editor.
Publ_Coord < PD_Actor.
Reviewer < PD_Actor.
[...]
```

State Definitions

The Peirce knowledge base contains these state definitions to describe that user John is a journal editor and Jack a reviewer:

```
[State: [User: #John]].
[State: [User: #Jack]].
[State: [Journal_Editor: #John]].
[State: [Reviewer: #Jack]].
```

Composition Norms

The set $\mathcal{D}_{CN}$ consists of legitimate composition norms #58-#65. These norms are represented, in order, as:

[4] http://www.cs.adelaide.edu.au/users/peirce/

[5] As in Peirce the symbols '@' and '.' cannot be used in the referent, they are both replaced by a '_'. The '>'-symbol indicates the prompt. A further explanation of the precise syntax of commands and graphs is given in [5] and is not repeated here.

```
[Perm_Comp: [Publ_Coord] <- (Agnt) <- [Init] -> (Obj) -
  -> [Terminate_State] -> (Rslt) -> [State: [Reviewer]]].
[Req_Comp: [Editor] <- (Agnt) <- [Exec] -> (Obj) -
  -> [Modify_Type] -> (Rslt) -> [Type: [Review]]].
[Perm_Comp: [Actor] <- (Agnt) <- [Control] -> (Obj) -
  -> [Specify] -> (Rslt) -> [Definition]].
[Req_Comp: [Editor] <- (Agnt) <- [Control] -> (Obj) -
  -> [Create_Type] -> (Rslt) -> [Type: [Edit]]].
[Forb_Comp: [Journal_Editor] <- (Agnt) <- [Eval] -> (Obj) -
  -> [Create_Type] -> (Rslt) -> [Type: [Edit]]].
[Req_Comp: [Reviewer] <- (Agnt) <- [Init] -> (Obj) -
  -> [Create_Type] -> (Rslt) -> [Type: [Edit]]].
[Perm_Comp: [Reviewer] <- (Agnt) <- [Control] -> (Obj) -
  -> [Create_Type] -> (Rslt) -> [Type: [Edit_Report]]].
[Forb_Comp: [Reviewer] <- (Agnt) <- [Eval] -> (Obj) -
  -> [Create_Type] -> (Rslt) -> [Type: [Edit_Report]]].
```

Invoked Norms Calculation

For each composition norm $d_{cn} \in \mathcal{D}_{CN}$, d_{cn} is in the set of invoked composition norms D_{CN_I} if there is a user match $u\theta_u a$, with u being some user and a the actor part of d_{cn} (see Def.1 and 2).

The (temporary) set of user referents R_U consists of the referents of the user concepts in state definitions of users. These definitions are retrieved by the following operation.

```
> (Specialisations) -> [[State: [User]]]?
[State: [User: #John]].
[State: [User: #Jack]].
true
>
```

$R_U = \{\#John, \#Jack\}$

Now, $\forall d_{cn} \in \mathcal{D}_{CN}$, with a the type label of the actor part of d_{cn}: if $\exists\, r_u \in R_U$ such that there is a specialization of $[\texttt{State} : [a : r_u]]$, then $d_{cn} \in D_{CN_I}$.

For example, for composition norm #58 (with $a = Publ_Coord$):

```
> (Specialisations) -> [[State: [Publ_Coord: #John]]]?
no specializations
true
>
> (Specialisations) -> [[State: [Publ_Coord: #Jack]]]?
no specializations
true
>
```

Thus, composition norm #58 $\notin D_{CN_I}$.

On the other hand, for composition norm #59 (with $a = Editor$):

```
> (Specialisations) -> [[State: [Editor: #John]]]?
[State: [Journal_Editor: #John]].
true
> (Specialisations) -> [[State: [Editor: #Jack]]]?
no specializations
true
```

Thus, since John is affected by it, composition norm #59 $\in D_{CN_I}$. Similar calculations can be made for the other norms. It can thus be derived that D_{CN_I} consists of composition norms #59-#65.

Active Norms Calculation

An invoked norm is also an active norm if there is a composition match between one of the active compositions and the norm composition part (see Def.3).

In order to calculate the active norms, several temporary graphs are needed:

• Every active composition $comp_a \in Comp_A$ is stored in a separate graph.

• For each invoked composition norm $d_{cn_i} \in D_{CN_I}$, its norm composition part is stored in a new graph. Now, $\forall d_{cn_i} \in D_{CN_I} : d_{cn_i} \in D_{CN_A}$ if at least one of the active composition graphs is a specialization of the norm composition part graph. For example:

• The active composition graphs are:

```
[Init] -> (Obj) -> [Create_Type] -> (Rslt) -> [Type: [Edit_Report]].
[Exec] -> (Obj) -> [Create_Type] -> (Rslt) -> [Type: [Edit_Report]].
[Eval] -> (Obj) -> [Create_Type] -> (Rslt) -> [Type: [Edit_Report]].
```

• For invoked composition norm #59, the norm composition part graph is:

```
[Exec] -> (Obj) -> [Modify_Type] -> (Rslt) -> [Type: [Review]].
```

Performing the specialization operation gives the following result:

```
> (Specialisations) -> [[Exec] -> (Obj) -> [Modify_Type] -> (Rslt) -
  -> [Type: [Review]]]?
[Exec]->(Obj)->[Modify_Type]->(Rslt)->[Type: [Review]].
true
>
```

Since the operation only returns the norm composition graph itself, and none of the active composition graphs, composition norm #59 $\notin D_{CN_A}$

• For invoked composition norm #60, however, the norm composition part graph is:

```
[Control] -> (Obj) -> [Specify] -> (Rslt) -> [Definition].
```

For this graph, the specialization operation returns:

```
> (Specialisations) -> [[Control] -> (Obj) -> [Specify] -> (Rslt) -
-> [Definition]]?
[Exec]->(Obj)->[Create_Type]->(Rslt)->[Type: [Edit_Report]].
[Init]->(Obj)->[Create_Type]->(Rslt)->[Type: [Edit_Report]].
[Control]->(Obj)->[Specify]->(Rslt)->[Definition].
[Eval]->(Obj)->[Create_Type]->(Rslt)->[Type: [Edit_Report]].
true
>
```

At least one (in fact, all three) of the active composition graphs are returned, so composition norm #60 $\in D_{CN_A}$. Similar calculations can be made for the other norms. D_{CN_A} consists of composition norms #60-#65.

Applicable Norms Calculation

For each combination of user u and active composition $comp_a$, a set of applicable norms $D_{CN_APPL(u,comp_a)}$ is defined (see Def.4). An active norm is in such a set if there are both a user match between u and the actor part of the norm, and a composition match between $comp_a$ and the norm composition part.

For example, to calculate $D_{CN_APPL(u,comp_a)}$, with u = John and $comp_a$ = [Init]− > (Obj)− > [Create_Type]− > (Rslt)− > [Type : [Edit_Report]]:

• To see whether composition norm #60, which has *Actor* as the type label of its actor part, is in this set, we first need to determine whether there is a user match:

```
> (Specialisations) -> [[State: [Actor]]]?
[State: [User: #John]].
[State: [User: #Jack]].
[State: [Journal_Editor: #John]].
[State: [Reviewer: #Jack]].
true
>
```

Thus, there is indeed a user match between John and the actor part of norm #60. Now, it must be seen if there is a composition match as well.

The norm composition part graph for norm #60 is:

```
[Control] -> (Obj) -> [Specify] -> (Rslt) -> [Definition].
```

The matches with the active composition graphs are:

```
> (Specialisations) -> [[Control] -> (Obj) -> [Specify] ->
(Rslt) -> [Definition]]?
[Exec]->(Obj)->[Create_Type]->(Rslt)->[Type: [Edit_Report]].
[Init]->(Obj)->[Create_Type]->(Rslt)->[Type: [Edit_Report]].
[Control]->(Obj)->[Specify]->(Rslt)->[Definition].
[Eval]->(Obj)->[Create_Type]->(Rslt)->[Type: [Edit_Report]].
true
>
```

Thus, $comp_a$ is in the set of results, and the composition match is therefore successful. Since both the required user and composition match exist, composition norm #60

$\in D_{CN_APPL(John,\ [Init]->(Obj)->[Create_Type]->(Rslt)->[Type:\ [Edit_Report]])}$.

Similar calculations can be made for the other composition norms in this set, as well as for the other applicable norm sets. A more efficient calculation would reuse the results of the user matches done in the calculation of the invoked norms, and the composition matches done for the active norms calculation. For clarity, the specialization operations were repeated here, however.

5 Discussion and Conclusions

Existing specification approaches are not very well-suited for network information system specification for virtual professional communities, since these require a legitimate user-driven approach. Traditional waterfall-based specification

methods, such as SDM, are quite rigid and depend to a large extent on external analysts controlling the specification process [1]. Other methods, notably those based on a socio-technical paradigm, such as Soft Systems Methodology or ETHICS, assign a more prominent role to active user participation in the specification process [11,6]. However, they still do not adequately support evolutionary systems development and are indiscriminate in which users to in involve in what particular specification change processes.

In this paper, we have demonstrated how to calculate the relevant group of users to involve in a particular specification change process. To this purpose, a user facing a breakdown in his work can identify *problematic knowledge definitions*, which he or she would like to see changed. Composition norms are essential to precisely regulate the specification processes needed to resolve these problematic definitions. They describe the meta-level change behaviour. This in contrast with numerous workflow modelling methods, either activity-based (i.e. specifying logistical workflows), or conversation-based (modelling communications and commitments) that do not provide guidelines on *who* is to change *what* [7]. In [3] we describe how to determine the *resultant deontic effect* of a set of applicable norms, which states whether a particular user is ultimately permitted, required, or forbidden to control (i.e. initiate, execute, or evaluate) a particular specification process. This, among other things, requires for occurring norm conflicts to be resolved, which we have done making use of work done in dynamic deontic logic such as described in [12]. The actual change process is a form of a conversation by the selected users from the relevant user group. A Specification Process Model, based on Van Reijswoud's speech-act theory-based Transaction Process Model [10], prescribes the conversational moves that the various users can make. A prototype web server with mail functionality has been developed that can be used to support the specification process of a restricted set of knowledge definitions. Several case studies have been done that demonstrate how this support can facilitate network evolution. The still limited functionality of the tool will soon be upgraded to provide robust support for the full specification process.

Conceptual graph theory provides the theoretical constructs and tools to allow for such specification knowledge to be represented in a concise way and for the necessary calculations to be carried out efficiently. In [2], the importance of finding new applications such as these for CGT was discussed. We have now concretely demonstrated how existing tools such as the Peirce conceptual graph workbench can be applied to supporting the legitimate user-driven specification process. Of course, much work still needs to be done on optimizing the algorithms used, and on the integration of standard conceptual graph tool functionality with the RENISYS tool. These optimization and integration problems is the subject of current and future research.

References

1. F.P. Brooks. *The Mythical Man-Month: Essays on Software Engineering.* Addison-Wesley, anniversary edition, 1995.

2. A. De Moor. Applying conceptual graph theory to the user-driven specification of network information systems. In *Proceedings of the Fifth International Conference on Conceptual Structures, University of Washington, Seattle, August 3–8, 1997*, pages 536–550. Springer-Verlag, 1997. Lecture Notes in Artificial Intelligence No. 1257.
3. A. De Moor. *Empowering the User: A Method for the Legitimate User-Driven Specification of Network Information Systems*. PhD thesis, Tilburg University, The Netherlands, 1999.
4. A. De Moor and G. Mineau. Handling specification knowledge evolution using context lattices. In *Proceedings of the Sixth International Conference on Conceptual Structures, ICCS'98, Montpellier, France, August 10–12, 1998*, pages 416–430, 1998.
5. G. Ellis and S. Callaghan. *PEIRCE User Manual*. Peirce Holdings International, Fitzroy, Australia, 1997.
6. R. Hirschheim and H.K. Klein. Realizing emancipatory principles in information systems development: The case for ETHICS. *Management Information Systems Quarterly*, 18(1):83–109, 1994.
7. M. Klein. Challenges and directions for coordination science. In *Proceedings of the Second International Conference on the Design of Cooperative Systems (COOP'96), Juan-les-Pins, France, June 12-14*, pages 705–722, 1996.
8. G. Mineau and O. Gerbé. Contexts: A formal definition of worlds of assertions. In *Proceedings of the Fifth International Conference on Conceptual Structures, University of Washington, Seattle, August 3–8, 1997*, pages 80–94. Springer Verlag, 1997. Lecture Notes in Artificial Intelligence, No. 1257.
9. J.F. Sowa. *Conceptual Structures: Information Processing in Mind and Machine*. Addison-Wesley, 1984.
10. V. Van Reijswoud. *The Structure of Business Communication: Theory, Model and Application*. PhD thesis, Delft University, 1996.
11. R. Vidgen. Stakeholders, soft systems and technology: Separation and mediation in the analysis of information system requirements. *Information Systems Journal*, 7:21–46, 1997.
12. R.J. Wieringa, J.-J.Ch. Meyer, and H. Weigand. Specifying dynamic and deontic integrity constraints. *Data and Knowledge Engineering*, 4:157–189, 1989.

From PROLOG++ to PROLOG+CG : A CG Object-Oriented Logic Programming Language

Adil Kabbaj[1] and Martin Janta-Polczynski[2]

[1] INSEA, Rabat, Morocco, B. P. 6217
Fax : (212) 7 77 94 57
akabbaj@insea.ac.ma
[2] Glendon College — York University
2275 Bayview Avenue — Toronto, Canada, M4N 3M6,
Fax 1 (416) 487–6852
janta@glendon.yorku.ca

Abstract. In [11], we presented PROLOG++, a CG-based conceptual and contextual extension of PROLOG. This paper discusses some limitations of PROLOG++ and presents a more expressive, efficient and uniform version, called PROLOG+CG which overcomes the limitations. PROLOG+CG is a CG object-oriented logic programming language suited for many CG applications. Contrary to PROLOG++ which has been implemented in C-Prolog, PROLOG+CG is implemented directly in JAVA 2. Thus and thanks to the change of the implementation, PROLOG+CG incorporates CG (both simple and compound) as a basic data structure, beside term and list. Other possibilities, discussed in this paper, are offered by PROLOG+CG due to its direct implementation in JAVA 2.

1. Introduction

To achieve more expressive power, the PROLOG language has been extended in at least two directions :

- *Conceptual extension* : a goal can be represented by a term (a predicate) or by a complex structure, like typed feature structure [1, 2, 3, 10, 18].
- *Contextual extension* : it is illustrated first by object-based PROLOG [14, 16, 4, 5] where a Prolog program is partitioned into objects (or worlds, theories, modules, bases, spaces or other similar terms), each object contains a set of rules. A goal is then resolved in the context of a specific object.
  Contextual extension of PROLOG is illustrated also by object-oriented PROLOG [7, 15, 17] where inheritance between objects is the norm.

In the Conceptual Graph community, some systems [6, 8, 9] incorporated a deductive component that interprets a set of rules, all the goals of a rule are represented by simple Conceptual Graphs (CG). These components do not subsume PROLOG and were not presented as extensions of PROLOG.

B. Ganter and G.W. Mineau (Eds.): ICCS 2000, LNAI 1867, pp. 540 −554, 2000.
© Springer-Verlag Berlin Heidelberg 2000

To fulfill this gap in the CG research (i.e., a need for a CG based extension of Prolog), we proposed in [11, 12] *PROLOG++* ; a conceptual and contextual extension of PROLOG.
This paper discusses some limitations of PROLOG++ and presents a more expressive, efficient and uniform version, called *PROLOG+CG* which overcomes the limitations (the name of the language is changed from PROLOG++ to PROLOG+CG to avoid the conflict with the PROLOG++ of Moss [17]).

The beta version of PROLOG+CG is available from the site :
www.insea.ac.ma/CGTools/PROLOG+CG.htm

The paper is organized as follows : section 2 gives a brief review of PROLOG++ with a description of some of its limitations. PROLOG+CG, the new version of PROLOG++ which overcomes these limitations, can be depicted by the following equation : *PROLOG+CG = J-PROLOG + CG + Object-Based + Inheritance.* Section 3 presents the core of PROLOG+CG : J-PROLOG with CG, section 4 introduces the object level while inheritance is discussed in section 5.
Section 6 gives an outline of current and future works concerning PROLOG+CG. We then conclude the paper.

2. Brief Review and Limitations of PROLOG++

A PROLOG++ program [11, 12] is composed of a declarative knowledge base DKB (type hierarchy + conceptual structures), a strategic knowledge base SKB (a set of objects) and standard Prolog rules. Each rule in an object is prefixed by a term which represents the descriptor of the object. CG is used basically in DKB to represent conceptual structures and it is used in SKB to represent the structural part of the object, the method definition and the method invocation. Here is an example of an object rule in PROLOG++ : the rule is prefixed by the descriptor "seriousDisease", its head is a CG and its tail begins with a message to the object "disease", the contain of this message is a CG.

```
seriousDisease::{[sensitive]-pat->[person: P]
                                 obj->[dispute] } :-
   disease::{[gotDisease]-obj->[diseaseCardiac]
                    pat->[person:P]-fatherOf->[man: M] &
               [beingDead]-pat->[man: M]
                              cause->[diseaseCardiac]},
   attrIndividu::{[person: P]-withAge->[age: A]},
   validateValue(A),
     ...    .
```

Please, notes that a goal in the tail of an object rule or in a standard Prolog rule can be a message or a term. CG unification is used when the goal is a message.
CG is considered in PROLOG++ as a "pseudo" data structure; a PROLOG++ program is translated to an equivalent C-Prolog program and a CG is translated to a list structure. A term that has a CG as an argument considers this later as a list, except

for CG operations (MaximalJoin, Generalize, Subsume, etc) which are provided by PROLOG++ as primitive terms/goals and which treat their arguments as CG.
Let us consider now some limitations of PROLOG++ :

- CG is not considered as a data structure of PROLOG. As explained above, when a CG is used as an argument of a term (other than a primitive CG operation), it is considered as a list. Term unification has not been extended to apply CG unification when the corresponding arguments of the terms to be unified are CG.
- A goal that is not a message, can be represented by a term only.
- Inheritance among objects was not considered.
- The interpreter of PROLOG++ has been implemented using C-Prolog. A PROLOG++ program is rewritten in an equivalent C-Prolog program. PROLOG++ is thus dependent upon C-Prolog.

To overcome these limitations we have developed our own version of PROLOG : J-PROLOG. J-PROLOG interpreter and environment are implemented in JAVA 2, assuring thus its portability.
Then, J-PROLOG has been extended by adding CG, object and inheritance. The result of these extensions of J-PROLOG is called PROLOG+CG.

3. The Core of PROLOG+CG : J-PROLOG with CG

The core of PROLOG+CG is J-PROLOG with the addition of CG as a basic data structure, beside Term and List. Thus, elementary data of PROLOG+CG are Numbers, Booleans, Identifiers and Strings. Composed data are Lists, Terms and CGs. An argument of a term, an element in a list or a value of a concept in a CG can be an elementary or a composed data.
The head of a rule can be a term or a CG. Finally, a goal in the tail of a rule can be represented by a term, a CG or a variable. See appendix 1 for the syntax of PROLOG+CG and appendix 2 for a brief description of its environment.

CG in PROLOG+CG. A CG g in PROLOG+CG has the following characteristics :

- *g is functional :* any concept of g has mutually exclusive incoming relations and mutually exclusive outgoing relations.
- g can be simple or compound.
- A concept of g has the following structure :
  [Type :Referent =Value] like [Age :MyAge =36]
  where *Type* stands for a specific concept type or for a variable, *Referent* (optional) stands for an individual identifier, a "multi-referent" or a variable, *Value* (optional) stands for a PROLOG+CG data (including a list, a term or a CG) or for a variable. A *mutual-referent*, which has the form '*Number', is used in the linear notation of a CG to identify several occurrences of the same concept.
- Relations in g are dyadic.

The use of CG requires the definition of a hierarchy of concept types and the declaration of instances of types (particulars). PROLOG+CG offers for that purpose *specialization rules* and *instantiation rules*.

Specialization rule. A specialization rule describes for a type Type0 its immediate subtypes Type1, Type2, …, TypeN. It has the following form :

```
Type0 > Type1, Type2, ..., TypeN.
```

Example :

```
Universal > Person, Action, Object, Attribute.
```

Currently, PROLOG+CG provides three primitive operations for concept types hierarchy :

- `subType(Type1, Type2)` which verifies if Type1 is a subtype of Type2.
- `maxComSubType(Type1, Type2, Type3)` which verifies that (or looks for) Type3 is the maximal common subtype of Type1 and Type2.
- `minComSuperType(Type1, Type2, Type3)` which verifies that (or looks for) Type3 is the minimal common supertype of Type1 and Type2.

Instantiation rule. An instantiation rule specifies for a type Type0 its instances Inst1, …, InstN :

```
Type0 = Inst1, ..., InstN.
```

Example :

```
Man = Jo, Mark.
```

Here is a PROLOG+CG program that makes use of CG, specialization rules and instantiation rules.

```
  // Specialization rules that describe the concept
  // types hierarchy
Universal > Person, Action, Object, Attribute.
Object > House, Restaurant, Walnut, Shell, Spoon.
Attribute > Classical, Age, Easily.
Person > Man, Woman.
Action > Perform, Go, Work, Buy, Eat, Search.

  // Instantiation rules that describe the instances
  // of some types
Man = Jo, Mark.
Woman = Mary, Jane.

  // Inference rules. Notes that a goal can be a term or
  // a CG. "x", "w" and "a" represent variables.

goodSister(x):-
    employee(x),  [Woman : x]-attr->[Classical] .

[Woman : w]-attr->[Classical] :-
   [Work]-
       -near->[House]-poss->[Woman: w]-ageOf->[Age = a],
       -agnt->[Woman : w],
```

```
   inf(a, 40).

    // An example of a fact described by a CG. This
    // later contains a multi-referent "*1" to identify
    // the concept [House : *1].
[Work]-
   -agnt->[Person : Jane]-
            -ageOf->[Age = 30],
            <-poss-[House: *1]<-nearOf- [Restaurant];,
   -near->[House : *1].

    // Examples of facts described by terms.
employee(Mary).
employee(Jane).

    // An example of compound CG
[Person]<-agnt-[Perform]-obj->
    [Action = [Eat]-
              -obj->[Walnut = wal2]-part->
                            [Shell : myShell = toto],
              -instr->[Spoon]-matr->[Shell: myShell]
     ]-manr->[Easily].


    // Examples of facts described by terms that
    // contain CG as their arguments.
sense("extract", [Search]-
                      -agnt->[Person],
                      -from->[Book],
                      -obj->[Information]).

sense("classical woman", [Woman]-attr->[Classical]).
```

Some requests concerning the above program are the following :

```
?- goodSister(x).
   {x = Jane}

?- goodSister(Mary).
   no.

?- [Woman : x]-attr->[Classical].
   {x = Jane}

?- [Man]<-agnt-[Perform]-obj->[Action =
         [Eat]-obj->[Walnut]-part->[Shell : x] ].
   {x = myShell}
```

```
?- sense("extract", g).
   {g = [Search]-
            -agnt->[Person],
            -from->[Book],
            -obj->[Information]}

?- sense("extract",
                [x]<-obj-[Action]-agnt->[Person]).
   {x = Information}

?- sense("extract",
                [x]<-obj-[Work]-agnt->[Person]).
   no.

   // search the CG associated to "classical woman"
   // and try to satisfy it (as the next goal of
   //   the request).
?- sense("classical woman", g), g.
   {g = [Woman]-attr->[Classical]}
```

The unification operation which is used by the PROLOG+CG interpreter has been extended to account for the use of CG as a new basic data structure. Thus, when the two elements to unify are CG, CG unification is called.

CG unification. Unify the CG g1 with the CG g2 implies to verify that g1 is a subgraph of g2 provided the unification of the corresponding concepts works [6]. A concept c1 of g1 can be unified with a concept c2 of g2 if the type, the referent and the value of c1 can be unified with the type, the referent and the value of c2 respectively. Two specific types t1 and t2 can be unified if they have a maximum common subtype t3 other than 'Absurd'. The result of referent unification should be conform to type t3. Since a concept value is a PROLOG+CG data, concept value unification corresponds to PROLOG+CG unification.

The CG unification algorithm adopted in PROLOG+CG can be described briefly as follows :

unify(g1, g2) =

1. Verify that all the relations of g1 are also relations in g2; /** *this is a weak but useful constraint. For instance, if g1 contains a relation that is not used in g2, then we can conclude that the two CG can not be unified.* **/
2. Determine an entry point for g1 and g2 :
   if g1 and g2 contain respectively two concepts c1 and c2 that have the same specific referent,
   then consider c1 and c2 as the two entry points for g1 and g2. In this case, unify(g1, g2) will be deterministic, without backtracking.
   else search two identical relations in g1 and g2 and consider them as the two entry points for g1 and g2 (or, in terms of concepts, consider the concept sources of the two relations as entry points). unify(g1, g2) could backtrack

in this case since g1 and g2 can contain several relations with the same identifier.

3. Starting with the unification of the two entry points, propagate the unification to the two graphs g1 and g2.
4. Check that g1 is a sub-graph of g2 : all the concepts and branches of g1 have been unified with concepts and branches of g2.
5. **if** the unification of g1 and g2 fails and entry points were determined by the search of identical relations in g1 and g2
   then backtrack to search for another identical relations (and restart from 3).

Please notes that since CG unification is integrated to Prolog unification, constraints that result from CG unification influence the rest of the Prolog unification and vice versa.

4. Objects in PROLOG+CG

Object in PROLOG+CG. *An object* is a set of "inference rules" prefixed by terms with an identical signature. An object has the following form :

```
T1::R1
T2::R2
…
Tn::Rn
```

Where T1, T2, …, Tn represent terms with the same signature and R1, R2, …, Rn stand for PROLOG+CG inference rules. The common signature of T1, T2, …, Tn constitutes the *descriptor of the object*.

An inference rule in PROLOG+CG. An inference rule, in an object or in the outmost context (i.e., the program), has one of the following two forms :

```
G.
Gh :-  G1, G2, …, Gn.
```

where :

- G and Gh represents a goal that can be a term or a CG. It can also be a variable if the rule is inside an object.
- G1, G2, … and Gn represent goals, each one can be a term, a CG, a variable or a message (see the next definition).

Sending a message. *Sending a message* to an object is expressed by a *composed goal* : `T::G`, where T represents a term and G a goal (i.e., which could be a term, a CG or a variable). `T::G` can be read : "send a message to the object O_T to execute (satisfy) the goal G". The descriptor of the object O_T is the same as the signature of T. To respond to a message `T::G`, the interpreter locates first an object with a descriptor that has the signature of the term T. Then it searches, inside the object, a rule `Ti::Ri` such that `T` can unify with `Ti` and `G` can unify with the head of the rule `Ri`.

Remark. A rule can be prefixed also by a CG : `CG::R`. In this case, all the rules of the program that are prefixed by CG constitute one object. Also and as a consequence, a message can have the form : `CG::Goal`.

The following program illustrates the use of objects in PROLOG+CG : an object with the descriptor `hamza` is defined. The object contains a fact that describes some attributes of `hamza` and a rule that describes a method to compute the age of `hamza`.

```
hamza::[PERSON]-DateOfBirth->[BIRTH]-ptime->
                              [DATE = (5, 04, 1995)].

hamza::Age(A) :-
   currentDate(D1),
   hamza::[PERSON]-DateOfBirth->[BIRTH]-ptime->
                                        [DATE = D2],
   diffDate(D1, D2, A).

currentDate((14, 12, 1999)).

diffDate((x_Day2, y_month2, z_year2),
          (x_Day1, y_month1, z_year1),
          (x_Day, y_month, z_year)      ) :-
   val(x_Day, sub(x_Day2, x_Day1)),
   val(y_month, sub(y_month2, y_month1)),
   val(z_year, sub(z_year2,  z_year1)), /.
```

An example of request (i.e., send the message Age(x) to `hamza`) :

```
?- hamza::Age(x).
    {x = (9, 8, 4)}
```

Here is another object-based program that illustrates how conceptual structures for a concept type can be encapsulated within an object. It shows also how they can be used.

```
Universal > Animate, Inanimate, Action.
Action > Extract.
Animate > Person.
Person > Student, Employee.
Student > ResearchAssistant.
Employee > ResearchAssistant.
Inanimate > Text.
Text > Book.

    // Conceptual structures for the type Extract
    // constitutes an object.
Extract(canon)::[Extract]-
                      -agnt->[Person],
                      -obj->[Inanimate].
```

```
Extract(schema)::[Extract]-
                        -agnt->[Person],
                        -obj->[Text],
                        -target->[Book].

Extract(schema)::[Extract]-
                        -agnt->[Person],
                        -obj->[Inanimate : *1],
                        -manr->[Strong],
                        -target->[Inanimate]-on->
                                    [Inanimate : *1].
```

Consider now the next rule : it checks if the given information in G can be unified with a schema for the given type v_type :

first, it creates a term v_term = v_type(schema) from the list (v_type, schema),

second, it searches a schema for the type v_type that can be unified with G.

```
checkSchemas(v_type)::G :-
   term_list(v_term, (v_type, schema)),
   v_term::G .
```

Let us consider now some requests concerning the above program :

```
    // search all the schemas for the type Extract :
?- Extract(schema)::G.
    {G = [Extract]-
                -agnt->[Person],
                -obj->[Text],
                -target->[Book]}

    {G = [Extract]-
                -agnt->[Person],
                -obj->[Inanimate]<-on-[Inanimate : *1],
                -manr->[Strong],
                -target->[Inanimate : *1]}

// Is the information [Extract]-target->[Inanimate]
// contained in one of the Extract schemas ?
?-checkSchemas(Extract)::[Extract]-target->[Inanimate].
    {}
    {}

?-checkSchemas(Extract)::[Inanimate]<-from-[Extract]
                                        -obj->[Person].
    no.
```

5. Object Inheritance in PROLOG+CG

Object inheritance in PROLOG+CG is based on the approach proposed by McCabe [15]. Inheritance between objects is defined by *inheritance rules*.

Inheritance rule. It has the following form :

```
Term1 <- Term2.
```

where Term1 and Term2 are two terms that represent two objects. The above inheritance rule means : the object identified by Term1 is a specialization of the object identified by Term2. If a message that is sent to an object cannot be satisfied, the interpreter will search an inheritance rule for that object (if it has one) in order to delegate the message to its super-object.
Here is an example that illustrates object inheritance in PROLOG+CG.

```
Universal > Form, Attribute.
Form > Rectangle.
Rectangle > Square.
Attribute > Perimeter, Surface, Width,Heigth,Beautiful.

Rectangle(H, W)::[Rectangle]-
                      -permOf->[Perimeter],
                      -surfOf->[Surface],
                      -widthOf->[Width = W],
                      -heigthOf->[Heigth = H]   :-
   not(inf(W, H)).

Rectangle(H, W)::Perimeter(P) :-
   val(P, mul(2, add(H, W))).

Rectangle(H, W)::Surface(S) :-
   val(S, mul(H, W)).

   // An inheritance rule.
Square(C) <- Rectangle(C, C).
Square(_)::[Square]-attr->[Beautiful].
```

Some requests concerning the above program are :

```
?- Rectangle(4,5)::Perimeter(P).
    {P = 18}

?- Square(4)::Perimeter(P).
    {P = 16}
```

The primitive goal CreateInstance. PROLOG+CG provides the primitive goal `CreateInstance(Ident, Term)` which enables the creation of an instance

Ident from an object identified by Term. When satisfied, createInstance(Ident, Term) will add the following inheritance rule to the program : `Ident <- Term`.

The following request illustrates the use of createInstance (assuming the above program) :

```
?- createInstance(sq1,  Square(6)),  sq1::Surface(S).
  {S =  36}
```

6. Current and Future Works

PROLOG+CG constitutes a promising platform for many kinds of applications. However in order to a more powerful and operational platform, current and future work is planned to extend PROLOG+CG and its environment at different levels.

Current Works
We expect to realize the following extensions before august 2000 :

- *At the language level and especially the primitives level :* incorporating other data structures (and related operations) like Vector, Hashtable and multi-media data (text, html, image, sound, video).
  Also, current works concern the various operations on simple and compound CG : MaximalJoin, Specialize, Generalize, Subsume, Analog, ExpandConcept and Contract.
- *At the interpreter level :* incorporating the "Expert System Mode". If this mode is activated, the interpreter, when it attempts to resolve an unknown goal, should ask the user for its truth value. Also, it should be able to justify its behavior; respond to the Why/How questions.
- *At the interface level :* replacing the text editor of PROLOG+CG environment by an hyper-text like editor. This later will be useful when data is a Vector, a Hashtable or a multi-media data. Also, the editor will enable a compression/decompression of a context : if the value of a concept is a CG, this later can be hidden (by a compression operation) or shown (by a decompression operation).
  Another current work concerning the interface is the development of a debugger with trace facilities.

Future Works
Some future works include :

- Complete the "Expert System Mode" by adding forward–chaining.
- Extend PROLOG+CG to enable constraint-based programming.
- Incorporate the "knowledge base dynamic formation algorithm" [12, 13]. With the integration of this component, PROLOG+CG could be used as a memory-based language and as a general-purpose Case-Based Reasoning Tool.
- Extend PROLOG+CG with 3D and animation facilities (Java 3D will be explored for that purpose).
- Extend PROLOG+CG to account for concurrent and multi-thread programming.

- Extend PROLOG+CG so that it can be used as a platform for the development of multi-agent systems.
- Extend PROLOG+CG to enable connections with some data bases systems.
- Incorporate the web facilities.
- Develop a wizard for the conception of applications that include interface units like menu, tool bars, button, text field, etc.

Of course, this list is not exhaustive and this is meant as an invitation to anyone who is willing to work in order to reduce this list!

7. Conclusion

This paper discusses some limitations of PROLOG++ [11, 12] and presents the new version, PROLOG+CG, which manages to overcome them. The basic limitations and restrictions were due to the fact that PROLOG++ has been implemented with C-Prolog and a PROLOG++ program is transformed to an equivalent C-Prolog program. Thus, no change has been made to C-Prolog (the language and its interpreter) in order that we may include for instance CG as a basic data structure (beside Terms and Lists) and so as to extend C-Prolog unification accordingly.
By developing our own version of Prolog, called J-PROLOG, we were able to modify the interpreter (including the unification operation) in order to give CG a status similar to that of Terms; as a basic data structure and as a representation of a goal. A PROLOG+CG program is directly "compiled" into an internal representation in terms of objects and JAVA structures (vector, hashtable, etc.).
With these changes, PROLOG++ has been reviewed and reformulated into a more expressive, efficient, simple and orthogonal language.
To assure the portability of the language, the implementation of PROLOG+CG (environment + interpreter) has been done with JAVA 2.

Appendix 1 : The Grammar of PROLOG+CG

```
Prolog+CGProgram = (Rule | Comment) {(Rule | Comment)} .
Rule = Specialization_Rule | Instantiation_Rule |
       Generalization_Rule |  Inference_Rule.
Specialization_Rule = TypeIdentifier  ">"  TypeIdentifier
                        {","  TypeIdentifier} ".".
Instantiation_Rule = TypeIdentifier  "="
                        ReferentIdentifier
                        {","  ReferentIdentifier} ".".
Generalization_Rule=ObjDescriptor "<-" ObjDescriptor ".".

Inference_Rule      =  Head [":-"  Tail] "." .
Tail  = Goal {","  Goal}.
Head  = SimpleHead  ["::"  (SimpleHead | Variable)] .
SimpleHead  =  (Term | CG).
Goal =  SimpleGoal ["::"  SimpleGoal].
SimpleGoal = Term | CG | Variable .
```

```
ObjDescriptor = Term .
Term = Identifier [ "(" PrlgCGData {"," PrlgCGData}
                                                ")" ].
PrlgCGData = Number | Boolean | Identifier | String |
             Variable | List | Term | CG .
List  =  "("  [ PrlgCGData  {","  PrlgCGData} [ "|"
                                    Variable] ]  ")".

CG =  Concept  [OutBranch | InBranch | Branchs] .
Branchs = "-"  (OutBranch | InBranch)
               {"," (OutBranch | InBranch) }  [";"]  .
OutBranch =  "-"  RelationIdentifier  "->"  CG .
InBranch  =  "<-"  RelationIdentifier  "-"  CG .
Concept  = "["  Type  [":"  Referent]  ["="  Value] "]" .
Type = TypeIdentifier | Variable .
Referent  = ReferentIdentifier | Multi_Referent  |
            Variable.
Value  = PrlgCGData .

Comment =  "//"  {Character} .
TypeIdentifier, ReferentIdentifier,
                  RelationIdentifier  =  Identifier .
Multi_Referent  = "*" {Digit} .

Number =  Digit { Digit } .
Boolean = "true" | "false" .
Identifier =  Letter Letter {Letter | Digit | "_" } .
String  = "" { Character-other-than" } "" .
Variable =  ("_" { Letter | Digit | "_" })  |  Letter |
            (Letter (Digit | "_") { Letter|Digit|"_" }).
```

Appendix 2 : The Environment of PROLOG+CG

The integrated environment of PROLOG+CG consists of a text editor, a "compiler" and the interpreter. The compiler performs a syntactic analysis of the program and if the analysis is successful, it generates an object file that contains an internal representation of the program in terms of objects and Java structures (vector, hashtable, etc.). The interpreter works on the object file.

The environment provides three fixed windows (Figure 1) : a) *Program window* which enables the programmer to edit his program, b) *Console window* which enables the programmer to state and edit his request and to get responses from the system, and c) *Debug window which is* used by the compiler to inform the programmer about the result of the syntactic analysis.

The environment provides also a window that shows the primitive operations of the language and a window that presents the PROLOG+CG manual. Figure 1 gives a snapshot of the PROLOG+CG environment.

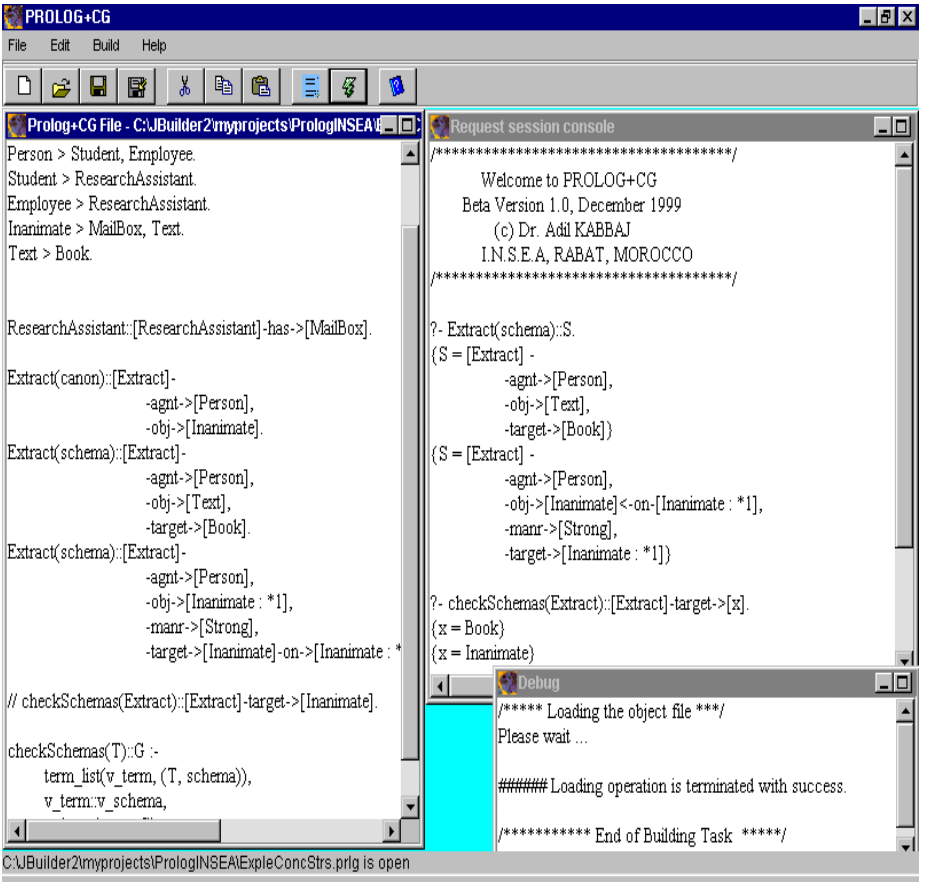


Fig. 1. PROLOG+CG environment

References

1. Ait-Kaci H. and R. Nasr. (1986), LOGIN : A logic programming language with built-in inheritance, Journal Logic Programming, 3, pp. 185-215.
2. Ait-Kaci H. and A. Podelski (1992), Logic programming with functions over order-sorted feature terms, in E. Lamma and P. Mello (Eds.), Extensions of Logic Programming, Springer-Verlag, pp. 100-119.
3. Carpenter B. (1992), The logic of typed feature structures, Cambridge Univ. Press.
4. Dichev C. (1993), Distributed knowledge and data processing, in Proceeding of ICO'93 Montreal, Canada, pp. 279-282.
5. Dichev C. (1992), Logic programming with worlds, in AI : Methodology, Systems, Applications, North-Holland, pp. 57-67.
6. Fargues J., Landau M-C, Duguord A. and Catach L. (1986), Conceptual Graphs for semantics and knowledge processing, IBM Journal of Research and Development, vol. 30:1, pp. 70-79.

7. Fukunaga K. and S. Hirose (1986), An experience with a Prolog-based object-oriented language, in Proc. Of OOPSLA'86, pp. 224-231.
8. Garner B. J. and E. Tsui (1988), General purpose inference engine for canonical graph models, knowledge-Based Systems, 1:5, pp. 266-278.
9. Garner B. J., E. Tsui, D. Lui, D. Lukose and J. Koh (1992), Extendible Graph Processing in Knowledge Acquisition, Planning and Reasoning, in Nagle T. E., J. A. Nagle, L. L. Gerholz and P. W. Eklund (Eds.), Conceptual Structures : Current research and practice, Ellis Horwood.
10. Herzog O. and C. –R. Rollinger (Eds.) (1991), Text Understanding in LILOG, Springer-Verlag.
11. Kabbaj A., C. Frasson, M. Kaltenbach and J-Y Djamen (1994), A Conceptual and Contextual Object-Oriented Logic Programming : The PROLOG++ language, in W. M. Tepfenhart, J. P. Dick and J. F. Sowa (Eds.), Conceptual Structures : Current Practices, ICCS'94, pp. 251-274.
12. Kabbaj A. (1996), Un systeme multi-paradigme pour la manipulation des connaissances utilisant la theorie des graphes conceptuels, Ph. D. Thesis, Univ. De Montreal, Canada.
13. Kabbaj A. (1995), Self-Organizing Knowledge Bases : The Integration Based Approach, in the Proc. Of the Int. KRUSE Symposium : Knowledge Retrieval, Use, and Storage for Efficiency, Santa Cruz, CA, USA, p. 64-68.
14. Kauffmann H. and A. Grumbach (1986), MULTILOG : MULTIple worlds in LOGic programming, in the proceeding of the 7th European Conference on AI.
15. McCabe F. G. (1992), L&O : Logic and Objects, Prentice-Hall.
16. Monteiro L. and A. Porto (1989), Contextual Logic Programming, in G. Levi and M. Martelli (Eds.), Proc. 6th Int. Conf. And Symposium on Logic Programming, MIT Press.
17. Moss C. (1994), Prolog++, The Power of Object-Oriented and Logic Programming, Addison-Wesley.
18. Pletat U. and K. von Luck (1990), Knowledge Representation in LILOG, in Blasius and al. (Eds.), Sorts and Types in Artificial Intelligence, LNAI n°. 418, Springer-Verlag.

A Cost-Bounded Algorithm to Control Events Generalization

Gaël de Chalendar, Brigitte Grau, and Olivier Ferret

LIMSI-CNRS
BP 133, 91403 Orsay, France
email: (gael, grau, ferret)@limsi.fr

Abstract. In order to build descriptions of prototypical situations, we first developed a system, MLK (Memorization for Learning Knowledge), allowing us to gather events related to similar situations starting from descriptions found in texts, these events being represented by conceptual graphs. One of the stages to build these prototypes consists of generalizing some similar graphs in order to produce a description. In this paper, we present a cost bounded algorithm of conceptual graph generalization, proceeding by ascending clustering. The use of costs on the operations of generalization allows us to control the growth of the search space.

1. Introduction

Text understanding requires knowledge about concrete situations that are developed in order to recognize links between the different events. Such knowledge is often represented by schemas whose content describes characters and events involved in a situation. However handcoding schemas is a very difficult task, even on a limited domain. Automatic acquisition of situation descriptions has been [1], [2] and [3], but essentially to learn new specialized situations of predefined ones. Inside a framework that does not make any hypothesis on the existence of general knowledge, we have first conceived the system MLK (Memorization for Learning Knowledge) [4], [5], able to learn from the accumulation of its own experience. MLK memorizes each specific situation found in texts by aggregating it with an already memorized description if a similarity between their events is recognized. This process leads to incrementally build aggregated Thematic Units (TU) that are precursors of general schemas. Each event inside a TU is represented by a weighted conceptual graph (CG) [6]. In order to build a general description of a situation from a TU, we have studied how to generalize these events, i.e. conceptual graphs, to find a description level accounting for different formulations of a same event. This kind of problem is dependant on ascending clustering methods from positive examples. Generalization algorithms of conceptual graphs have already been proposed [7], [8], but they only generalize concepts, and not the graph itself. Generalizing both concepts and graphs entail a combinatory explosion when searching for all the possibilities. Therefore, we have developed a cost-bounded generalization algorithm that limits the search space and leads us to propose informative generalizations. A cost is associated with each generalization operator, and we take advantage of semantic knowledge to produce meaningful descriptions of events.

B. Ganter and G.W. Mineau (Eds.): ICCS 2000, LNAI 1867, pp. 555-568, 2000.
© Springer-Verlag Berlin Heidelberg 2000

2. System Overview

The system input is made of a set of conceptual graphs describing events related to a situation of the same kind, for example *to come in a house, to attack a woman, to stab a man with a knife, to arrest the murderer*, etc. In order to elaborate a description of the situation from these events, a main stage is to find a right level of description leading to generalize those specific graphs describing a same kind of event. In the preceding example, it would consist of proposing a description where *the woman attack* and *the man stabbing* are generalized, whereas the two other events are left as it. Then the problem can be formulated as follows: given a set of conceptual graphs, find among them those that can be generalized while keeping an informative description level. Our purpose is not to find the common generalization of all the events, but a description level keeping the specificity of the kinds of events represented by the graphs. Coming back to the example, our goal is not to generalize the four events in a single one that would be *to carry out an action.*
The method we propose consists of developing a sub-set of the possible generalizations of each graph, by generalizing concepts and removing relations. Generalization is controlled by the use of semantic knowledge: a lattice of concept types and constraints on the arguments related to some concepts, given by canonical graphs. It avoids overgeneralizations that would not have any meaning. Developing all the generalizations of a graph is unconceivable, even if limiting them by comparison with domain knowledge. Therefore, in order to limit the size of the generalization space, a cost is associated to the generalization operators, allowing the system to associate a cost to each graph produced and to limit their formation by fixing a threshold. Costs are defined according to the task and encode the generalization level that is searched. All the generalizations are organized in configurations allowing the system to know which initial graphs are generalized. These configurations represent the different possibilities to describe the initial situation. The choice of one of them is done according to the number of initial graphs that are generalized and to the cost of the different configurations. Thus, the description level that is built is constrained both by the semantic knowledge and by the costs associated to the generalized graphs. This method allows us to bypass the difficulty coming from the impossibility to define negative examples in our task and then to use classical learning algorithms.

3. Conceptual Graphs

Semantic knowledge is represented in a lattice of types of concept and by canonical graphs associated to some types in order to precise their thematic roles (i.e. relations), as *agent*, *object*, etc., and semantic constraints on concepts that might fill them (see Fig. 1). We do not follow the Conceptual Graphs standard draft NCITS.T2/98-003. The "syntactic" knowledge held by star graphs is coded by our canonical graphs. Conceptual graphs that represent events (see Fig. 2) are obtained by application of formation rules to one or several canonical graphs, as described in Sowa [6]. So, these graphs are built with respect to the constraints defined in the knowledge base. A graph resulting from the application of these rules is a specialization of one (or several) initial graph(s). In Fig. 2, the graph results from a maximal joint between two graphs

produced after restrictions on types of concept in the canonical graphs associated to *stab* and *body*. See [9] for a detailed presentation of the formation rules and their relation with subsumption. The class of CGs we consider is the class of simple CGs derived from canonical graphs, with n-ary relations and with a distinguished concept node explained below.

[Stab]-
(agent) → [Human]
(dest.) → [Body]
(instrument) → [CuttingEdge]

Fig. 1. A canonical graph

[Stab]-
(agent) → [Man]
(dest.) → [Body] -
(partOf) → [Man]
(instrument) → [Knife]

Fig. 2. An event

As our graphs represent events, we distinguish a concept that is significant for the event type, named predicate (*Stab* in Fig. 2), this concept playing a particular role inside our application. Other kinds of concept result from abstractions done during the aggregation process in MLK. They already generalize instances linked to a same predicate that are found in texts. See [4] and [5] to find details about the process that build these graphs.

4. Graphs Generalization

4.1 Method

Some graphs belonging to the initial set may contain different predicates while being very similar, as in Fig. 3. If graphs are identical except for their predicate, and if these predicates are semantically close, these graphs have to be replaced by a graph that generalize them. The same principle applies if the graphs only differ by some details in the predicate arguments. The problem is then to find the least common generalization of several graphs, given one does not *a priori* know which graphs in the initial set have to be generalized.

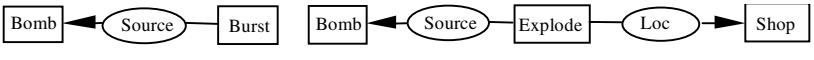


Fig. 3. Similar conceptual graphs with different structures and concepts

Such a problem is similar to conceptual clustering, with conceptual graphs as the description language of the concepts to be learned [10]. Initial conceptual graphs are equivalent to first order logic formulas, made of a conjunction of positive predicates, the concepts and the relations of the graphs. This problem is close to the work of Mineau and Bournaud. Even if we do not search for a classification, which is the problem they have to solve, the generalization space has also to be generated in order to choose the generalization level we want to reach. In their work, a graph is represented by the set of its relations to avoid comparison of graphs, that is a NP-complete problem. However, this approach cannot be applied in our application given that the structure of the graphs may be modified during the generalization process

when removing relations. We have to reason on graphs themselves in order to keep specific links between a predicate and its arguments, i.e. to maintain their connectedness. It is then impossible to generate meaningful generalizations in a limited time without controlling the growth of the generalization space. The process we propose is a cost-bounded algorithm that associates a cost to each generalization operator and uses domain knowledge to avoid overgeneralizations. This algorithm has been conceived without taking into account the specificity of our application other than the presence of a central concept in conceptual graphs. So we consider that types of relations are organized in a lattice, even if our knowledge representation does not use this possibility.

We have defined three primitive generalization operators:

1. concept generalization: replacing a type of concept by its supertype in the lattice,
2. relation generalization: replacing a type of relation by its supertype in the lattice,
3. relation removing: removing a relation in the graph, and some associated concepts.

We have not defined an operator for removing concepts because removing a concept entails the removal of all the relations it is linked to. If we come back to the example in Fig. 3, removing the concept *Shop* leads to remove the relation *Loc*, and conversely removing the relation *Loc* leads to remove the concept *Shop* in order to keep a connected graph (in such a case, we keep the connected component that contains the predicate). Removing relations is then sufficient since removing concepts is strictly included in the results of the removal of relations, which may entail the removal of several concepts and relations.

These operators create a partial order between the resulting graphs, as defined in [6] and [9]. Precisely, the subsumption relation defined for conceptual graphs is the relation induced by the existence of a projection between two graphs [11].

We associate a cost to each operator since we dispose neither a domain theory (general knowledge about situations for example), nor a characterization of the pieces of knowledge we want to learn that would give us a formal proof of the quality of the generalization. We use an empirical evaluation based on semantics and the estimation of the loss of information when generalizing.

In order to find this cost, we have reasoned on the effect of each operation in terms of loss of information it entails. In our context, the cost of each operator is related to the concept it applies to: predicate or not, concept derived from the canonical graph of the predicate or not. When generalizing a concept that is not derived from the canonical graph of the predicate, named common concept, the generalization is made on characteristics that are peripheral to the event. Removing relations, that, as already said, causes the removal of concepts, entails the suppression of some of these characteristics. As a matter of fact other concepts cannot be removed, as one cannot remove relations belonging to the canonical graph of the predicate without losing the meaning of the graph (the canonicity of the resulting graph is no more verified). Lastly, the operation we consider as very costly is the generalization of the predicate, since, as we have seen in part 2, we want to avoid many generalizations of the predicates in order to keep kinds of events specific to the described situation. So the least common generalization is a graph achieved by preferentially generalizing concepts other than the predicates.

We have defined the following order of the costs of the operators, this ordering characterizing the loss of information:

generalization of a common concept ≤ generalization of a concept derived from the canonical graph ≤ removal of a relation ≤ generalization of the predicate

The relative order is more significant than absolute values that defines costs. The value given to each operator is connected to the threshold that is fixed to compute generalizations for a given application, the number of operations it is likely to realize, and the importance of each operation in relation to the others. These values have to be fixed experimentally according to the application.

Let us now present some specificity due to the conceptual graph formalism and our application. Firstly, it is not possible to suppress the predicate, since the resulting graph(s) would no longer refer to the described event. Secondly, if two non-connected graphs result from a suppression, we only keep the graph containing the predicate. Thirdly, a concept is never generalized in a concept more general than the one in the canonical graph. And last, when generalizing a predicate, a verification is done using the new canonical graph to ensure the canonicity of the resulting graph. A consequence of having this distinguished concept is that the complexity of the operations on CGs is reduced due to less possible matching between two graphs. But that does not reduce a lot the size of the effectively searched space.

4.2 The Generalization Algorithm

One stage of the algorithm consists in building the generalization space which contains, at the beginning, the conceptual graphs to be generalized, i. e. the root nodes. During the processing, each node resulting from a generalization is inserted in this space only if it does not already exist. A node of depth n is the result of n generalizations of a root node. For example, a conceptual graph that contains 3 concepts and 2 relations may be processed by applying 5 generalizations of concept or relation and 2 removal of relation. It would generate, at most, 7 new nodes. The cost associated to each node, related to the path towards the root node, is equal to the cost of its father plus the cost of the operation which has given it birth. The cost of a root node is zero. Applying such an algorithm until a common generalization is found, if it exists, is exponential in the general case. Bounding the set of the possible generalizations by fixing a maximal cost on the nodes entails a quite reasonable practical complexity (this latter point will be detailed in the following section) and all the generalizations having a cost less than this threshold are computed.

Fig. 4 shows an example where three conceptual graphs (the root nodes x1, x2 and x3) have been generalized in the graphs (a, b, c, d), (c, b, e) and (e, f, g, d) respectively. Values on the branches are the costs of the applied operators.

Costs which stop the generalization process, more than maintaining the semantics of the graphs, yield the possibility to control the growth of the generalization space. In order to improve more effectively the performance of the algorithm, we have implemented some principles. For example, if the cost to find a supertype common to the predicate of a graph and the predicates of each other graph of the initial set oversteps the given threshold, this graph is removed from the initial set. By using this principle, the algorithm applies on graphs that may be a generalization of two root graphs, and not on the very initial graphs, given they all contain a different predicate (see *Initialization* part of the algorithm in Fig. 6). On another hand, graphs are

indexed by their predicate and their number of relations and concepts, such as comparison of graphs is only done if these characteristics are identical.

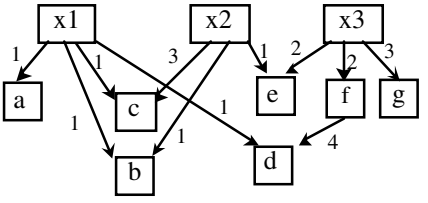


Fig. 4. A generalization space built from 3 root graphs

The aim of the algorithm is to generate all the possible descriptions of the initial set, the set of configurations. A configuration is a set of conceptual graphs, generalized or not, that represents a partition of the initial examples, the root graphs. After application of the algorithm, only the graphs that generalize several root graphs with the root graphs themselves are kept. Other nodes, that just generalize a unique initial conceptual graph, are useless for our purpose (for the example in Fig. 4, retained nodes appear in the first column in Fig. 5). The generalization cost of a node is the average cost of the generalization costs of this node, these latter costs correspond to the different path issued from the different root nodes. Finding the set of configurations consists of building all the subsets of nodes such as each initial node is generalized once and only once in the subset. Each configuration is evaluated by the average cost of its elements.

Nodes	Average costs	Origins	
c	(1+3)/2 = 2	x1	x2
b	(1+1)/2 = 1	x1	x2
d	(1+6)/2 = 3.5	x1	x3
e	(1+2)/2 = 1.5	x2	x3
x1	0	x1	
x2	0	x2	
x3	0	x3	

Configurations = {[c, x3] (2), [b, x3] (1), [d, x2] (3.5), [e, x1] (1.5), [x1, x2, x3] (0)}

Fig. 5. Results of the generalization algorithm

Configurations are computed during the generalization process (cf. *Updating configurations* in the algorithm). At each step, partial configurations, corresponding to the retained generalizations, are updated if the new graph generalizes at least two root graphs.

If more than one configuration has been built at the end of the processing, we choose the one that verifies the following criteria, applied in this order: a) the configuration that corresponds to the generalization of the maximum number of root graphs; b) the configuration containing less numerous elements; and c) the

configuration having the minimal cost, other than zero, considering it is the most likely to be the most common specific description of the initial events. By means of these criteria, we encode that the best description of the initial situation corresponds to the maximal regrouping of graphs, whose cost is less than the limit. In the example of Fig. 4, these criteria lead to choose the configuration [b, x3] having an average cost equal to 1. These criteria fit our application, even if, for other kinds of applications, another order or other criteria may have to be found.

E, set of initial graphs, E = {gi} and *n* = |E|.
Cgi, sets of generalized graphs from each root graph *gi*.
G, generalization space, the set of all computed generalized graphs. Each generalized graph is related to a list of root graphs it generalizes.
Cp, set of partial configurations, initialized to an empty set
cgp, cost to generalize a predicate
la, threshold that stops generalization

Notation: a -> b : a associated to b

Initialization
For all $g \in E$, and **For all** $g' \in E$ **do**
  Create a set *Cg*
  If the predicate of *g* has a common supertype, *sc,* with length *d* of the path between the predicate of *g'* and *sc*, such as *d* x *cgp* < *la*
  then,
    create a copy g_C of *g*,
    replace the predicate of g_C by *sc*
    add g_C ot *Cg*
    add *g* to *G*
  endIf
endFor

Abstraction
While it exists a node to develop **do**
  For all $Cgi \in Cg$, **do**
    Selection of a node *nd* from *Cgi*
    Updating configurations with *nd* ***comingFrom*** *Cgi*
    generalize *nd* -> *son(nd)*
    add *son(nd)* to *Cgi*
  endFor
endWhile

Compute the final result

Selection
Depends on the way the generalization space is searched: in our case, the first found.
Suppress the chosen graph from *Cgi*.

Updating configurations with *nd* ***comingFrom*** *Cgi*
/ updating of the generalization space/
If $nd \notin G$,
then add *nd* -> {root graph of *nd*} in *G*
else
  If *nd* had already been generated from the same root graph
  then
    If the cost of *nd* is lesser,
    then replace the old graph by *nd*
    endIf
  else
/ updating of the partial configurations /
  add, in *G,* the new root graph of *nd* to the list of its root graphs.
  remove all the configurations containing *nd*
  For all *cp* = {a, b, c ...}->{x1, x2, x3, ...}$\in$ *Cp* **do**
    If the list of root graphs of *cp* does not contain any *x'i*, root graphs of *nd*
    then
      create a copy of *cp*, add *nd* to the left member and the *x'i* to the right member:
      new configuration = {a, b, c,..., nd}->{x1, x2, x3, ..., x'1,x'2, x'3}
      add the new configuration to *Cp*
    endIf
  endFor
  create a new configuration with *nd*
  endIf
endIf

Fig. 6. The generalization algorithm

The algorithm we propose is general enough to be used in other applications. We will see in the latter sections that heuristics coming from the application domain and expressed with simple numeric values, entail the considerable pruning of the search

space, allowing the algorithm to find only the more informative generalizations. As for all algorithms depending on heuristics, it is not easy to formally demonstrate the complexity of our algorithm, thus we have worked on a demonstration of the adequacy and the efficiency of this method by extensive tests.

5. Results

The process has been applied with success to the results of MLK, but this does not entail a validation on a quantity of data sufficiently large, because text representations that compose the input of MLK are handcoded at this moment. That is why, in order to test our algorithm in a better way, we conceived a test generator. Its knowledge (lattice of types and canonical graphs) is compatible with our application and the generator is parameterizable so that it can simulate different kinds of application.

5.1 Domain Knowledge

Our application exploits a lattice of types of concepts and a set of canonical graphs. The lattice (see Fig. 8), results from the work of Chibout [12] who created an ontology of approximately 3,000 concepts for verbs and entities. We extracted a sub-part from this ontology so that to obtain a lattice regular enough in depth and in breadth. Thus, 368 concepts were extracted and distributed in a lattice with an average depth of 5.59 for a branching factor of 2.75. We have selected 8 verbs from the 128 verbs retained to assign canonical graphs to them. These canonical graphs are defined relative to the types retained in the lattice and to produce homogeneous bench tests, so that global results will have a signification.



Fig. 7. Canonical graph of the concept denoted by the verb "*to drink*"

The 8 types are *ToCauseDeath*, *ToCauseToFeel*, *ToDivide*, *ToBreak*, *ToFly*, *ToAbsorb*, *ToEat* and *ToDrink*. Subtypes inherit of the canonical graph of their father. Figure 7 shows the canonical graph of *ToDrink*. The *Function* concept is the supertype of 55 concepts. Some of them are *Artisan*, *Killer* or *Deputy*. *Liquid* is the supertype of, among others, *Water*, *Alcohol* and *Syrup*

The lattice of relation types is limited in our application to a flat lattice: all the relations have the same supertype (T) and the same subtype (⊥). For our tests, we have used four relations: *Relation*, *Agent*, *Object* and *Place*.

5.2 Bench Test Generation

We have generated sets of graphs with controllable properties for the generalization. By specializing canonical graphs in directions randomly chosen and in different

depths, our task is then to retrieve the original common graphs, or, better, some specialization of them that might not have appeared during the generation phase. So, our generator builds sets of graphs such as each set possesses a partition of its graphs where each part (named family) is made of graphs that are specialized from a graph more specific than their common canonical graph. This definition of the searched generalizations comes from our application for which it is not suitable to generalize events by their canonical graph: it does not fit an informative description.

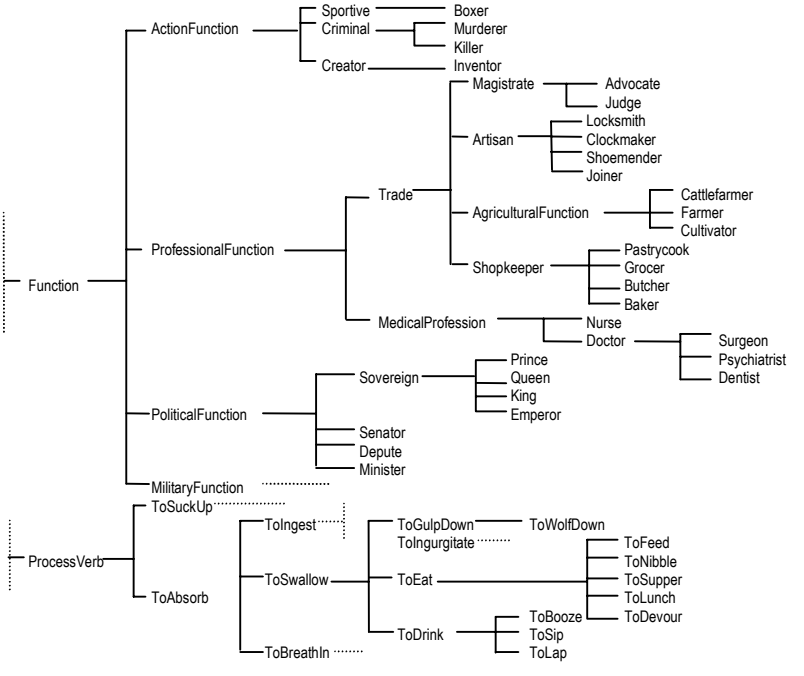


Fig. 8. Extracts from the concept ontology

To produce a bench test, the generator randomly chooses *n* predicates (P1 and P2 in Fig. 9), each one being the origin of one family. The canonical graph of each predicate is then specialized several times (s1 times in family 1 and s2 times in family 2) to generate *n* graphs, named ancestor graphs. Each ancestor graph is differently specialized (k1, k2, …, k5 times in family 1 and k1, k2 and k3 times in family 2), creating different specializations named cousin graphs.

Input data of the generalization process are the set of all the cousin graphs of different families (the 8 cousin graphs), the goal being to retrieve the set made of the ancestor graphs. The numbers of specialization operations applied (restriction of type belonging to the canonical graph or not, addition of a relation) are set at the time of the bench test specification.

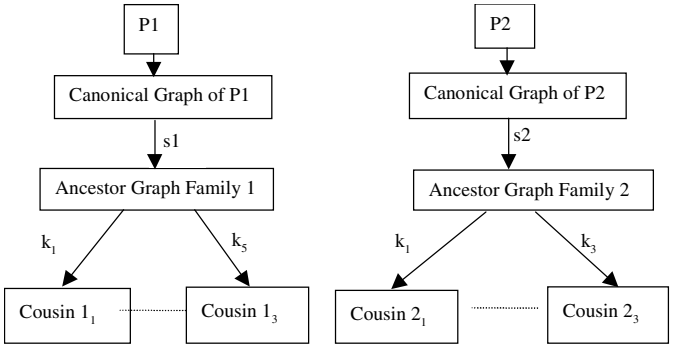


Fig. 9. Bench test generation framework

We can see below an example of test containing four graphs with two families of two graphs each. The graphs have been obtained following the parameters presented in Fig. 10.

Graph 1:
```
[ToNibble]{
– (Agent) -> [Nurse],
– (Object) -> [Cake],
– (Rel) -> [Apple]}
```
Graph 2:
```
[ToDevour]{
– (Agent) -> [Nurse],
– (Object) -> [Meat],
– (Rel) -> [Rice]}
```
Graph 3:
```
[ToKill]{
– (Object) -> [Reptile],
– (Agent) -> [Assassin],
– (Rel) -> [Car]}
```
Graph 4:
```
[ToEmbed]{
– (Agent) -> [Killer],
– (Objet) -> [Insect],
– (Rel) -> [Water]
```

Predicates	**Canonical Graphs**	**Specializations for ancestors**		**Specializations for cousins**		
		predicate	***canonical graph concepts***	***predicate***	***canonical graph concepts***	***concept adding***
ToEat	`[ToEat] -> (Agent) ->[Function] (Object) -> [Food]`	0	3	1	1	1
				1	1	1
ToCause Death	`[ToCauseDeath]-> (Agent) -> [Function] (Object)-> [Alive]`	0	3	1	2	1
				1	2	1

Fig. 10. Parameters for generating a set of graphs

This example entails the construction of 6 configurations in a total time of 115s. We can see below the optimal solution obtained in 15 seconds and a non-optimal solution. Costs of operations are specified in section 5.4.

Optimal Solution:

```
[ToEat]{
– (Agent) -> [Nurse],
– (Object) -> [Food]}
Cost: 8.0
[ToCauseDeath]{
– (Agent) -> [Criminal],
– (Objet) -> [Animal]}
Cost: 10.0
```

Non-optimal Solution:

```
[ToEat]{
– (Agent) -> [MedicalProfession],
– (Object) -> [Aliment]}
Cost: 10.0
[ToCauseDeath]{
– (Agent) -> [Criminal],
– (Objet) -> [Animal]}
Cost: 10.0
```

If we look again at the definition of a good solution given section 4.2, the optimal solution in this example respects the established criteria: maximum of root graphs generalized, minimum number of graphs in the configuration and minimal cost in the case of configuration equality. In the non-optimal configuration, the concept *Nurse* in the two graphs of the first family has been overgeneralized in *MedicalProfession*. So, we really have obtained the best solution for a cost inferior to those of the other solution. The four other solutions, not presented here, have inferior costs but do not generalize all the graphs.

We will now describe two bench tests. The first one tests a difficult case with very specialized graphs and little generalization costs. It entails the generation of a great number of configurations and high computation times. The second one shows the efficiency of the system with higher costs.

5.3 A Borderline Case

We produced 33 sets of 4 graphs containing 3 to 5 concepts and belonging to 2 families. These graphs were produced by applying 3 to 8 specialization operations on canonical graphs. The threshold for the generalization was set to 10 and a limit of processing time was fixed to 1 hour by case. For this data set, we set low values to costs of operations in order to test the borderline case: the costs allow the system to compute all the ancestor graphs. These costs are the following:

1. any concept generalization: 1,
2. generalization of one concept of the canonical graph: 1,
3. predicate generalization: 2,
4. relation suppression: 1.

The generalization process finds 32.15 configurations on average by bench test, in an average time of 28mn, 30% of the generalization processes reaching the time limit. The 70% solved cases are solved in an average time of 15mn.

5.4 Costs Utilization

To show the efficiency of mindfully chosen costs, we carried out the same kind of tests, but with costs multiplied by 2. We generated 60 bench test with 4 graphs each. With a time limit set to 1 hour, we obtained 12,2 configurations on average by case. With 20mn., we obtained 11.9. So, 97.5% of the configurations obtainable in 1 hour were obtained in less than 20mn. The following results were obtained in this time limit: 96.7% of the bench tests lead to at least one solution and 73,3% of the cases did not reach the time limit.

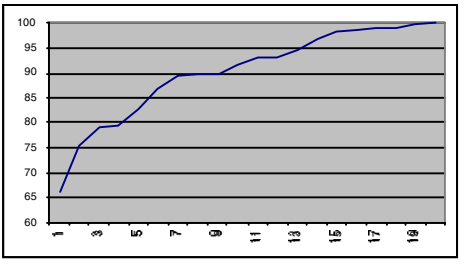


Fig. 11. Percentage of computed configurations relatively to the processing time

Figure 11 shows the percentage of configurations obtained for all bench tests in function of the time. It shows that in a reasonable time of 7mn., 90% of the solutions obtainable in 20mn (92.3% of those obtainable in 1 hour) are already obtained. Thus, according to task requirements, if it is possible to miss some solutions, then one can parameterize the generalization system with processing times really acceptable.

For these tests, the relevant solutions (i.e. informative) are thus those that contain generalized graph(s) equal to the ancestor graphs from which the graphs of the tests are issued. This relevant solution can only be found if, for each graph, the sum of the costs of the specialization operations applied to them is lower or equal to the generalization limit, here 10. The number of bench tests in this case is 22 (37%). With a processing time set to one hour, the relevant solution is found for all these cases and in 20mn., 20 solutions are found (90.9%). In the other 63%, for which the theoretical cost of the optimum solution is higher than the limit, we find nevertheless 15.8% of configurations in which the number of generalized graphs is the same as the number of families. This can be explained by the fact that cousin graphs are randomly generated. If by chance the specialization process goes the same way for some graphs, then the effective relevant solution is more specific than the ancestor graph.

5.5 Analysis

It has to be noted that in the first series of tests, the obtainable relevant solutions are obtained in 70% of the cases (23/33). In the second series, 100% of the obtainable solutions are found in one hour, however it contains a least number or test cases. So, with low costs, a great number of solutions is found and we also miss some good solutions. On the contrary, with high costs, the number of possible solutions is lower and the system finds all of them. When we will apply the system on real data, we will have to choose costs according to these two extremes.

These tests show that a cost-bounded algorithm allows us to generalize conceptual graphs in a reasonable time, in tasks where costs on the generalization operators are definable. Differences between the two series of tests show the benefits of modulating the costs and the limit threshold in order to produce interesting generalizations and to avoid the production of graphs corresponding to overgeneralizations in the application, i.e. valid graphs with a too general description level.

It is not an easy task to show the validity of solutions obtained with unsupervised learning. Thus, we have conceived our test protocol in such a way that the searched solution can be characterized. In our application framework, the validation could be

done externally by an expert who would answer the following question: "Does the produced solution seem coherent to you: are events described at a correct description level compared to the described situation?". In case of non-satisfaction, it is possible to execute the process again to search for new generalizations. Another possibility would be to validate a system using the learned knowledge.

6. Previous Works

Mineau [7] and Bournaud [8] have conceived algorithms to build a classification from an initial set of conceptual graphs. Even if we do not want to classify data, finding a generalization level can be seen as an equivalent problem as it requires the algorithm to compute possible generalizations. The major difference comes from the generalization operators used. Suppression of relations entails the necessity of a complete description of each generalized graph and not only a partial description by their relations. To limit the effective complexity, we have introduced costs associated with operators. These costs have to be fixed relative to the type of generalizations expected in the application. As in [8], we also use domain knowledge to constrain generalization.

Our work comes under Inductive Logic Programming (ILP) domain [13] and our problem is particularly close to the work of Esra Erdem and Pierre Flener [14], who redefine the minimal generalization in order to find a minimal set of generalized clauses in function of an over-generalization criterion. In the field of ILP and conceptual graphs, our kind of problem is studied in the work of Marc Champesne on the reduction of the search space by the use of the notion of "empirical subsumption" [11]. However his results can only be applied to non-connected graphs whereas our algorithm handles connected ones.

About our application itself, i.e. learning descriptions of prototypical situations, our work is different of [1], [2] and [3] essentially because we do not hypothesize the existence of previous knowledge about situations and learning is completely unsupervised. The system MLK, which implements this application, gathers and selects events and their relevant characteristics when recurrent situations are found in texts, and the system proposed in this paper assumes the construction of a general description with respect to semantic knowledge.

7. Conclusion

Learning structures in order to describe concrete situations has lead us to construct the system MLK, able to identify events linked to a same situation. In the purpose of generating general descriptions for these situations, we have studied the generalization of events represented by conceptual graphs. However the generalization algorithm we propose is independent enough of our particular context and could be used with other applications.

This algorithm is completely implemented in Smalltalk. The tests made show the benefit of fixing costs on generalization operators in applications having only a weak domain theory.

The protocol of bench tests generation we have developed allows us to control different parameters such as the definition of an optimal solution, the number of operations to be applied in order to find this solution and the homogeneity of the bench tests. It presents two major advantages: testing the efficiency of our approach and allowing us to find values for the diverse parameters of the generalization algorithm by successive tests.

At this time, we are working on the automatic acquisition of ontologies from partial syntactic analysis of phrases. This work will allow us, in a middle-term prospect, to apply the described generalization process to a great amount of data very similar to Thematic Units. In a more long-term prospect, this should entail the application of the complete MLK system on data automatically extracted from large volumes of texts.

References

1. Raymond Mooney and Gerald DeJong. Learning schemata for natural language processing. In Proceedings of the Ninth International Joint Conference on Artificial Intelligence, Los Angeles, 1985.
2. Michael J. Pazzani. Integrating explanation-based and empirical learning methods in OCCAM. In Proceedings of the Third European Working Session on Learning, Glasgow, 1988.
3. Ashwin Ram. Indexing, elaboration and refinement: incremental learning of explanatory cases. Machine Learning, 10(3):7-54, 1993.
4. Olivier Ferret and Brigitte Grau. An Aggregation Procedure for Building Episodic Memory, 15th IJCAI, Nagoya, 1997.
5. Olivier Ferret et Brigitte Grau, Construire une mémoire épisodique à partir de textes : pourquoi et comment ?, RIA, vol.12, n°3, pp. 317-409, 1998.
6. John F. Sowa. Conceptual Structures: Information Processing in Mind and Machine. Addison Wesley, 1984.
7. Guy W. Mineau. Induction on Conceptual Graphs: Finding Common Generalizations and Compatible Projections. Conceptual Structures - current research and practice. Eds. T. E. Nagle, J. A. Nagle, L. L. Gerholz and P. W. Eklund, New York, Ellis Horwood, 1992.
8. Isabelle Bournaud. Construction de hiérarchies conceptuelles pour l'organisation de connaissances. LMO'96, 1996.
9. Gerard Ellis. Compiled Hierarchical Retrieval. Conceptual Structures - current research and practice. Eds. T. E. Nagle, J. A. Nagle, L. L. Gerholz and P. W. Eklund, New York, Ellis Horwood, 1992.
10. K. B. Michalski and R. E. Stepp. Learning from observation: Conceptual clustering. In Machine Learning, An Artificial Approach, Vol. 1, pp. 331-363, Morgan Kaufmann, 1983.
11. Marc Champesme. Using Empirical Subsumption to Reduce Search Space in Learning. International Conference on Conceptual Structures, ICCS'95. Santa Cruz, California, USA, 1995.
12. Karim Chibout and Anne Vilnat, Computational Processing of Verbal Polysemy with Conceptual Structures, Acts Conceptual Structures: Theory, Tools and Applications (6th ICCS), Montpellier, LNCS volume 1453, p. 367-374, 1998.
13. Stephen Muggleton et Luc De Raedt, Inductive Logic Programming: Theory and Methods. Journal of Logic Programming 1994 :19,20 :629-679.14. Esra Erdem et Pierre Flener. A Re-Definition of Least Generalizations, and Construction Modes as a New Declarative Bias for ILP, Unpublished, 1998, http://www.cs.utexas.edu/users/esra/papers/ErdFle98.ps.

Author Index

Baader, Franz 292
Baget, Jean-François 277
Baldwin, James F. 193
Buche, Patrice 207
Burmeister, Peter 385

Chalendar, Gaël de 555
Chaudron, Laurent 357
Chein, Michel 127
Cole, Richard 438
Compton, Paul 113
Corby, Olivier 468

Dau, Frithjof 263
Devlin, Keith 1
Dieng, Rose 468

Eklund, Peter W. 399, 453

Ferré, Sébastien 371
Ferret, Olivier 555

Ganter, Bernhard 342, 483
Gatzemeier, Felix H. 27
Genest, David 127
Gerbé, Olivier 157
Grau, Brigitte 555
Groh, Bernd 332, 453

Hébert, Cédric 468
Haemmerlé, Ollivier 207
Harras, Gisela 13
Hereth, Joachim 421
Holzer, Richard 385

Janta-Polczynski, Martin 540

Kabbaj, Adil 540
Keeler, Mary 82
Keller, Rudolf K. 157
Kocura, Pavel 235
Kuznetsov, Sergei O. 342

Levinson, Robert 496

Maille, Nicolas 357
Martin, Philippe 41
Martin, Trevor P. 193
McGuinness, Deborah L. 100
Mineau, Guy W. 140, 157, 221
Molitor, Ralf 292
Moor, Aldo de 525
Mugnier, Marie-Laure 172

Pollandt, Silke 306
Prediger, Susanne 249

Ridoux, Olivier 371

Shapiro, Stuart C. 510
Sowa, John F. 55
Stumme, Gerd 421, 438, 453
Suryanto, Hendra 113

Tzanavari, Aimilia 193

Wille, Rudolf 306, 317, 332, 421, 453
Wille, Uta 421