Frithjof Dau, Marie-Laure Mugnier, Gerd Stumme (Eds.)

Common Semantics for Sharing Knowledge:

Contributions to ICCS 2005 13th International Conference on Conecptual Structures, ICCS 2005

Kassel, Germany, July 2005

kassel university press

Editors
Frithjof Dau
Technische Universität Darmstadt
FB Mathematik, AG 1
Schloßgartenstr. 7
D-64289 Darmstadt
Germany

Email: dau@mathematik.tu-darmstadt.de
Web: http://www.mathematik.tu-darmstadt.de/~dau
Tel: (++49) (0)6151-163095
Fax: (++49) (0)6151-163317

Marie-Laure Mugnier
LIRMM
(CNRS & University of Montpellier II)
161, rue ADA
34392 Montpellier Cedex 5
France

Email: mugnier@lirmm.fr
Web: http://www.lirmm.fr/~mugnier/
Tel: (++33) (0)67 41 85 39
Fax: (++33) (0)67 41 85 00

Gerd Stumme
Universität Kassel
Fachbereich Mathematik/Informatik
Fachgebiet Wissensverarbeitung
Wilhelmshöher Allee 73
34121 Kassel
Germany

Email: stumme@cs.uni-kassel.de
Web: http://www.kde.cs.uni-kassel.de/stumme
Tel.: (++49) (0)561 804-6251
Fax.: (++49) (0)561 804-6259

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

ISBN 3-89958-138-5
URN urn:nbn:de:0002-1383

Cover: Bettina Brand Grafikdesign, München
Printed in Germany by Unidruckerei der Universität Kassel

Preface

The 13th International Conference on Conceptual Structures (ICCS 2005) will be held in Kassel, Germany, during July 17–22, 2005. Information about the conference can be found at `http://www.kde.cs.uni-kassel.de/conf/iccs05` .

This volume consists of contributions complementing the main proceedings of ICCS 2005, entitled"Common Semantics for Sharing Knowledge" (F. Dau, M.-L. Mugnier and G. Stumme (Eds.), LNAI 3596, Springer, 2005).

The title "Common Semantics for Sharing Knowledge" of this year's conference was chosen to emphasize one hand the overall aim of any knowledge representation formalism: to support the sharing of knowledge; and on the other hand the importance of a common semantics to avoid distortion of the meaning. We understand that both aspects are of equal importance for a successful future of the research area of conceptual structures within the next years. We are thus happy that the papers presented at ICCS 2005 address both applications and theoretical foundations.

"Sharing knowledge" can also be understood in a separate sense. Thanks to the German Research Foundation DFG we were able to invite nine internationally renowned researchers from adjacent research areas. We are looking forward to stimulating presentations and lively discussions, with the hope of bidirectional knowledge sharing. Eventually even the ground can be laid for establishing common semantics between the respective theories.

We wish to express our appreciation to all the authors of submitted papers, to the members of the editorial board and the program committee, and to the external reviewers for making ICCS 2005 a valuable contribution to the knowledge processing research field.

July 2005

Frithjof Dau
Marie-Laure Mugnier
Gerd Stumme

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), at the University of Maryland (1994), at the University of California (Santa Cruz, 1995), at Sydney (1996), at the University of Washington (Seattle, 1997), at the University of Montpellier (1998), at Virginia Tech (Blacksburg, 1999), at Darmstadt University of Technology (2000), at Stanford University (2001), at Borovets, Bulgaria (2002), at Dresden University of Technology (2003), and at the University of Alabama (Huntsville, 2004).

General Chair

Gerd Stumme	University of Kassel, Germany

Program Chairs

Frithjof Dau	Darmstadt Technical University, Germany
Marie-Laure Mugnier	University of Montpellier, France

Editorial Board

Galia Angelova (Bulgaria)
Michel Chein (France)
Aldo de Moor (Belgium)
Harry Delugach (U. S. A.)
Peter Eklund (Australia)
Bernhard Ganter(Germany)
Mary Keeler (U. S. A.)
Sergei Kuznetsov (Russia)
Wilfried Lex (Germany)
Guy Mineau (Canada)
Bernard Moulin (Canada)
Peter Øhrstrøm (Denmark)
Heather Pfeiffer (U. S. A.)
Uta Priss (U. K.)
John Sowa (U. S. A.)
Rudolf Wille (Germany)
Karl Erich Wolff (Germany)

Program Committee

Anne Berry (France)
Tru Cao (Vietnam)
Dan Corbett (Australia)
Olivier Corby (France)
Pavlin Dobrev (Bulgaria)
David Genest (France)
Ollivier Haemmerlé (France)
Roger Hartley (U. S. A.)
Udo Hebisch (Germany)
Joachim Hereth Correia (Germany)

Richard Hill (U. K.)
Pascal Hitzler (Germany)
Kees Hoede (The Netherlands)
Julia Klinger (Germany)
Pavel Kocura (U. K.)
Robert Kremer (Canada)
Leonhard Kwuida (Germany)
M. Leclère (France)
Robert Levinson (U. S. A.)
Michel Liquière (France)
Carsten Lutz (Germany)
Philippe Martin (Australia)
Engelbert Mephu Nguifo (France)
Sergei Obiedkov (Russia)
Simon Polovina (U. K.)
Anne-Marie Rassinoux (Switzerland)
Gary Richmond (U. S. A.)
Olivier Ridoux (France)
Daniel Rochowiak (U. S. A.)
Sebastian Rudolph (Germany)
Eric Salvat (France)
Janos Sarbo (the Netherlands)
Henrik Schaerfe (Denmark)
Thanwadee T. Sunetnanta (Thailand)
William Tepfenhart (U. S. A.)
Petko Valtchev (Canada)
Sergei Yevtushenko (Germany)
GQ Zhang (U. S. A.)

External Reviewers

Sadok Ben Yahia (Tunisia)
Richard Cole (Australia)
Jon Ducrou (Australia)
Letha Etzkorn (U. S. A.)
Markus Krötzsch (Germany)
Boris Motik (Germany)
Anthony K. Seda (Ireland)

Table of Contents

Full Papers

Position Papers

Detecting Knowledge Flows in Weblogs

Anjo Anjewierden[1], Robert de Hoog[2], Rogier Brussee[3], and Lilia Efimova[3]

[1] Human Computer Studies Laboratory, University of Amsterdam, Kruislaan 419, 1098 VA Amsterdam, The Netherlands, anjo@science.uva.nl
[2] Faculty of Behavioural Science, University of Twente, PO Box 217, 7500 AE Enschede, The Netherlands, R.deHoog@edte.utwente.nl
[3] Telematica Instituut, PO Box 589, 7500 AN Enschede, The Netherlands, {Rogier.Brussee,Lilia.Efimova}@telin.nl

Abstract. Scanning the internal and external environment of an organisation to trace current and emerging topics of discourse, is an important task for knowledge management. Results of this scanning process can be used to anticipate on new developments and for detecting possible knowledge bottlenecks. This paper addresses the unobtrusive identification of conceptual structures exchanged and possibly shared in weblogs as a case to explore the opportunities for automated support for this scanning process. The research is also motivated by the observation that many domains defy formal conceptualisation in the sense that professionals, in particular knowledge workers, differ on the meaning of the concepts and terminology and the relations between them. In such domains, we use the field of knowledge management itself as an example, there is an insufficient basis for conceptual formalisation, the best we can do is finding out whether professionals agree and share, or disagree.
Informally, we define a *knowledge flow* to be the communication of some knowledge, in our case represented as a natural language post on a weblog to a receiver, a reader of the post in question. Since we cannot detect if people read a post, we will only consider those receivers that acknowledge reading by referring back to the sender in a separate post. The research challenges we pursue are to determine what knowledge is being exchanged and to what degree the participants in the communication share conceptualisations.
We have implemented the identification of knowledge flows in a tool called BlogTrace. BlogTrace uses semantic web technology to represent weblogs and linkage structures, term extraction to determine concepts in natural language posts, and a statistically founded algorithm which computes a relevance metric for the concepts extracted and their relation to other concepts for any particular weblog author.
Based on several assumptions, software was written that had to answer knowledge management relevant questions concerning the frequency, intensity and topics of knowledge flows in a community of bloggers, which act like members of an actual organisation. The software was tested on a set of questions and the results indicate that this can provide hints on what topics and to what extent people are exchanging knowledge.

1 Introduction

Professional knowledge workers, for example in the areas of law, journalism, computing and knowledge management, are increasingly using weblogs (blogs for short) to exchange and share information in their area of expertise. Blogs provide a low cost, easy to use, and more or less informal webpublishing platform. Although blogs are often viewed as “personal journals”, and have been compared to diaries, knowledge workers succeed in using them as an effective tool to create *spontaneous* virtual communities in which observations are passed on, questions are answered and issues discussed.

A weblog can be viewed as a collection of posts. Each post has a precise hyperlink on the web called the permalink, a title, and a publication date. The permalink, which is a simple URL, makes it possible for other bloggers to link to a post and thereby comment or reflect. Posts are generally self contained and permalinks are essential, technically and socially, to keep the blogosphere together.

One of our research interests is in conceptualisations. More specifically: can we derive conceptualisations from (large) bodies of natural language text? Weblogs provide an ideal data source for doing this kind of research, because by the nature of blogs we know they are written by a single person. This provides an opportunity to derive personal conceptualisations. By comparing conceptualisations of different bloggers we can therfore obtain a metric of overlap and, implicitly, difference. From a broader, application oriented, perspective the ability to derive and compare conceptualisations from (semi)public sources in an unobtrusive way, can be very useful for organisations that have to scan the internal and external environment for tracing and detecting current and emerging topics of discourse. In an actual organisation the involvement of individuals in this discourse can be of help for knowledge management to stay informed about the waxing and waning of topics of interest that arouse the attention of its knowledge workers.

In this paper, we propose the idea of a *knowledge flow*. In the context of weblogs a knowledge flow could take place when a post of one blogger links back to a post of another blogger. Such a link could, of course, be accidental or trivial ("Have a look at ..."). It could also be the instance of a genuine exchange of knowledge between the bloggers involved. Our hypothesis is that the likelihood of the latter is larger when the overlap of conceptualisations between the bloggers is present in the terms used in the linked posts. This hypothesis is an intuitive operationalisation of the assumption that using terms in combination frequently and knowing what they mean are closely related.

We have implemented the idea of detecting knowledge flows in a tool called BlogTrace. With the implementation of the analysis of knowledge flows it becomes possible to retrieve pairs of linked documents that are about a similar topic.

The paper is organised as follows. Sect. 2 provides a theoretical background for the idea of knowledge flows. Sect. 3 describes our approach to automatically extract conceptualisations from text documents and relates it to formal concept analysis. Sect. 4 presents the RDF/OWL based architecture to represent weblog data and, finally, Sect. 5 discusses the results of applying the knowledge flow idea based on the tools and methods described in the previous sections.

2 Communicating Knowledge, Action and Context

Basically, a knowledge flow is a specific type of information flow. From communication science [8], [4] we can borrow the sender-message-receiver model to describe these flows. The sender and receiver are points in a communication network, which exchange messages. Who the sender and who receiver is, depends on who takes the initiative. The sender-message-receiver model can be rephrased by the following three questions:

- Who says
- What
- To whom

In its most simple form detecting and analysing knowledge flows entails identifying the "Who" and "Whom" and the nature of the "What".

For knowledge flows in weblogs, the easy part is the "who". By definition it is the person who is the author of the weblog.

The "What" is much trickier. The problem is that for human communication many "messages" can be understood only if the implicit or explicit context is taken into account. This context can be the author's physical, cultural or economical environment. More mundanely it consists of previous anecdotes, people the author has been in contact with or the web pages read. Thus determining which context the message is addressing is an important part of determining the content of the message. Fortunately, the publicness of the blog medium and the corresponding potentially large and diverse audience forces people to be fairly explicit about their context.

Moreover using links and copy paste makes it easy to make the context clear if, as is often the case, this context is other blogs or other web content.

The "to whom part" is also subtle. Many people can read a weblog and be influenced by it. Existing tools for tracing weblog readership paints a partial picture and are usually only available to the weblog author. Therefore, for the purpose of this paper, we only consider knowledge flows in a communication pattern where the initial post is followed by posts by another blogger who links back.

There is a second, more philosophical (and debatable) justification for this approach: we want to distinguish between a "mere" information flow and a knowledge flow. Obviously this depends on what we mean with "knowledge" and how we can distinguish it from "information". We take the point of view taken by [6], p. 37 and others

Knowledge is a capacity to act

This definition implies that the "What" is determined by what (if anything) the receiver *does* with the message in addition to its content. The one visible action that we can determine is what a reader puts in his or her blog.

From a practical perspective analysing knowledge flows could be useful in an organisational context, for example for:

- Monitoring the *frequency* and *intensity* of crucial flows between people in and outside the organisation (the "Who" and "Whom").
- Evaluating the content of the crucial flows between organisational entities to detect *bottlenecks* and emerging *problems/topics* (the "What").
- Monitoring the development of *flows over time* to keep track of developments in the knowledge inside the organisation.

These general problems can be made more specific as queries to be directed at a set of weblogs as illustrated in Sect. 5.

3 Extracting Conceptualisations from Text

In order to determine what the content is of the information or the knowledge that flows between the sender and receiver of the messages, we need to understand how the symbolic information in messages relates to the real world. Unfortunately, we have limited access to this world. Thus we think of the content of the messages as a representation of the real, message external, world, *a conceptualisation*. The flow of knowledge is then the way the conceptualisation of the message sender is serialised in the conceptualisation in the message and the way the conceptualisation of the receiver is build up from the conceptualisation in the message. This process is lossy.

Over the last decades there has been a significant amount of research into capturing conceptualisations with an emphasis on using formal and machine inspectable representations. Under the flag of ontologies ([5]) and the semantic web, it has captured a wider audience and led to knowledge representation languages such as RDFS and OWL. The approach aims to formalise structures that are believed to exist in some abstract sense in the real world, thereby making classification and inferencing possible.

In view of the above our aim is not to discover or formalise relations between concepts but to discover how bloggers (or other writers) describe "the world" without any *a priori* claim on the validity of those descriptions. The approach settles on finding patterns in the traces bloggers leave through their use of terms and linkage, and doing relatively crude statistical analysis of correlations. We assume that those correlations are a result of bloggers using an underlying conceptualisation. We *do not* assume that this conceptualisation is a faithful representation of the world, that it is shared between bloggers, or that the discovered correlations are the underlying conceptualisation themselves.

If we accept that web presence is part of the "real world" we have at our disposal links and the use of orthographical equivalences of terms. In summary, our approach is as follows:

1. Identify terms and links that potentially point to concepts. Here we make a distinction between terms that point to names of people and terms that represent the subject matter of the text. See also Sect. 3.1.

2. Once the concepts have been identified, we need to establish whether they are semantically related, at least according to a blogger. For this we rely on the assumption that if there is some sort of semantic relation between terms, these terms are often used together in the same post. As an operationalisation of the strength of the relation we use a statistical measure for the "risk" (co-occurrence) of using one term given that blogger is using another term in the same post. See Sect. 3.2.

3. The output of the above analysis is a two dimensional table of co-occurrences for using terms in combination. A typical weblog contains thousands of terms, so the table contains millions of entries. We then select the high co-occurrence combinations and graphically represent the resulting clusters of terms for the end-user. Preliminary experience shows that visual inspection and human experience often suggest an underlying semantic relation between terms. See Sect. 3.3 for the methodology and Sect. 5 for examples.

3.1 Term and Name Extraction

This section briefly describes how we extract terms and names from weblog posts. A more detailed account can be found in [1]. The main challenges with respect to language technology are the identification of meaningful terms, the extraction of the names of people, and expanding abbreviations to their long form.

We define a meaningful term to be a (possibly compound) term that refers to a single concept irrespective of the specific textual rendering. The (semantic) term class of a meaningful term is defined as the set (equivalence class) of all terms referring to the same concept and is denoted with square brackets around the term. The first task is therefore to find meaningful terms and the second problem is to collect the terms that belong to the same term class. Often a meaningful term corresponds linguistically to a sequence of nouns. Because of the definition of a term class, the meaningful terms KM and knowledge management are both members of the same term class [knowledge management] as they refer to the same concept (provided of course that KM is an abbreviation for knowledge management in a given weblog). Similarly, inflected forms (e.g. plurals, past tense), misspellings, alternate spellings and user provided synonyms are also treated as members of a term class. The analysis of a weblog proceeds as follows:

1. Identify potential terms. The algorithm scans over the posts and collects all sequences of words separated by stop words. For example, the sentence: "This is knowledge management research ...," results in the following potential meaningful terms being recorded: *knowledge*, *management*, *research*, *knowledge management*, *management research* and *knowledge management research*. These terms are then normalised using the CELEX dictionary [2], for example *supporting informal learning* gives rise to the term class [support informal learn].

2. Expand abbreviations. The second step in processing a weblog is expanding the short forms of abbreviations to their corresponding long form. Because of the noisy nature of weblogs, traditional abbreviation finding algorithms that rely on the short and long forms appearing next to each other do not work. The algorithm we use is based on the idea that the long form must be a meaningful term and that both the long and the short forms appear relatively frequent. A stop list of very common abbreviations (e.g. PC, CD, OS, etc.) is used to prevent accidental expansions.

3. Delete implied and low frequency terms. The next step is to delete all terms that are implied by longer terms. For example, if all occurrences of *management research* are part of *knowledge management research* then the former is redundant and can thus be ignored. A term has to appear at least four times in a given weblog to be considered for analysis.

4. Names of people are recognised using a gazetteer of first names based on the one in GATE[4] to which we added all names of players from the list published by international chess federation to get a better world-wide coverage of names and a gazetteer with person name prepositions such as *de* (Robert de Hoog) and *bin* (Osama bin Laden). The algorithm also disambiguates names if only the first name is used, a frequent practice in weblogs. This is partially based on names occurring in the anchor text of a link.

3.2 Co-occurrence Analysis and Concept Structures

Intuitively a term or link B co-occurs with a term or link A if the frequency of A in posts containing B is much higher than the frequency of term B in posts not containing term A. This is not symmetric in A and B: for example in a document on management, the term *knowledge* could co-occur with *management*if the phrase knowledge management would be an important source of occurrences of the word knowledge, but if the document contains many uses of the term management alone or in phrases like customer relation management, management would not co-occur with knowledge. We quantify co-occurrence using the following statistics.

Definition: Let $n(B \mid A)$ (respectively $n(B \mid \neg A)$) be the number of occurrences of the term B in posts that contain the term A (respectively do not contain the term A), and likewise let $n(* \mid A)$ (respectively $n(* \mid \neg A)$) be the total number of terms in the posts that contain the term A (respectively do not contain the term A). Then the **co-occurrence degree** $c(B \mid A)$ is defined as

$$c(B \mid A) = \frac{n(B \mid A)/n(* \mid A)}{n(B \mid \neg A)/n(* \mid \neg A)}, \quad 0 \leq c(B \mid A) \leq \infty$$

We say that B co-occurs with A to at least degree k if $c(B \mid A) \geq k$. Note that $c(B \mid A) = 1$ if B is as frequent in posts containing as it is in posts not containing A, i.e. that term B and A seem to be unrelated. See [1] (section 3.3) for a discussion of the statistical variance of the co-occurrence measure.

The co-occurrence matrices can be thought of as a statistical version of the formal concept lattice [3] defined by the "occurs in" relation between terms as intent and posts as extent. Consider a (large) set of posts $\{P_1, P_2, \ldots P_N\} = \{P_i\}_{i \in I}$, and the subsets of posts (or paragraphs) $\{P_a, P_b, P_c \ldots\} = \{P_i\}_{i \in I_A}$ containing A respectively $\{P_x, P_y, P_z, \ldots\} = \{P_j\}_{j \in I_B}$ containing B. If $\{P_j\}_{j \in I_B}$ is *not* a subset of $\{P_i\}_{i \in I_A}$, there are two formal concepts $C_A = \langle A \mid P_{i i \in I_A} \rangle$ and $C_B = \langle B \mid P_{i i \in I_B} \rangle$. Otherwise C_B does not exist as a seperate concept, but only the subconcept $C_{AB} = \langle AB \mid P_{i i \in I_A \cap I_B} \rangle$ exists (similar statements hold with A and B interchanged). However this is a rather strict point of view. If statistically B cooccurs with A much more often then it does not, then statistically speaking we should consider the concepts C_B and C_{AB} as equivalent. The co-occurrence $c(B \mid A)$ is a quantitative measure for this idea. It is the fraction of posts of C_A "in" C_{AB} divided by the fraction of posts of $C_\emptyset - C_A$ "in" $C_B - C_{AB}$. In particular, if $c(B|A) = \infty$, C_B does strictly not exist as a separate formal concept.

The previous discussion addresses only the top layer

$$\begin{array}{ccc} & \langle \emptyset \mid \{P_i\}_{i \in I} \rangle & \\ & / \quad \backslash & \\ \langle A \mid \{P_i\}_{i \in I_A} \rangle & & \langle B \mid \{P_i\}_{i \in I_B} \rangle \\ & \backslash \quad / & \\ & \langle AB \mid \{P_i\}_{i \in I_A \cap I_B} \rangle & \end{array}$$

of the formal concept lattice. Although we have not pursued this, we can strengthen the interpretation of the co-occurrence matrix as a statistical version of a concept lattice as follows. Define $n(B_1 B_2 \ldots \mid A_1 A_2 \ldots)$ as the number of occurences of the *collection* of terms $B_1, B_2, \ldots$ in posts containing the collection of terms $A_1, A_2, \ldots$ and define the co-occurrence degrees

[4] http://gate.ac.uk

$c(B_1B_2 \ldots \mid A_1A_2 \ldots)$ similarly as above. Again $c(B_1B_2 \ldots \mid A_1A_2 \ldots) = \infty$ means that there is no separate concept $\langle B_1B_2 \ldots \mid \ldots \rangle$ but only a formal subconcept $\langle B_1B_2 \ldots A_1A_2 \ldots \mid \ldots \rangle$ of the concept $\langle A_1A_2 \ldots \mid \ldots \rangle$.

3.3 Detecting Knowledge Flows

A potential knowledge flow occurs when, in a weblog post, one blogger includes a hyperlink to a weblog post of another blogger. We hypothesize that the intensity of a knowledge flow between two linked posts depends on the terms used in these posts and the conceptualisations of both bloggers as determined by the co-occurrence algorithm described in the previous section.

The operationalisation for the knowledge flow intensity is based on the idea of comparing the co-occurrence degree $c(B \mid A)$ of the terms appearing in the linked posts. To bootstrap the process we assume that the knowledge flow search process is initiated with a cue term A.

Cue term. The cue term A must appear in both posts.

Shared terms. We define a shared term to be a term B that occurs in both posts and for which both bloggers have a co-occurrence degree k such that $c(B \mid A) \geq k$. The informal interpretation of a shared term is that both bloggers associate it with the cue term on the basis of the entire blog.

Agreed terms. A term B that occurs in both posts and for which one blogger has a co-occurrence degree k such that $c(B \mid A) \geq k$.

Private terms. A term B that occurs in precisely one of the posts and for which $c(B \mid A) \geq k$ holds for the blogger of the post.

Qualifying terms as shared, agreed and private is a first step towards finding knowledge bottlenecks and potential conflicts. Looking at terms each blogger frequently relates to the cue term provides a (crude) way to access how far the meaning of the term is shared as illustrated in Sect. 5.

4 Data Acquisition and Representation

Although data on which algorithms described in the previous section could be applied is widely available. Most data is not in a format suitable for processing. This section describes the main functions of BlogTrace. BlogTrace is a software environment that defines ontologies to represent links between documents, documents themselves and linguistic information that can be used by the co-occurrence and knowledge flow algorithms.

A lot of understanding of weblogs comes from bloggers themselves often with generalisations based on personal experiences, observed blogging practices or an analysis of unrepresentative samples. However, for a researcher looking at the blogging phenomenon it is especially important to understand if it is representative for weblogs as a whole, for a specific blogging subculture or just for an individual blogger. This points to a need for a software environment that accomodates blog research. BlogTrace is such an environment.

We will briefly describe the architecture of BlogTrace (see Fig. 1). The most critical aspect is turning the HTML pages that represent a weblog into a more suitable representation through spidering and mining. This process consists of extracting metadata (name of the blogger, title of the blog), identifying the parts of the HTML that are posts and turning the post attributes (title, permalink URL, date and body) into metadata. Although weblogs are generated by software, there is a hugh variety in the layouts used. Our approach to the spidering and mining problem is to run induction algorithms on the HTML pages. These algorithms generate patterns (for example based on attributes in HTML elements and/or the position in the DOM-tree). A calibration algorithm then tries to find a stable set of patterns that fit the natural order of post attributes.

The output of the blog spider is represented as RDF using the semantic web library described in [7]. One file contains all metadata about the blog itself, all permalinks and all hyperlinks inside the posts. This data can be used to perform linkage and community type research at the blog level.

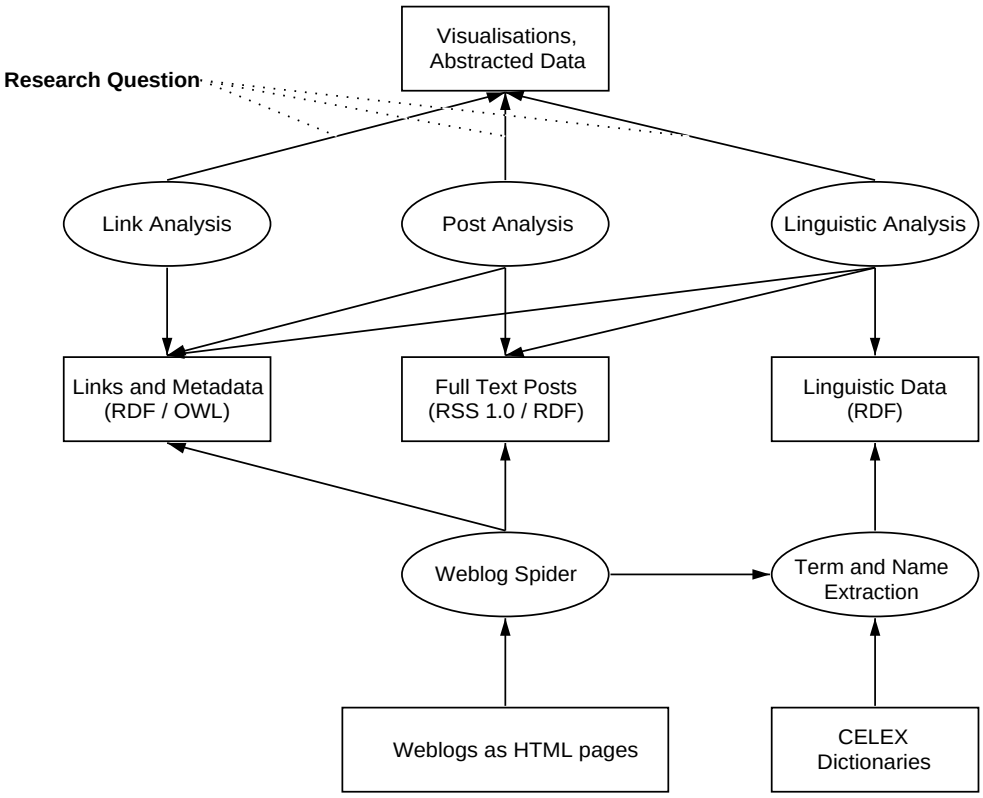


Fig. 1. Overall architecture of BlogTrace

The second file contains the full texts of the posts in RSS 1.0. This data can be used for analysis at the post level. The third file contains linguistic data abstracted at both the blog and post levels (terms, term frequency and in which posts these terms are used), see Sect. 3.1. The linguistic data supports the identification of conceptual structures, both for individual bloggers and for a community of bloggers and is fundamental for the knowledge flow analysis in this paper.

Although BlogTrace currently only works for weblogs, the spider looks for features only blogs have, there are no restrictions to apply it to other types of documents that have the same structure: text interspersed with links.

The metadata about a weblog is represented using class `docs:Weblog`, most properties refer to the Dublin Core[5] and FOAF[6] ontologies:

```
<docs:Weblog rdf:about="http://anjo.blogs.com/metis/"
  dc:creator="Anjo Anjewierden"
  dc:description="The source code is the ultimate documentation"
  dc:title="Anjo Anjewierden"
  rdfs:label="http://anjo.blogs.com/metis/"
  foaf:nick="Anjo Anjewierden">
  <docs:hasRssFeed rdf:resource="http://anjo.blogs.com/metis/rss.xml"/>
</docs:Weblog>
```

A hyperlink (HTML: `<a href="...")` is represented using class `link:SimpleLink` from a link ontology. The `link:sourceDocument` is the document in which the hyperlink was found and `link:targetDocument` is the document to which the hyperlink points.

```
<link:SimpleLink
    link:anchorText="city metaphor to explain blogging">
  <link:sourceDocument
```

[5] http://purl.org/dc/elements/1.1/
[6] http://xmlns.com/foaf/0.1/

```
    rdf:resource="http://anjo.blogs.com/metis/2004/06/city_metaphor_f.htm
  <link:targetDocument
     rdf:resource="http://blog.mathemagenic.com/2004/06/07.html"/>
</link:SimpleLink>
```

Posts are represented using `rss:item` from the RSS 1.0[7] ontology. The `link:SimpleLink` above was extracted from the post represented as RSS below.

```
<rss:item rdf:about="http://anjo.blogs.com/metis/2004/06/city_metaphor_f.
  <rss:title>City metaphor for blogging</rss:title>
  <rss:link>http://anjo.blogs.com/metis/2004/06/city_metaphor_f.html</rss
  <dc:date>2004-06-10</dc:date>
  <rss:description>
<p>Another train discussion we had was Lilia's
<a href="http://blog.mathemagenic.com/2004/06/07.html">city metaphor
to explain blogging</a> post.  [...]
  </rss:description>
</rss:item>
```

Using the above data generated by the spider we can then exploit OWL to define higher-level notions. For example, in order to identify a knowledge flow we require links from weblog posts to other posts which can be defined in OWL as follows (in N3 notation):

```
link:WeblogPostLink rdfs:subClassOf link:SimpleLink;
  rdfs:comment "A WeblogPostLink is a SimpleLink if and only if
                both the source and the target documents are
                weblog posts (RSS items)";
  owl:intersectionOf (link:SimpleLink
                      [ a owl:Restriction;
                          owl:onProperty link:sourceDocument;
                          owl:someValuesFrom rss:item
                      ]
                      [ a owl:Restriction;
                          owl:onProperty link:targetDocument;
                          owl:someValuesFrom rss:item
                      ]).
```

5 Experimental Results

The ideas and software described in the previous sections were tested on a collection of 35 weblogs of people who are loosely related through a common interest in knowledge management. As the main goal is not to detect existing or emerging communities, limiting the test to these is in line with the notion of scanning flows in an organisation. We describe the experimental results as questions a knowledge manager might pose and present the answers using screendumps from BlogTrace and the relation with earlier sections of the paper.

Question 1: What are the terms and conceptualisations bloggers use? Questions like this are addressed by the *Sigmund* module part of BlogTrace. Sigmund (Fig. 2) uses the term extraction techniques from Sect. 3. The second browser displays the persons referred to in a blog and the fourth browser the terms extracted and their frequency. At the bottom is a network that displays the shared conceptualisations of two bloggers. The network is derived from all term tuples A and B for which the co-occurrence degree $c(B \mid A) \geq k$ holds for both bloggers. Sigmund is described extensively in [1].

[7] http://purl.org/rss/1.0/

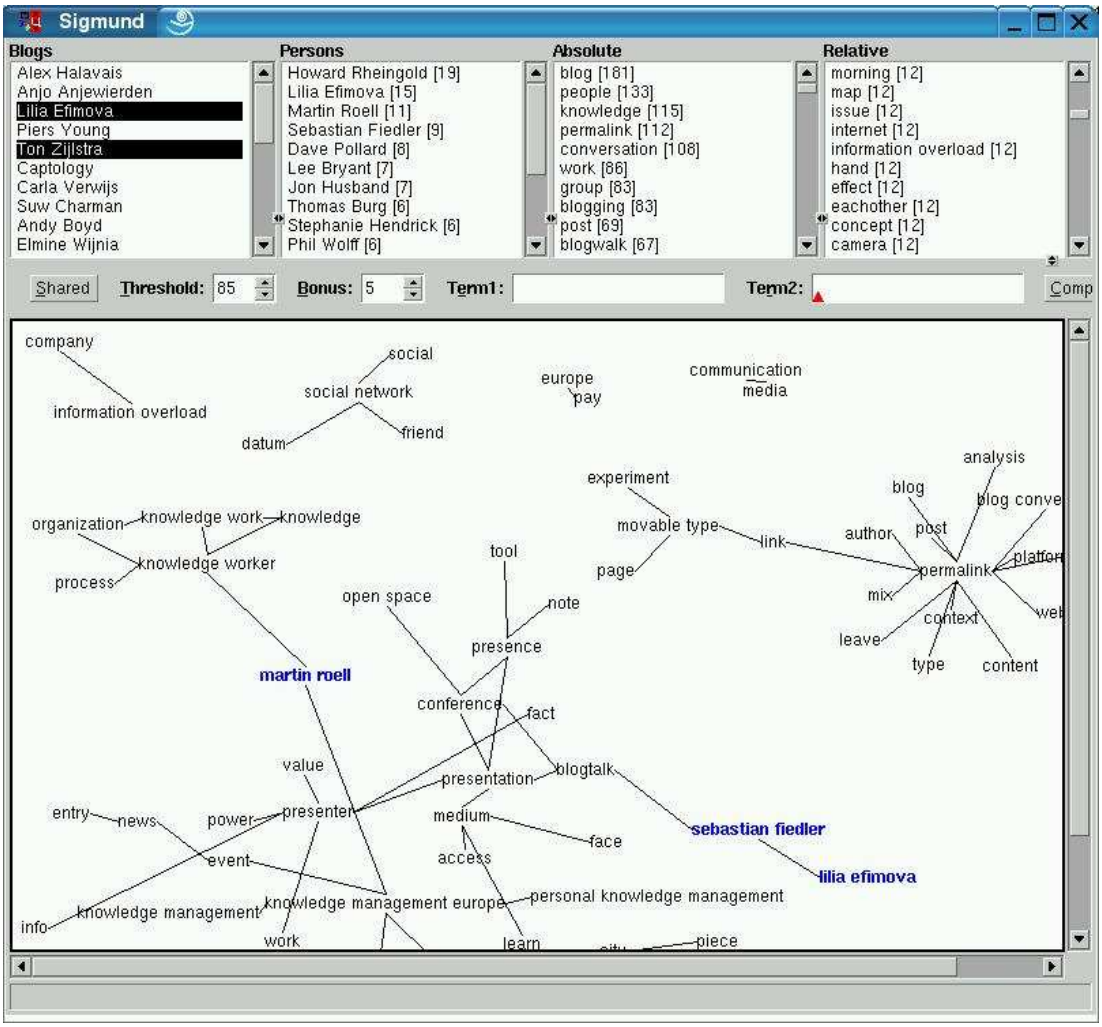


Fig. 2. Terms and conceptualisations

Question 2: Given a concept $C*$ find out which bloggers are using these concepts in their posts and to which other bloggers they have links concerning $C*$. The analysis by BlogTrace yields the results shown in Fig. 3. The figure displays (top left to top right): blogs, weblog posts inside a selected blog and links in a selected blog or post. The large area at the bottom is used to display results from end-user queries.

The concept $C*$ in this case is the term *information overload* entered in the search line in the middle part of the screen. This search term can be either free keywords or selected terms from a predefined vocabulary. The selected function is searching for *Knowledge flows*. The lower part of the window shows in a tabular form all the pairs of posts that are linked through *information overload*. The first part of the table shows the source and the target post and the weblogs to which they belong. The **Shared terms** (see Sect. 3.3 for the definition) are the terms both bloggers associate with *information overload*. The **Agreed terms** are terms the linked bloggers in these posts associate with *information overload* but do not belong to the shared terms. The **Source only** category contains terms only the source blogger associates with *information overload*. The same holds for the **Target only** category.

Question 3: Given an answer on Question 2, what is the amount of agreement or disagreement between bloggers about a concept? As was said already, initially we take the amount of term overlap as an indicator of the agreement and disagreement in a knowledge flow. Later we will try to analyse the content of the posts on specific terms that can indicate agreement or disagreement. The answer to this question is implicit in Fig. 3, but BlogTrace contains a convenient facility to

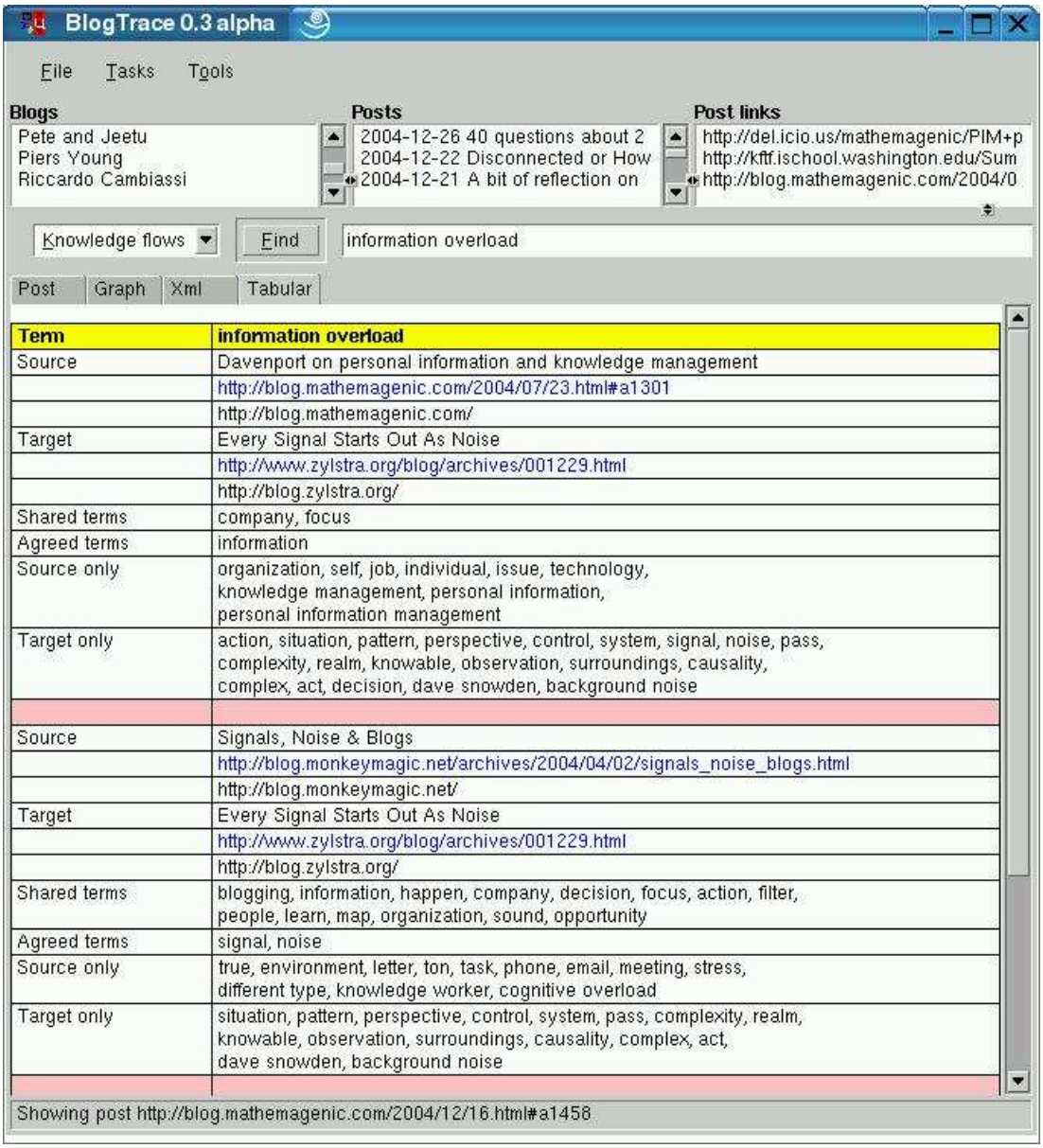


Fig. 3. Knowledge flows as terms

make this more easy to see. By selecting the *Kfbw sparklines* function after entering a search term, the tool presents the results using the idea of a *sparkline*[8].

In Fig. 4 the search term is *knowledge management*, to show that the software works for different concepts. Pairs of posts are depicted and the middle column gives a sparkline indicating the amount of overlap by using the following colour codes:

- **Shared terms** Shown in dark grey.
- **Agreed terms** Shown in light grey to the left and right of the shared terms.
- **Source/Target only** To either side of the shared/agreed terms in medium grey.

The smaller the dark and/or light grey bar the less overlap. The placement of the medium grey area gives an indication of the terms which are used only by one of the bloggers in their posts. By clicking on a post title both the linked posts are displayed in the lower window, enabling a closer inspection of their content.

The results described in this section show that BlogTrace can be used to analyse weblogs to detect knowledge fbws in general, but also concerning specific concepts, between bloggers

[8] http://www.edwardtufte.com/bboard/q-and-a-fetch-msg?msg_id=0001OR&topic_id=1&topic=

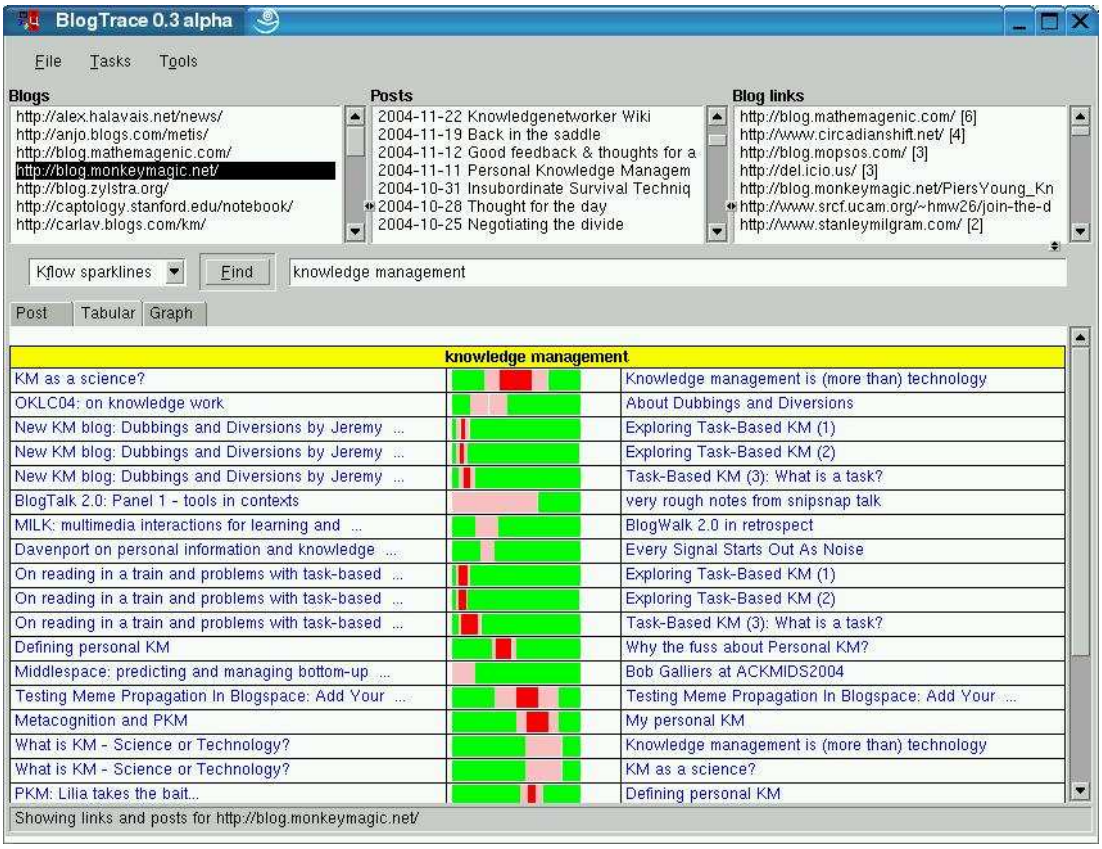


Fig. 4. Knowledge flows displayed as sparklines

through cross-referencing in their posts. From a knowledge management perspective these results can be used to find answers for the kind of areas of interest mentioned in Sect. 2: monitoring the frequency and intensity of flows between people in- and outside an organisation, evaluating the content of these flows to detect bottlenecks (who agrees/disagrees with whom concerning what) and emerging problems and topics (agreed and/or target/source only terms).

6 Conclusions

In this paper we have investigated whether shared conceptualisations can be extracted from weblogs. Weblogs can be seen as an instance of a wider class of publicly accessible information sources which are bound to a specific individual. The notion of a knowledge flow was introduced to identify mutual concept dependent linkages between individual bloggers. Based on several assumptions, statistical methods and software was developed that had to answer knowledge management relevant questions concerning the frequency, intensity and topics of knowledge flows in a community of bloggers, which act like members of an actual organisation.

The software was tested on a set of example queries related to questions one might have about knowledge flows. During this process at least two important problems were detected at the technical level that need a solution in the future. **Quoting**. Quotes are used frequently in weblog posts. At the moment these quotes become part of the vocabulary of the quoter and thus incorrectly influence the conceptualisations derived. **Abbreviations**. The current method of processing weblogs cannot detect that in two *different* blogs shared abbreviations are used. As several concepts are often expressed as an abbreviation (for example PKM for “personal knowledge management”), important concepts can be missed.

Of course, the work reported here is only the start of a process that can lead to software that has a richer potential for analysing knowledge flows between weblogs and other types of documents. The following issues will be taken up:

1. The software does not make extensive use of natural language analysis. This limits the power of the analysis. On the other hand, full blown natural language analysis of weblogs is not the overall goal of the research. There should be some middle ground in which certain aspects of natural language processing can be used to improve the concept detection process and also the ability to identify agreement and disagreement at a semantically richer level than term overlap alone.

2. There are obviously other ways for individual weblogs to share conceptualisations than through explicit linking. One such way is a kind of indirect sharing when two bloggers don't link to each other but both refer to the same external link. An exhaustive list of direct and indirect referencing tactics used by bloggers shows that these can enhance the ability of the software to detect and analyse knowledge fbws. A related issue is that of *weblog conversations* in which multiple bloggers link over time.

As a caveat one should keep in mind that the current analysis process is based on several assumptions that can be invalidated in the future.

Acknowledgements. This work was partly supported by the Metis project[9]. Metis is an initiative of the Telematica Instituut, Basell and Océ. Other partners are CeTIM, Technical University Delft, University of Amsterdam, University of Tilburg and University of Twente (all in The Netherlands).

References

1. Anjo Anjewierden, Rogier Brussee, and Lilia Efimova. Shared conceptualizations in weblogs. In Thomas N. Burg, editor, *BlogTalks 2.0: The European Conference on Weblogs (July 2004)*, pages 110–138, Vienna, February 2005. Danube University of Krems.
2. R.H. Baayen, Richard Piepenbrock, and Léon Gulikers. The CELEX lexical database (release 2) [CD-ROM]. Philadelphia, PA: Linguistic Data Consortium, University of Pennsylvania, 1995.
3. Bernhard Ganter and Rudolf Wille. *Formal concept analysis. Mathematical foundations*. Springer-Verlag, 1999.
4. Claude A. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 623–656, 1948.
5. Stefan Staab and Rudi Studer, editors. *Handbook on ontologies*. Springer-Verlag, 2004.
6. Karl-Erik Sveiby. *The new organizational wealth: Managing and Measuring Knowledge-Based Assets*. Berrett Koehler, 1997.
7. Jan Wielemaker, Guus Schreiber, and Bob Wielinga. Prolog-based infrastructure for RDF: Scalability and performance. In Dieter Fensel, Katia Sycara, and John Mylopoulos, editors, *2nd International Semantic Web Conference (ISWC)*, Sanibel Island, FL, USA, October 2003.
8. Sven Windahl, Benno Signitzer, and Jean T. Olson. *Using communication theory: An introduction to planned communication*. Sage Publications, 1992.

[9] http://metis.telin.nl

A Lattice Representation of Relations, Multivalued Dependencies and Armstrong Relations

Jaume Baixeries[1] and José Luis Balcázar[1]

Dept. Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya
c/ Jordi Girona, 1-3
08034 Barcelona
{jbaixer, balqui}@lsi.upc.edu

Abstract. We present a lattice-based formalism to relate, in a novel way, different representation methods for relational data. Specifically, relations given extensionally by tuples, multivalued dependencies, and Armstrong relations for multivalued dependencies. The semantics of this formalism is based on a closure operator used to calculate the lattice. We prove that this representation yields the set of multivalued dependencies that hold in a set of tuples as well as an Armstrong relation.

1 Introduction

Data, in a broad sense, can be represented in many different manners: sets of tuples, constraints (known as database dependencies or implications, depending on the context), first-order logic formulæ, Armstrong relations, a large variety of data structures, etc. Lattices have also been used to represent knowledge of structured data, being Formal Concept Analysis (FCA) one of the main examples of applications of this kind. There have been different attempts to relate those formalisms one to each other, and we draw our attention specifically to links between lattices and different kinds of constraints and Armstrong relations, noting that, in most cases, a lattice is not explicitly exposed, but implicitly as a set of closed sets (the reader can refer to [8] for a more detailed explanation of the equivalence of closure systems and lattices).

Let us consider relationships between lattices and implications, that have been proved in [17], and also in [24]. In [2], the authors show the relationship between the lattice representation for a set of binary data and a Horn approximation for this same set of data. Relationships between lattices and functional dependencies (FD) have been shown in different manners in [5], [7], [17], [18], [22] and [29], where *agree sets*, *maximal sets* and *closed sets* are, in fact, representations of lattices. In [12] the authors define a lattice so that FD's can be derived from this representation using a closure operator that has a set of FD's as an input. In [1] the authors define a closure operator (based on the previous work in [21]) to calculate functional dependencies with a lattice, based on the

set of tuples, instead of a set of functional dependencies. Applications of lattice representations of FD's appear in [11], where the authors prove that such a lattice detects whether a relation is in second, third or Boyce-Codd Normal Form, which is a key factor in database design.

Some authors use a lattice approach in order to find an Armstrong relation for a given set of constraints. In [15] closed sets are used to create an Armstrong relation for a set of FD's, which, in fact, constitute a lattice, although the author does not explicitly mention it. In [19] the authors use concepts closely related to closure operators (and hence, to lattices) to reason about how to construct an Armstrong relation, or a cover for a set of FD's and to decide whether an FD can be derived from a set of FD's. All these items are also key subjects in database design. Finally, the work in [9] offers a lattice representation of a set of FD's and an algorithm to find a critical basis.

A link between degenerated multivalued dependencies (DMVD) and lattices has been proved in [4] where the lattice was not only useful to derive a set of DMVD's but also to find an Armstrong relation.

We finally would like to point out the links between Multivalued Dependencies (MVD) and lattices that have been developed so far. First, the work by [10], in which a lattice representation for FD's and MVD's is presented and some properties for those lattices are examined, together with a possible extension to join and embedded dependencies. In [23], concepts related to closure operators are used to describe efficient algorithms for the implication problem. A formalization of MVD's is also given in [26] in terms of a closure. In [16] the authors use a closure-related approach to find efficient algorithms to mine MVD's. In [3], the authors describe the semantics of MVD's and other related expressions using a lattice-oriented point of view.

The conclusion is that lattices have been widely used, more implicitly than explicitly, to describe different kinds of constraints (dependencies) and to perform different tasks, basically calculation of FD's and MVD's, the implication problem, Armstrong relations, and normalization in FD's.

The purpose of this paper is to propose an alternative lattice-based method to express tuples or a set of MVD's. We will prove that this representation calculates correctly the set of MVD's that hold in a set of data and constructs an Armstrong relation for a set of data or a set of MVD's.

Apart from having a new structure that will be as useful as those previously mentioned, we will have an FCA representation of data in which each concept will have now a specific semantics. In this case, since the semantics is given by the closure operator, the lattice will express partitions that show what sets of attributes are independent from each other for a given set of classes of tuples.

Our aim is twofold: on the one hand, to offer a novel lattice representation of MVD's and Armstrong relations, that combines the expressiveness of FCA with the capacity to calculate other representations of data using one single structure. This combination can offer a way to see database dependencies as extracted knowledge from the set of data with a particular semantics (dependency/independency of attributes, for instance). On the other hand, we wish to

advance towards the goal of relating the potential connection between the notions of dependency that can be exemplified by Armstrong relations and those that can be characterized in terms of a lattice representation ([4]).

The paper will start with a definition section to explain our notation as well as the combinatorial objects we will be dealing with. In the following section we will show our main results that can be summarized in a new Galois connection, that gives rise to a closure operator that permits us to calculate a lattice, from which to derive the set of MVD's that hold in a set of data. We will also show a way to create this lattice directly from a set of MVD's. Then, we will prove that a subset of the lattice produces an Armstrong relation for the data/MVD's that it represents. A concluding remarks section will end the paper.

2 Basic Definitions

In this section we will fix our notation and give some basic definitions and properties that will be used throughout the paper.

The definitions and propositions concerning database theory have been taken from [13] and [28]. For a complete explanation of MVD's, the reader can refer to [6], [14], [26], [27] and [30]. We will be dealing with a triple $(\mathcal{U}, Dom, r)$, where $\mathcal{U} := \{X_1, \ldots, X_n\}$ is a set of attributes, $Dom := \{Dom(X_1), \ldots, Dom(X_n)\}$ is the set of domains of values for each attribute, and $r \subseteq Dom(X_1) \times \cdots \times Dom(X_n)$ is a relation. A tuple $t \in r$ is a mapping from $\mathcal{U}$ into the union of the domains, such that $t(X_i) \in Dom(X_i)$ for all i. We will use capital letters from the end of the alphabet for sets of attributes, and we do not distinguish single attributes from singleton sets of attributes.

Definition 1. *Let $X \subseteq \mathcal{U}$ and $C \subseteq r$. We define the* **projection** *of μ onto X as $\pi_X(\mu) := \{t(X) : t \in \mu\}$, where $t(X)$ is the usual restriction of the mapping t on X.*

Definition 2. *Let X, Y two disjoint subsets of attributes and μ, μ' two classes of tuples. We define the* **cartesian product** *$\pi_X(\mu) \times \pi_Y(\mu') := \{t | t(X) \in \mu$ and $t(Y) \in \mu'\}$, as a set of tuples on $X \cup Y$.*

Definition 3. *A* **multivalued dependency** *$X \twoheadrightarrow Y$ holds in r iff whenever two tuples t_1, t_2 such that $t_1(X) = t_2(X)$ are in r, there is a tuple t_3 in r such that $t_1(X) = t_3(X)$, $t_3(Y) = t_1(Y)$ and $t_3(\mathcal{U} \setminus (X \cup Y)) = t_2(\mathcal{U} \setminus (X \cup Y))$.*

From now on we express such a MVD as $X \twoheadrightarrow Y|Z$, where $Z := \mathcal{U} \setminus (X \cup Y)$. In case either Y or Z are empty, the natural interpretation is that the corresponding multivalued dependency is trivial in the sense that it always holds. We will now define the dependency basis, $DEP(X)$, of a set of attributes X ([25]).

Definition 4. *Let $X \subseteq \mathcal{U}$, the* **dependency basis** *of X, $DEP(X)$, as the coarsest partition of $\mathcal{U}$ such that for all non empty Y's for which $X \twoheadrightarrow Y$ holds, Y is a union of elements of $DEP(X)$.*

Because of the reflexivity property for MVD's, each of the attributes in X is a singleton in $DEP(X)$. The previous definition can help us to summarize a group of MVD's with the same left side in a single expression as follows:

Definition 5. *For a partition of attributes $Y := \{Y_1|\ldots|Y_n\}$ of $\mathcal{U}$, we define a* **generalized MVD** *$X \twoheadrightarrow Y_1|\ldots|Y_n$ for a relation r if for all MVD's $X \twoheadrightarrow Y_i$ hold in r and, for each MVD $X \twoheadrightarrow Z$ that holds in r, Z is the union of some of the Y_i.*

We now define the basic concepts related to lattice theory that will be used. In [1] the closure operator relates sets of attributes with partitions of tuples following the intuitions of [21]. This allowed us to find the closures of attributes as well as their generators, and, later, to form the FD's that hold in that dataset. Modeling multivalued dependencies requires us to move up to a substantially more complex setting, where the Galois connection does not relate sets of tuples with sets of attributes, nor partitions of tuples with sets of attributes; rather, we have to consider families of classes of tuples that no longer constitute partitions, and relate them with partitions of attributes. Some remarks on our notation follow. By convention, we use variants of capital P, Q and R for partitions of attributes and their classes; variants of C mean classes of tuples, and they are considered together in sets of classes that play the role of partitions (although instead they may overlap), for which we use variants of μ as notation. Commas will separate various classes in a set, but we use $|$ to separate the different classes of a partition when we need to emphasize that they are disjoint. We assume that $\mathcal{U}$ and r are fixed for the remainder of this section, and denote $\mathcal{C}$ the set of simple sets of classes of tuples from r, that is, sets of classes of tuples such that there is no class in the set with is contained in another; and $\mathcal{P}$ the set of all possible partitions of $\mathcal{U}$.

We say that a partition P **refines** another partition P' if each class in P is a subset of some class of P'; in this case we also say that P' **coarsens** P. We denote this partial order with the standard ordering sign, oriented (unlike [20]) in the following way: $P' \leq P$ indicates that P' coarsens P or, equivalently, P refines P'. The lattice $L := (\mathcal{P}, \leq)$ is a complete lattice ([20]).

For sets of classes of tuples, we use the pre-order $\preceq$, with the following meaning: $C \preceq C'$ iff C' coarsens C, formally, iff $\forall \mu_i \in C : \exists \mu_j \in C' : \mu_i \subseteq \mu_j$. Because of the restriction on the set $\mathcal{C}$, the relation operator $\succeq$ is antisymmetric and then, it becomes a partial order. It is easy to see that with this restriction the operations meet and join in $\mathcal{C}$ are always defined: the join operator $C \vee C'$ returns the union of the sets that are in C and in C', deleting all the sets that are included in another set, and the meet operator $C \wedge C'$ returns the sets that result from the intersection of all the sets in C with all the sets in C', deleting all the sets that are included in another set. The top element is $\{\{\mathcal{U}\}\}$ and the bottom element is $\{\{\emptyset\}\}$. Then, the lattice $\mathcal{C}$ is a complete lattice ([8]).

Working our way towards the Galois connection, the next definitions relate partitions of attributes and classes of tuples.

Definition 6. *Let* $P := \{P_1|P_2|\ldots|P_n\} \in \mathcal{P}$. P **matches** *a class of tuples* μ *if* $\mu = \pi_{P_1}(\mu) \times \pi_{P_2}(\mu) \times \cdots \times \pi_{P_n}(\mu)$.

Of course the left-to-right inclusion always holds; the matching indicates that P hits a consistent way of projecting chunks out of μ. The following property is easily seen:

Proposition 1. *Let* P, P' *be two partitions of attributes with* $P \leq P'$, *and let* P' *match* μ; *then* P *also matches* μ.

In the extreme case, the coarsest possible partition matches everything, since in fact projecting on all the attributes leaves the set of tuples invariant. Along the other dimension, that is, classes of tuples, the smaller they are, the easier it may be to get finer matching partitions; in the limit, if C is a single tuple, it gets obviously reconstructed from all the projections so that every P indeed matches it.

Example 1. The following example will assume a relation r with four attributes $\{A, B, C, D\}$ and six tuples as follows: $r := \{\{a, a, a, a\}, \{a, a, b, b\}, \{a, b, b, a\}, \{a, c, b, b\}, \{b, a, a, b\}, \{b, b, b, a\}\}$. We have that the classes matched by $\{A|BCD\}$ are $\{\{1, 2, 3, 4\}, \{5, 6\}, \{3, 6\}\}$, since they can be expressed as

$$\{\{a\} \times \{aaa, abb, bba, cbb\}, \{b\} \times \{aab, bba\}, \{a, b\} \times \{bba\}\}$$

On the other hand, the partition of attributes $\{AD|B|C\}$ matches the classes of tuples $\{\{1, 5\}, \{2, 4\}, \{3, 6\}\}$, where the classes can be expressed as

$$\{\{aa, bb\} \times \{a\} \times \{a\}, \quad \{ab\} \times \{a, c\} \times \{b\}, \{aa, ba\} \times \{b\} \times \{b\}\}$$

3 Main Results

3.1 Definition of a new Galois connection

We are now ready to present a formal definition of a pair of operators that will form a Galois connection. These are:

Definition 7. *Operator* ϕ: $\mathcal{P} \to \mathcal{C}$. *This operator receives a partition of the set of attributes* $\mathcal{U}$: $P = \{P_1, \ldots, P_n\}$ *and returns the set of all the maximal classes of tuples* $\{\mu_1, \ldots, \mu_m\}$ *matched by* P.

Definition 8. *Operator* ψ: $\mathcal{C} \to \mathcal{P}$. *This operator receives a set of maximal classes of tuples* $C = \{\mu_1, \ldots, \mu_m\}$ *and returns a partition of the set of attributes* P *that is the finest partition of attributes that matches that set of classes.*

It is easy to see that there is always one only partition that given a class of tuples is the finest that matches that class. This is because of the following proposition:

Proposition 2. *If two partitions of attributes P_1, P_2 match μ, then $P_1 \vee P_2$ matches μ.*

Proof. Let us suppose that $P_1 \vee P_2$ does not match μ. It is because there is some combination of some values of, at least, two attributes should not appear. But two attributes will be in different classes iff they are in different classes in P_1 or in P_2. Then, since they will also be in different classes in one of them, all possible combinations of their values will appear in μ, which is a contradiction.■

Example 2. On the same data as our running example, we can consider some partitions of attributes and organize their matched classes into one or more partitions. Thus: $\phi(\{AC|B|D\}) = \{\{1|2,4|3,6|5\}, \{1|2,5|3,6|4\}\}$, and $\phi(\{AB|C|D\})$ $= \phi(\{A|B|C|D\}) = \{\{1|2,4|3,6|5\}\}$. In the opposite direction, $\psi(\{\{1,3,6|2,4,5\}\})$ $= \{ABC|D\}$.

The operator ψ could have an equivalent reformulation such that ψ returns *all* the partitions that match a set of classes of tuples. This is so because by Proposition 2 the set of partitions of attributes that match a given set of classes of tuples is a principal ideal, and then, it can be uniquely represented by the finest of such partitions. With this reformulation, we have that $\psi(C) = \{P \in \mathcal{P} | C \preceq \phi(P)\}$, and then, since both sets $\mathcal{P}$ and $\mathcal{C}$ are complete lattices, then, the pair (ϕ, ψ) is a Galois connection ([17]).

As a consequence, we have that the closure operator $\Gamma := \psi.\phi$ is a map $\Gamma : \mathcal{P} \to \mathcal{P}$ and the set of closed sets of $\mathcal{P}$ can be defined as follows: $\mathcal{P}_{CL} := \{X | X \in \mathcal{P}, X = \Gamma(X)\}$. This set, which is closed under the operator $\wedge$, plus the order relation inherited from $\mathcal{P}$ form a complete lattice ([20]).

3.2 Calculating MVD's with a Closure Operator

One of the main results of this paper consists in proving that the set of MVD's that holds in a relation can be identified by means of the closure operator defined in the previous subsection. We define a $P \in \mathcal{P}$ as a **generator** of a partition $P' \in \mathcal{P}_{CL}$ if $\Gamma(P) = P'$. Before we split the proof into both directions, we present two technical propositions.

Proposition 3. *Let r be a relation, and $P := \{X_1| \ldots |X_n|Y\}$ a partition of attributes where all X_i are singletons. There is always a maximal μ class matched by P such that μ contains all the tuples that have a given value of X in r.*

Proof. We fix a value $x \in \pi_X(r)$. We only need to prove that a class containing all the tuples $\mu' := \{t | t(X) = x\}$ can be matched by P, which is the following: $\{x\} \times \pi_Y(\mu)$. This class is matched by P, and contains all the tuples that have the same X value. ■

Proposition 4. *Let μ be a class of tuples matched by $P = X_1|\ldots|X_p|Y_1...Y_n$, and let the generalized MVD $X \twoheadrightarrow Y_1|\ldots|Y_n$ hold in r. Then, the class of tuples $\mu' = \pi_{X_1}(\mu) \times \cdots \times \pi_{X_p}(\mu) \times \pi_{Y_1}(\mu) \times \cdots \times \pi_{Y_n}(\mu)$ is in r.*

Proof. By contradiction. Let us suppose that there is a tuple t belongs to μ' but that does not appear in r. This tuple t has, at least, two values $t(Y_i) = y_i$ and $t(Y_j) = y_j$ that do not appear together in r (note that those values that do not appear together are in Y since the projections of the X_i are singleton in both μ and μ'). Since they are in μ', it is because they appear in some tuples in μ. Since they are not together in r, they are not together in μ, and since all the $\pi_X(\mu)$ values are combined with all the values of $\pi_Y(\mu)$, then, there is at least one tuple t_1 such that $t_1(X) = t_2(X)$ and $t_1(Y_i) = y_i$ and $t_2(Y_j) = y_j$. Now, since the generalized MVD $X \twoheadrightarrow Y_1|\ldots|Y_n$ holds in r and also in μ, we have that the MVD $X \twoheadrightarrow Y_i|Y \setminus Y_i$ holds as well. It means that since tuples t_1 and t_2 are in r, then, the tuple t_3 is such that $t_3(X) = t_1(X)$, $t_3(Y_i) = y_i$, and $t_3(Y \setminus Y_i)$ contains y_j, must be in r, which is a contradiction. ∎

Proposition 5. *Given a relation r, let $P := \{X_1|X_2|\ldots|X_p|Y_1|\ldots|Y_m\}$ be a closed partition according to the operator Γ defined by r, in which $p \geq 1$ and the X_i are singleton attributes; assume that $P' := \{X_1|X_2|\ldots|X_p|Y_1\ldots Y_m\}$ is a generator of P. Then, the MVD $X_1X_2\ldots X_p \twoheadrightarrow Y_1|\ldots|Y_m$ holds in r.*

Proof. We have to prove that for every pair of tuples t_1, t_2 such that $t_1(X) = t_2(X)$ we have a tuple t_3 such that $t_3(X) = t_1(X)$ and $t_3(Y_i) = t_1(Y_i)$ and $t_3(\mathcal{U} \setminus (X \cup Y_i)) = t_2(\mathcal{U} \setminus (X \cup Y_i))$. According to Proposition 3, t_1, t_2 should be together in one maximal class matched by P'. Now, since $\Gamma(P') = P$, both match the same maximal classes of tuples. Then, $\mu = \pi_{X_1}(\mu) \times \cdots \times \pi_{X_p}(\mu) \times \pi_{Y_1}(\mu) \times \cdots \times \pi_{Y_m}(\mu)$, which guarantees that the tuple t_3 will be in μ and, hence, in r. ∎

Proposition 6. *Let $X \twoheadrightarrow Y_1|\ldots|Y_n$ be a generalized MVD that holds in a relation r, where $X = X_1, X_2, \ldots, X_p$ and each of these is a single attribute; assume that the righthand side is as refined as possible, that is, coincides with the dependency basis $DEP(X)$. Then, the partition $P := \{X_1|X_2|\ldots|X_p|Y_1|Y_2|\ldots|Y_n\}$ is a closed partition according to the operator Γ derived from r, and $P' := \{X_1|X_2|\ldots|X_p|Y_1Y_2\ldots Y_n\}$ is a generator of P.*

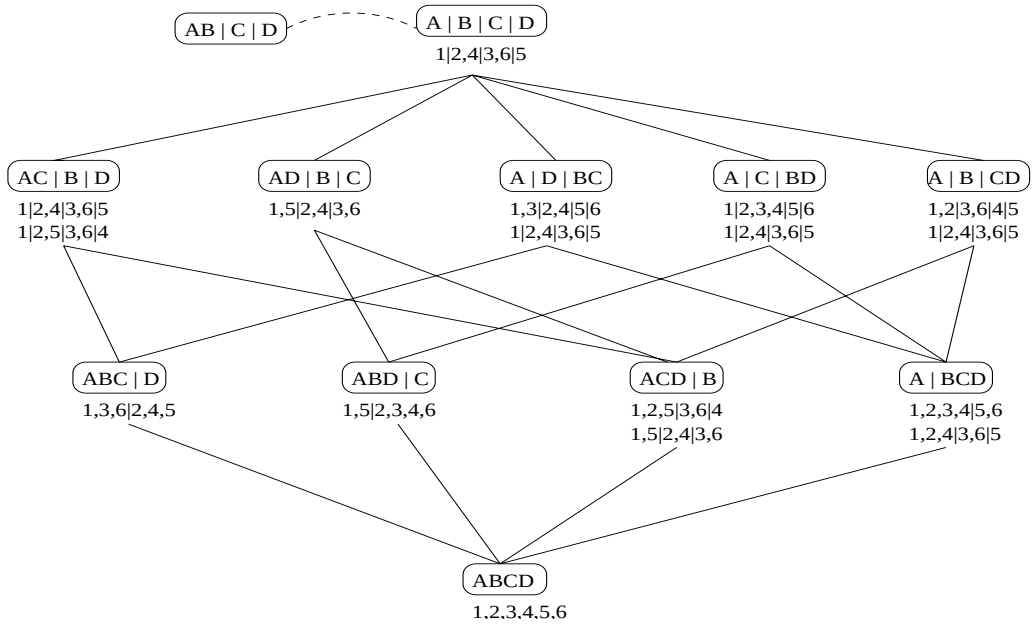
Proof. We must prove that all the maximal classes that are matched by P' are also matched by P. Let us suppose that there is a maximal class μ that is matched by P' and that it is not matched by P and have a contradiction. It is, we have $\mu := \pi_{X_1}(\mu) \times \cdots \times \pi_{X_p}(\mu) \times \pi_Y(\mu)$ and $\mu' := \pi_{X_1}(\mu) \times \cdots \times \pi_{X_p}(\mu) \times \pi_{Y_1}(\mu) \times \cdots \times \pi_{Y_n}(\mu)$, and $\mu \subset \mu'$. Since the generalized MVD $X \twoheadrightarrow Y_1|\ldots|Y_n$ holds in r, by Proposition 4 we have that $\mu' \subseteq r$. The class μ' is matched by P and then, by P', and the fact that $\mu \subset \mu'$ contradicts the maximality of μ. ∎

Combining the previous statements, we obtain our characterization of multivalued dependencies in terms of the closure operator Γ.

Theorem 1. *Let r be a relation, and Γ the closure operator on partitions of attributes obtained from r. Then, a generalized MVD $X_1 \dots X_p \twoheadrightarrow Y_1 | \dots | Y_n$ holds iff $\Gamma(\{X_1 | \dots | X_p | Y_1 \dots Y_n\}) = \{X_1 | \dots | X_p | Y_1 | \dots | Y_n\}$, for singletons X_i.*

Following the Example 1, the operator Γ defined previously yields the complete lattice of Example 3. We have marked with a dashed line the connection between the single nontrivial generator, $\{AB|C|D\}$, and its closure; according to our main result, this means that in the given dataset the generalized MVD $CD \twoheadrightarrow A|B$ holds (recall here that empty Y or Z yield trivial dependencies that always hold, such as $ABC \twoheadrightarrow D$, for one). Indeed, as it can be seen in Example 3, the generalized MVD $CD \twoheadrightarrow A|B$ can be formed as stated in Propositions 5 and 6 by the combination of a generator and its closure, which are, in this case, $\{AB|C|D\}$ and $\{A|B|C|D\}$ since $\Gamma(\{AB|C|D\}) = \{A|B|C|D\}$. Each node in the figure is labeled as well by some partitions of tuples that consist of matched classes (only maximal matched classes are shown).

Example 3. The concept lattice associated to our running example.



3.3 Calculating a Lattice from a Set of MVD's

It is trivial to show that, given a set Σ of MVD's, it is possible to construct a lattice that represents Σ and those dependencies that are logical consequence of Σ: we construct an Armstrong relation r from Σ (see [15]), then calculate the lattice with the operator Γ. However, we give a specific procedure to derive a lattice that represents a set Σ of MVD's and all the dependencies that are a consequence of this set. In order to proceed, we need to characterize the partitions that will appear in $\mathcal{P}_{CL}$ depending on the information that is provided by Σ for

a partition of attributes. It is easy to see (as the next lemma proves) that if a partition of attributes is a dependency basis for some set of attributes, then, it is closed and, hence, appears in $\mathcal{P}_{CL}$.

Lemma 1. *Let $P \in \mathcal{P}$, if P is a dependency basis for some set of attributes X then $\Gamma(P) = P$.*

Proof. Let us suppose that $P < P'$ and $\Gamma(P) = P'$. Since P' is finer that P, some class of P will be split in P', but none of these classes will contain X (since P is the dependency basis of X, all the attributes of X will be singletons in P). It means that some MVD's with X in their lefthand side will be formed (according to Theorem 1). At least, one of them would have in its righthand side a refinement of some class of P, which means that according to Definition 4, P would not be able to construct that MVD which has X in its lefthand side. But this is a contradiction with the fact that P is the dependency basis of X. ∎

The reverse of this implication does not hold, however. We still need to characterize the rest of partitions that may be in $\mathcal{P}_{CL}$ without being a dependency basis for some set of attributes. We will do it in the next lemma. First, we need to present the next proposition which is a technicality needed for this characterization that will be given in the next lemma. For $P, P' \in \mathcal{P}_{CL}$, the relation $P \blacktriangleleft P'$ means that $P < P'$ and there is no other partition Z such that $P < Z < P'$ unless $P' = Z$. For $P \in \mathcal{P}_{CL}$ the set of attributes that are singletons in P is $single(P)$.

Proposition 7. *Let $P \in \mathcal{P}_{CL}$. Let k be the number of classes in P that have more than one attribute. Then, there is at least k different $P'_1 \dots P'_k \in \mathcal{P}_{CL}$ such that $P \blacktriangleleft P'_i$ for all $i \in \{1 \dots k\}$.*

Proof. Let $P := \{P_1 | \dots | P_m | P_{m+1} | \dots | P_{m+k}\}$ $(k > 1)$ where for all $i > m$ P_i has, at least, two attributes. We will prove that for any pair of classes of attributes P_i, P_j $(i, j > m)$, the partition P' that will result from the splitting of P_i and the partition P'' that will result from the splitting of P_j will not match the same maximal classes of tuples. Which means that, since P is closed, and since no pair of split classes of attributes will match the same maximal classes of tuples, there are, at least, k differents sets of maximal classes of tuples that will be matched by partitions of attributes that will result from one split of P. We will suppose that there is some pair of classes of attributes that can be split and that match (both splits) the same classes of tuples, and derive a contradiction.

We now pick two classes of attributes P_i, P_j $(i, j \geq m)$ that are renamed as X and Y respectively. We let P' the partition of attributes that results from splitting X in P, and P'' the result of splitting Y in P. Since P is closed, then there will be some maximal class of tuples matched by P that will be split when matched by P' or P''. Let μ be one maximal class matched by P, that is split into different classes by P' or P'' (we assume that X is split into X_1, X_2 and Y is split into Y_1, Y_2). For the sake of clarity, and since we will be reasoning only on X and Y, we group the rest of the classes that are not X nor Y in one sole class

Z. We have that $\mu := \pi_Z(\mu) \times \pi_X(\mu) \times \pi_Y(\mu)$ since μ is matched by P. When X is split, we have that $\mu := \pi_Z(\mu) \times \bigcup_{\mu_i \in \mu} (\pi_{X_1}(\mu_i) \times \pi_{X_2}(\mu_i)) \times \pi_Y(\mu)$, where $\mu := \bigcup \mu_i$. Since all the values in $\pi_Y(\mu)$ are combined by the cartesian product with all the values in $\bigcup_{\mu_i \in \mu} (\pi_{X_1}(\mu_i) \times \pi_{X_2}(\mu_i))$ we can rewrite the last equation as follows: $\mu := \pi_Z(\mu) \times \bigcup_{\mu_i \in \mu} (\pi_{X_1}(\mu_i) \times \pi_{X_2}(\mu_i) \times \pi_Y(\mu))$. This class of tuples is matched by P'. On the other hand, we can reason alike when splitting Y, but, because of our hypothesis, the classes μ_i are the same. Then, we have that P'' matches the class of tuples $\mu := \pi_Z(\mu) \times \bigcup_{\mu_i \in \mu} (\pi_X(\mu) \times \pi_{Y_1}(\mu_1) \times \pi_{Y_2}(\mu_i))$. Since the classes μ_i are the same in both equations, for one class μ_i we have that $\pi_{X_1}(\mu_i) \times \pi_{X_2}(\mu_i) \times \pi_Y(\mu) = \pi_X(\mu) \times \pi_{Y_1}(\mu_1) \times \pi_{Y_2}(\mu_i)$, and, hence, $\pi_{X_1}(\mu_i) \times \pi_{X_2}(\mu_i) = \pi_X(\mu)$ and $\pi_{Y_1}(\mu_i) \times \pi_{Y_2}(\mu_i) = \pi_Y(\mu)$, which contradicts the fact that P is the finest partition that matches μ. ∎

We now characterize the sets that are in $\mathcal{P}_{CL}$ depending on whether they are a dependency basis and, if not, under which conditions.

Lemma 2. *Let $P \in \mathcal{P}_{CL}$. Then, P is a dependency basis for some subset of attributes, or it is not a dependency basis for any set of attributes and it is the meet of at least two partitions of attributes that are closed.*

Proof. By contradiction, we suppose that P is closed, is not the dependency basis for any set of attributes, and is not the meet of at least two closed sets. Suppose that P is composed by singletons and only one class that has more than one attribute. In this case, $single(P)$ must have a dependency basis. Since it cannot be P, then some P' such that $P < P'$ is the dependency basis of $single(P)$. But according to Proposition 6, then P must be a generator of P', which contradicts the fact that it is closed.

If P has more than one with more than one attribute, since we suppose that there is only one P' such that $P \blacktriangleleft P'$, it contradicts Proposition 7 that states that there must be as many partitions P' such that $P \blacktriangleleft P'$ as classes with more than one attribute in P. ∎

With this characterization, given a set of MVD's Σ, we can proceed to construct a lattice that will derive all the MVD's in Σ and only those MVD's that are logically implied by Σ: we calculate the dependency basis for all the possible sets of attributes, which according to Lemma 1 will be closed. Once we have this preliminary lattice, we add those sets that, not being a dependency basis for any set of atributes are the meet of some other set of closed partitions of attributes, as stated in Lemma 2 (it will have to be performed recursively). Both Lemmas 1 and 2 guarantee that this construction is a valid lattice, and that it represents Σ.

3.4 Armstrong Relations

In this section we will provide a method to calculate an Armstrong relation r' for a set of closed partitions of attributes. We recall that an Armstrong relation is a relation that obeys only a given set of dependencies Σ, and that for any dependency σ that is not logically derived from Σ, σ does not hold in r' (see [15]). Armstrong relations have two basic applications: on the one hand, they can be used, given a set of dependencies, to decide if a given dependency is a logical consequence of them (known as the implication problem) On the other hand, an Armstrong relation can be useful, from a database designer point of view, in order to have a relation that only shows relevant structural facts of the database and that has a size which normally should be smaller than the original data. In this sense, the Armstrong relation can be a tool to redesign and analyze a relation. The method that we will describe outputs an Armstrong relation given a lattice of partitions of attributes. This lattice can be created by the operator Γ or as explained in Section 3.3. This method is structurally similar to that in [4] with one main difference: we will prove that it is only necessary to use a subset of $\mathcal{P}_{CL}$, instead of the whole set.

In order to explain the construction of an Armstrong relation, we will define some concepts that will be used later. Given a partition of attributes P, we define $tuples(P)$ as a set of tuples formed by the following procedure: let $P := \{P_1|\ldots|P_m\}$, for each class of attributes we create a set of two different values if the class has more than one attribute, one otherwise. It is important to note that in case the class has more than one attribute, the pair of values will be different attribute wise and that all the values in $tuples(P)$ will be unique to the tuples in $tuples(P)$. Then, we apply the cartesian product to these values. For instance, $tuples(\{ABC|D\}) = \{a_1b_1c_1, a_2b_2c_2\} \times \{d_1\} = \{a_1b_1c_1d_1, a_2b_2c_2d_1\}$ and $tuples(\{AB|C|D\}) = \{a_3b_3, a_4b_4\} \times \{c_3\} \times \{d_2\} = \{a_3b_3c_3d_2, a_4b_4c_3d_2\}$.

Given the lattice $\mathcal{P}_{CL}$ induced by the closure operator Γ, we define $cover(\mathcal{P}_{CL})$ as the subset of $\mathcal{P}_{CL}$ such that $\forall Q_i \in cover(\mathcal{P}_{CL})$, Q_i is meet-irreducible. Or, stated in other words, the closure under meet of $cover(\mathcal{P}_{CL})$ equals $\mathcal{P}_{CL}$ and is minimal. In the following two propositions, we will characterize the role of $tuples(P)$ for a $P \in \mathcal{P}_{CL}$ and the interaction with the rest of partitions in $\mathcal{P}_{CL}$.

Proposition 8. *Let $P \in \mathcal{P}$. Then, for all $P' \not\leq P$, P' does not match $tuples(P)$.*

Proof. P' has two attributes Y, W in two different classes that are in the same class in P. By construction, $tuples(P)$ has two values for Y and two values for W, but each value of Y is combined only with a unique value of W (not all the combinations of values of Y and W will appear in $tuples(P)$). However in P', since both attributes are in different classes, because of the cartesian product all possible combinations of different values of Y and W will appear, which avoids P' from matching $tuples(P)$. ∎

This proposition can be rephrased as follows: $tuples(P)$ witnesses that P is a closed partition but only for P. Now, we can prove that an Armstrong relation can be derived from $cover(\mathcal{P}_{CL})$.

Theorem 2. $\bigcup_{Q_i \in cover(\mathcal{P}_{CL})} tuples(Q_i)$ *is an Armstrong relation for a relation represented by a lattice* $\mathcal{P}_{CL}$.

Proof. According to Theorem 1, it is enough to prove that this relation produces the same set of closed partitions. Equivalently, that if a given partition of attributes is closed under r, it is also closed under $r' = \bigcup_{Q_i \in cover(\mathcal{P}_{CL})} tuples(Q_i)$, and that if this partition is not closed under r it is not closed under r' either. We will prove it in two steps:

($\Rightarrow$) We suppose that P is closed in r but not closed in r' and derive a contradiction.

If $P \in cover(\mathcal{P}_{CL})$, $tuples(P)$ will be in r'. If we suppose that P is not closed in r', then a more refined partition must be its closure. According to Proposition 8, any refinement of P will not match $tuples(P)$, which is a contradiction.

If $P \notin cover(\mathcal{P}_{CL})$, then $P = \bigwedge Q_i$, for some $Q_i \in cover(\mathcal{P}_{CL})$. It means that $\forall Q_i : tuples(Q_i) \in r'$. Now, let us suppose that P is not closed in r'. It means that $P < \Gamma(P) = R$. But since $P = \bigwedge Q_i$, then $\exists Q_j : R \not\leq Q_j$, because if it was not the case, then, $\bigwedge Q_i = R$. Then, according to Proposition 8, R will not match $tuples(Q_j)$, but by Proposition 1 P will match $tuples(Q_i)$, which is a contradiction.

($\Leftarrow$) We suppose that P is a partition closed in r' but not in r and derive a contradiction.

Let $P' > P$ and $P' = \Gamma(P)$ in r, and let P_i be a class that is not in P'. Since P is closed in r', a partition formed by splitting P_i will not match the tuples matched by P. By construction of r', this class of tuples that is split when partitioning P_i is representing a closed partition of attributes, let it be Q. P_i and Q have in common the same class P_i. For the rest of the classes, Q may be finer than P :$P < Q$. In r, if $P < Q < P'$, then, it is not possible that $\Gamma(P) = P'$ because it would violate $P < Q \rightarrow \Gamma(P) < \Gamma(Q)$ (unless $P = Q$, which would mean that P is closed). If Q and P' are incomparable, since both are closed, $P' \wedge Q$ must be closed as well. It is obvious that $P \in \{P', Q\}^l$. If $P' \wedge Q = P$ then P must be closed in r, which contradicts previous assumptions. If $P' \wedge Q \neq P$ then, some there is some P'' such that $P'' > P$ and $P' \wedge Q = P''$, and P'' must be closed. But in this case, P' cannot be the closure of P because if would violate $P < P'' \rightarrow \Gamma(P) < \Gamma(P'')$. The case $P' < Q$ cannot happen because the class P_i is split in P' but not in Q. ∎

4 Concluding Remarks

In this paper we have shown a lattice-oriented method to calculate and represent data, whether presented as a set of tuples or a set of MVD's. When presented as a set of tuples, the lattice calculates the set of MVD's that hold in that set of tuples. This representation also calculates an Armstrong relation for the set of data or MVD's that it represents. We also have shown how to perform an implication

problem test for MVD's. It remains to be done a study on normalization (along the lines of [11]).

The representation that we have presented, not only permits to calculate MVD's and Armstrong relations; it also offers a lattice representation (similar in nature to those offered in FCA) of the data/MVD's. In this sense, it adds the expressivity of FCA to the calculation capabilities that have been proved in this paper.

Out of the scope of this paper is the algorithmic approach that can be derived from the results that have been shown. However, we would like to point out that in order to find the MVD's that hold in a given set of data, it is only necessary to keep track of those partition of attributes that are closed according to the closure operator Γ, and not the whole set of closed partitions. Moreover, the extends for each closure (it is, the set of classes of attributes matched by the intend) are not of interest for this purpose, so, this information could be removed as well.

References

1. Baixeries J. *A Formal Concept Analysis framework to model functional dependencies.* Mathematical Methods for Learning (2004).
2. Balcázar, J.L., Baixeries J. *Discrete Deterministic Data Mining as Knowledge Compilation.* Workshop on Discrete Mathematics and Data Mining in SIAM International Conference on Data Mining (2003).
3. Balcázar, J.L., Baixeries J. *Characterizations of multivalued dependencies and related expressions.* Discovery Science 2004.
4. Baixeries J., Balcázar, J.L. *Characterization and Armstrong relations for Degenerated Multivalued Dependencies using Formal Concept Analysis.* The Third International Conference on Formal Concept Analysis 2005.
5. Bastide Y., Taouil R., Pasquier N., Stumme G., Lakhal L. *Mining Frequent Patterns with Counting Inference.* SIGKDD Explorations 2(2): 66-75 (2000).
6. Beeri C., Fagin R., Howard J. H. *A complete axiomatization for functional and multivalued dependencies in database relations.* Jr. Proc. 1977 ACM SIGMOD Symposium, ed. D. C. P. Smith, Toronto, pp. 47-61.
7. Carpineto C., Romano G. *Inferring Minimal Rule Covers from Relations.* Computational Intelligence, Volume 15, numer 4, 1999, pages 415-441.
8. Davey B.A., Priestley H.A. *Introduction to Lattices and Order.* Second edition. Cambridge University Press, 1990, 2002.
9. Day A. *The lattice theory of functional dependencies and normal decompositions.* International Journal of Algebra Computational 2 (4) (1992) 409-431.
10. Day A. *A lattice interpretation of database dependencies.* Semantics of Programming Languages and Model Theory (M. Droste and Yu. Gurevich, eds), Gordon and Breach, London.
11. Demetrovics J., Hencsey G., Libkin L., Muchnik I. *Normal Form Relation Schemes: A New Characterization.* Acta Cybernetica. 10(3): 141-164 (1992).
12. Demetrovics J., Libkin L., Muchnik I. *Functional Dependencies in Relational Databases: A Lattice Point of View.* Discrete Applied Mathematics 40(2): 155-185 (1992).

13. Fagin R. *Multivalued dependencies and a new normal form for relational databases.* ACM Transactions on Database Systems 2, 3, Sept. 1977, pp. 262-278.
14. Fagin R., Vardi Y. V. *The theory of data dependencies: a survey.* Mathematics of Information Processing, Proceedings of Symposia in Applied Mathematics, AMS, 1986, vol. 34, pp. 19-72.
15. Fagin R. *Armstrong databases.* Invited paper, Proc. 7th IBM Symposium on Mathematical Foundations of Computer Science, Kanagawa, Japan, May 1982.
16. Flach P., Savnik I. *Discovery of multivalued dependencies from relations.* Intelligent Data Analysis,volume 4 (3,4): 195–211, November 2000.
17. Ganter, B., Wille R. *Formal Concept Analysis. Mathematical Foundations.* Springer, 1999.
18. Guigues J.L., Duquenne V. *Familles minimales d'implications informatives resultant d'un tableau de données binaires.* Math. Sci. hum., 24(95):5–18, 1986.
19. Gottlob G., Libkin L. *Investigations on Armstrong relations, dependency inference and excluded functional dependencies.* Acta Cybernetica 9(4): 385-402 (1990).
20. Grätzer, G. *General Lattice Theory.* Birkhäuser Verlag, 2000.
21. Huhtala Y., Karkkainen J., Porkka P., Toivonen H. *TANE: An Efficient Algorithm for Discovering Functional and Approximate Dependencies.* The Computer Journal 42(2): 100 - 111, 1999.
22. Lopes, S., Petit J-M., Lakhal L. *Functional and approximate dependency mining: database and FCA points of view.* Special issue of Journal of Experimental and Theoretical Artificial Intelligence (JETAI) on Concept Lattices for KDD, 14(2-3):93-114, Taylor and Francis, 2002.
23. Mok W. Y., Embley D. W. *On improving Dependency Implication Algorithms.* Information Processing Letters 64 (1997), pages 135 - 141.
24. Pfaltz, J.L. *Transformations of Concept Graphs: An Approach to Empirical Induction.* 2nd International Workshop on Graph Transformation and Visual Modeling Techniques. GTVM 2001, Satellite Workshop of ICALP 2001, Crete, Greece. Pages 320-326. July 2001.
25. Sagiv Y. *An algorithm for inferring multivalued dependencies with an application to propositional logic.* Journal of the ACM, 27(2):250-262, April 1980.
26. Thalheim B. *Entity-Relationship Modelling. Fundamentals of Database Technology.* Springer. Berlin 2000.
27. Thalheim B. *Conceptual Treatment of Multivalued Dependencies.* 22nd International Conference on Conceptual Modeling, Chicago, IL, USA, Lecture Notes in Computer Science Springer 2003, pp 363 - 275.
28. Ullman J.D. *Principles of Database and Knowledge-Base Systems.* Computer Science Press, Inc. 1988.
29. Valtchev P., Missaoui R. *Building Galois (Concept) Lattices from Parts: Generalizing the Incremental Approach.* Proceedings of the ICCS 2001, LNCS 2120, Springer Verlag, pp. 290-303, Stanford (CA), 2001.
30. Zaniolo C., Melkanoff M. A. *On the Design of Relational Database Schemata.* TODS 6(1): 1-47 (1981).

Conceptual Graphs for Knowledge Querying in VN-KIM

Tru H. Cao, Hai T. Do, Bao T.N. Pham, Tuyen N. Huynh and Duy Q. Vu

Faculty of Information Technology
Ho Chi Minh City University of Technology
Vietnam
tru@dit.hcmut.edu.vn

Abstract. The most appealing advantage of conceptual graphs (CG) is in their graphical knowledge representation for human consumption. On the other hand, for years, researchers and practitioners have put much effort in implementation of systems in which knowledge is directly stored as CGs for machine processing. However, none of those systems is widely used for large-scale applications outside the CG community. Meanwhile, recently, there have been new technologies developed as open sources for efficient management of knowledge and information represented in the Resource Description Framework (RDF), which can be mapped to and from CGs. This paper introduces our knowledge and information management system VN-KIM employing such a technology, called Sesame, and using simple CGs in its interface for expressive and user-friendly queries. We present an algorithm mapping simple CGs to queries in SeRQL, the query language of Sesame, for retrieving knowledge and information stored as RDF graphs. The system interface provides graphical tools for editing query CGs with property constraints.

1 Introduction

At ICCS'2000, the authors of [4] emphasized their final goal for CG research that was to apply CG to real world problems. That is probably also what the CG community is expecting for, especially with widely-used and large-scale applications. Since the first official introduction of CG in a book ([14]), there have been significant advances of the theory and much effort to realise its ideas, as having been presented in previous CG workshops and conferences. Looking at the front-end, it is clear about the advantage of CG in knowledge representation and visualization, in comparison to the traditional logical formalisms. Looking at the back-end, there has been a belief that knowledge storing and reasoning directly on CGs would be more efficient than on the forms of other languages.

Regarding recent development of some CG systems, CoGITaNT ([4]) is a platform to handle CGs at all levels, for internal representation, operations and reasoning, and human-computer interface. CGWorld ([8]) also uses CGs for queries, storage, and operations. In WebKB-2 ([11]), knowledge is stored in an extended model of CG, RDF, and terminological logics, and knowledge retrieval is performed as graph operations. However, the query forms expressed by textbox patterns at the interface of WebKB-2 are fixed and limited to only a central object with its relations to other objects. In Corese ([6]), both knowledge and queries are represented in RDF extended with some CG features, which are then converted into CGs for matching using CG operations.

Meanwhile, new technologies for management of knowledge and information, in particular on the Web, have emerged. For examples, Sesame ([10]) is for storing and querying RDF and RDF Schema (RDFS), Lucene ([9]) is for indexing and searching XML, and GATE ([7]) is for information extraction. Those are open sources and have been used in many large-scale systems. A technology is not necessarily the best possible solution to a problem, but it can do the job efficiently and for a large number of customers. That is especially important in today world, where there are problems that cannot be solved by any single theory or technique, and a system should be developed in a common popular framework to be shared and incorporated with others.

In this work, we make a turn from most of the existing systems using CG, by employing CG only for query expression, while knowledge storage and operations are handled by other technologies, in particular Sesame. This does not mean to compromise the believed good features of CG operations, such as graph projection or inference, because those features could be mapped and realized into another language or system, not necessarily in the CG form. The system we are developing is named VN-KIM, as inspired by KIM (Knowledge and Information Management) ([12]), for managing ontology and knowledge base of named entities mentioned in Vietnamese daily news, doing automatic extraction and annotation of entity classes in web pages, and retrieving those knowledge and annotated documents.

Firstly, for the paper being self-contained, Section 2 and Section 3 summarize the basic features of Sesame and its query language SeRQL ([13]). Section 4 presents our algorithm to map a query CG to one of SeRQL. Then Section 5 introduces VN-KIM knowledge base and query module. Finally, Section 6 concludes the paper and outlines future work.

2 Sesame

In order to effectively manage and exploit knowledge and information represented in RDF and RDFS, a system for efficient storage and expressive queries of large quantities of RDF and RDFS statements is a demand. Sesame is such an open framework meeting that need. The most essential design principle of Sesame is to make it extendable and adaptable to different operating environments. That idea has been realized in a generic modular architecture that enables Sesame to be installed on different database management systems (DBMS) and accessed via various communication protocols, as shown in Figure 2.1.

For achieving it, Sesame's developers have introduced storage and inference layers (SAIL) between upper modules and lower data repositories. They provide a set of basic methods for storing, querying, and manipulating RDF data provided, while optimized implementation of those methods is handled by a particular DBMS in use. In addition to memory and file storage mechanisms, Sesame currently supports the three DBMSs PostgreSQL, MySQL and Oracle 9i. Moreover, SAILs can also be stacked on top of each other, whence the developers can add in new SAILs for supporting further operations such as caching, concurrency handling, or inference.

Since clients can be on very different operational environments such as a Web server or a standalone application, Sesame offers a flexible communication mechanism. By placing all common communication methods at the module Request

Router and putting all protocol-specific methods at the protocol handlers, Sesame can support different communication protocols such as HTTP, RMI, and SOAP.

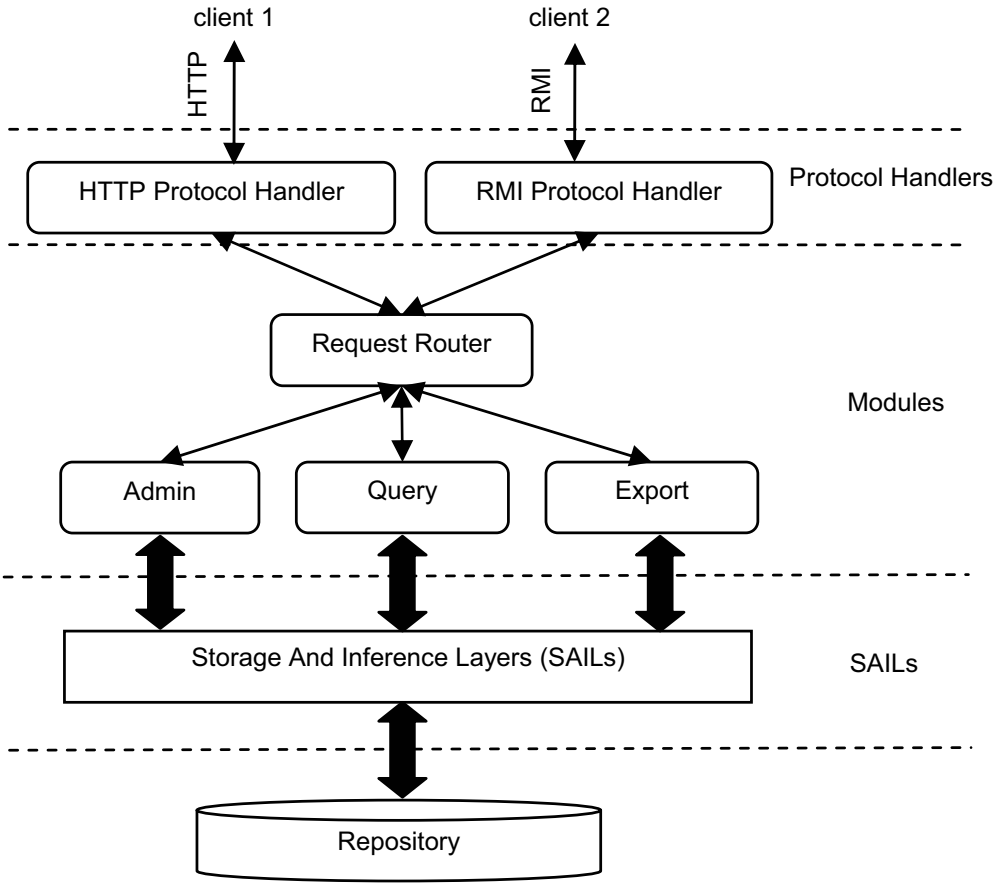


Fig. 2.1. Sesame architecture

Besides the above-mentioned essential features for storage and communication, Sesame also provides modules for querying, administration, security and versioning. In brief, it is a generic architecture that can be installed on a variety of different DBMSs, and a good candidate for storing, managing, and retrieving large-scale knowledge bases on the Web.

3 SeRQL

Along with the emergence of RDF/RDFS standard, many RDF/RDFS query languages have been proposed, such as RQL and RDQL. However, they still lack a strong path expression ability, which helps to simplify query formulation. Therefore, Sesame offers SeRQL (Sesame RDF Query Language) as a new RDF/RDFS query language that inherits the best features of RQL and RDQL, and equipped with some new features. SeRQL also supports subsumption, where a concept or relation type in a query can match with its sub-types in a knowledge base.

The basic building blocks of SeRQL are universal resource identifiers (URI), literals, and variables. A URI in Sesame can be written in two formats, either as a full URI, e.g., <http://www.dit.hcmut.edu.vn/vnkim/vnkimkb.rdf#Country_1>, or an abbreviated URI, e.g., <vnkimkb_rdf:Country_1>. In this case, the prefix vnkimkb_rdf will be mapped to the full string http://www.dit.hcmut.edu.vn/vnkim/vnkimkb.rdf by Sesame query engine.

RDF literals represent properties of objects and can optionally be attached with language or data type tags. Variables denote URIs or properties that are wanted, and identified in queries by names.

All RDF statements in a repository can be regarded as a graph, and searching can be performed by matching a query graph, consisting of path expressions, to that RDF graph. One outperforming feature of SeRQL as compared to other RDF/RDFS query languages is that it allows path expressions of arbitrary length. For example, path expressions of length 1 in SeRQL for the query "Find all cities that are located in Vietnam" could be as follows:

```
{City} <rdf:type> {<vnkimo_rdfs:City>},
{City} <vnkimo_rdfs:locatedIn> {Country},
{Country} <rdf:type> {<vnkimo_rdfs:Country>},
{Country} <rdfs:label> {"Vietnam"}
```

The second and the fourth path expressions, for instance, can be combined and rewritten as the following path expression of length 2:

```
{City} <vnkimo_rdfs:locatedIn> {Country} <rdfs:label> {"Vietnam"}
```

In SeRQL, nodes and edges in path expressions can be variables or URIs, while nodes can also be literals. In this example, City and Country are variables, "Vietnam" is a literal, and <vnkimo_rdfs:City> and <vnkimo_rdfs:Country> are URIs. This SeRQL query can be illustrated as a graph as shown in Figure 2.2.

A SeRQL selection query is built up from the four main clauses SELECT, FROM, WHERE, and USING NAMESPACE. The SELECT clause is used to determine which variables must be returned and in which order. The FROM clause contains path expressions. The WHERE clause expresses constraints on the values of variables. Finally, the USING NAMESPACE clause declares namespaces that are used for the mapping of abbreviated URIs. For example, the full SeRQL query for "Find all cities in Vietnam whose population is over one million" is:

```
SELECT City
FROM
    {City} <rdf:type> {<vnkimo_rdfs:City>},
    {City} <vnkimo_rdfs:locatedIn> {Country},
    {City} <vnkimo_rdfs:hasPopulation> {Population},
    {Country} <rdf:type> {<vnkimo_rdfs:Country>},
    {Country} <rdfs:label> {"Vietnam"}
WHERE
    Population > "1000000"^^xsd:positiveInteger
USING NAMESPACE
    rdf = <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
    rdfs = <http://www.w3.org/2000/01/rdf-schema#>
    vnkimo_rdfs  = <http://www.dit.hcmut.edu.vn/vnkim/vnkimo-rdf-schema#>
```

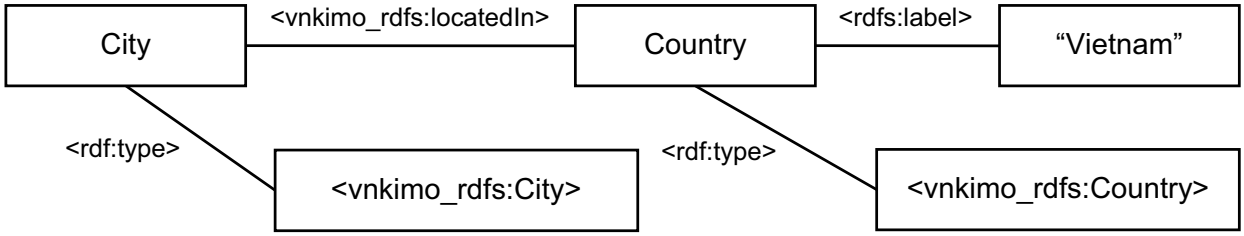


Fig. 2.2 Graph representation of a SeRQL query

SeRQL also provides some shortcuts to simplify query formulation. One of those shortcuts is the notation for branches in path expressions. That is useful for the case when one subject has several relations with other objects. Instead of repeating the subject for each path expression, those path expressions can share the common subject by using a semi-colon. For example, the above path expressions could be shortened as follows:

```
{City} <rdf:type> {<vnkimo_rdfs:City>};
       <vnkimo_rdfs:locatedIn> {Country},
{Country} <rdf:type> {<vnkimo_rdfs:Country>};
          <rdfs:label> {"Vietnam"}
```

Another useful shortcut is for reified statement, where a node is itself a statement. For example, one can have a statement in the form below:

```
{ {reifiedSubj} <reifiedPred> {reifiedObj} } <pred> {obj}
```

This would be equivalent to the following set of path expressions:

```
{reifiedStatement} <rdf:type> {rdf:Statement},
{reifiedStatement} <rdf:subject> {reifiedSubj},
{reifiedStatement} <rdf:predicate> {reifiedPredicate},
{reifiedStatement} <rdf:object> {reifiedObj},
{reifiedStatement} <pred> {obj}
```

Finally, it is worth mentioning that property constraints can be combined using the logical *not*, *and*, *or* operators in the WHERE clause. That makes SeRQL be an expressive query language for RDF knowledge bases, as SQL for relational databases.

4 CG-SeRQL Mapping

From the previous section, one could see that SeRQL is a powerful query language for RDF graphs, but it is not suitable for end-users, who are not familiar with the language's syntax and RDF structure. With the graphical knowledge representation, CGs are more readable. It has also been shown that there is a close mapping between CGs and RDF graphs ([6], [15]). In order to exploit Sesame infrastructure, we use simple CGs ([5]) extended with the queried referent at the interface level and map them to SeRQL for knowledge querying. We define query CGs as follows.

Definition 4.1 A *query CG* is a non-nested CG whose concept referents can be either an individual referent, the generic referent, denoted by '*', or the queried referent, denoted by '?'.

The generic referent means that it does not care about a matched individual referent. The queried referent represents the referent of an entity in question. Each query CG is associated with value constraints on the properties of its concepts.

Let us present the mapping via an example first. Suppose that one wants to find all persons who were born in Saigon and have height greater than 1.8m. This query can be expressed by the following SeRQL query, whose FROM clause can be mapped to the query CG in Figure 4.1:

```
SELECT x1, z1
FROM   [{x1} <rdfs:label> {z1}],
       {x1} <rdf:type> {<www.dit.hcmut.edu.vn/vnkim/vnkimo_rdfs#Person>},
       {x1} <www.dit.hcmut.edu.vn/vnkim/vnkimo_rdfs#has_birthplace>
              {<www.dit.hcmut.edu.vn/vnkim/vnkimo_rdfs#City_43>},
       {x1} <www.dit.hcmut.edu.vn/vnkim/vnkimo_rdfs#height> {p1}
WHERE  p1 > "1.8"
```

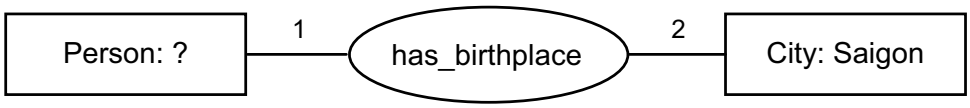


Fig 4.1. An example query CG

In this example, the queried concept [Person: ?] is translated into the pair of variables (x1, z1) in the SELECT clause of the SeQRL query, where x1 represents the identifier and z1 represents the label of a result entity. The path expression in square brackets [{x1} <rdfs:label> {z1}] is optional, for retrieving and displaying the label of a result entity if available. The individual concept referent Saigon specifies a known entity rather than the entity's name, and is translated into its identifier. The relations, including the implicit properties like label, type, and height, in the query CG are translated into RDF statements in the FROM clause. The constraint on height values are specified in the WHERE clause.

Following is pseudocode of the algorithm to translate a query CG into its equivalent SeRQL clauses:

```
For each relation in the query CG do
{
    Get all domain and range concepts of the current relation
    // process each concept sequentially
    For each concept do
    {
       If the concept is new // i.e. it has not been processed before
       {
          If the concept referent is '?' or '*'
          {
                If the concept referent is '?'
                {
                   Assign to it a variable starting with 'x' // e.g., x1, x2, x3, …
                   Assign to it a label variable starting with 'z' // e.g., z1, z2, z3, …
                   Append these variables to the SELECT clause
                   Append the "rdfs:label" statement for this concept to the FROM clause
                }
                If the concept referent is '*'
                {
                   Assign to it a variable starting with 'y' // e.g., y1, y2, y3, …
                }
```

```
                Append the "rdf:type" statement for this concept to the FROM clause
                Append the constrained property statements for this concept to
                the FROM clause
                Assign to the constrained properties variables starting with 'p'
                // e.g., p1, p2, p3, …
                Append the property constraints to the WHERE clause
            }
            If the concept referent is specific, assign to it its identifier as a
            dummy variable
            Remember this concept having been processed
        }
    }
    // end of process concepts
    Assign to the current relation its URI
    Append the statement for the current relation to the FROM clause
    // {domain concept variable} <current relation's URI> {range concept variable}
}
// process the disconnected concept nodes
For each concept that has not been visited yet, process it as above.
```

5 VN-KIM Query Module

5.1 Ontology and Knowledge Base

World Wide Web is moving to its next generation, called Semantic Web ([1]), in which not only humans can read and understand web pages, but computers can also understand and process them automatically. A clear disadvantage of the current web can be seen in search engines such as Google, which often return many useless documents that have nothing to do with what a user wants to find. The main reason is that no machine-understandable semantics is given to words in a document.

KIM ([12]) is a "killer application" for Semantic Web. It is firstly a knowledge-based system of popular named entities in the world. It can automatically extract the class of a named entity in a web page and annotate that information in the web page. Those semantics-enhanced web pages are then stored for efficient searching using entity names and their classes. For example, one can search for documents about "Washington" as a person, while current search engines may return any document that contains "Washington" though it is the name of a press or a university. Figure 5.1 shows a web page with highlighted named entities recognized by KIM.

Currently KIM ontology consists of about 250 classes, grouped into three upper-level concepts: Entity, EntityResource, and LexicalResource. Among them, Entity is the most important concept and specialized into Object (for people, locations, …), Happening (for events, situations, …), and Abstract (for addresses, professions, …). It also includes about 100 relations and attributes, such as subRegionOf for the class Location or hasPosition for the class Person. KIM knowledge base contains about 80,000 entities of the most important and popular people, cities, organizations, ..., in the world. For storing and managing the ontology and knowledge base, KIM employs Sesame. Annotated documents are indexed and retrieved using Lucene, by entity names and classes rather than keywords. KIM information extraction engine is based on GATE, an architecture for developing natural language processing applications.

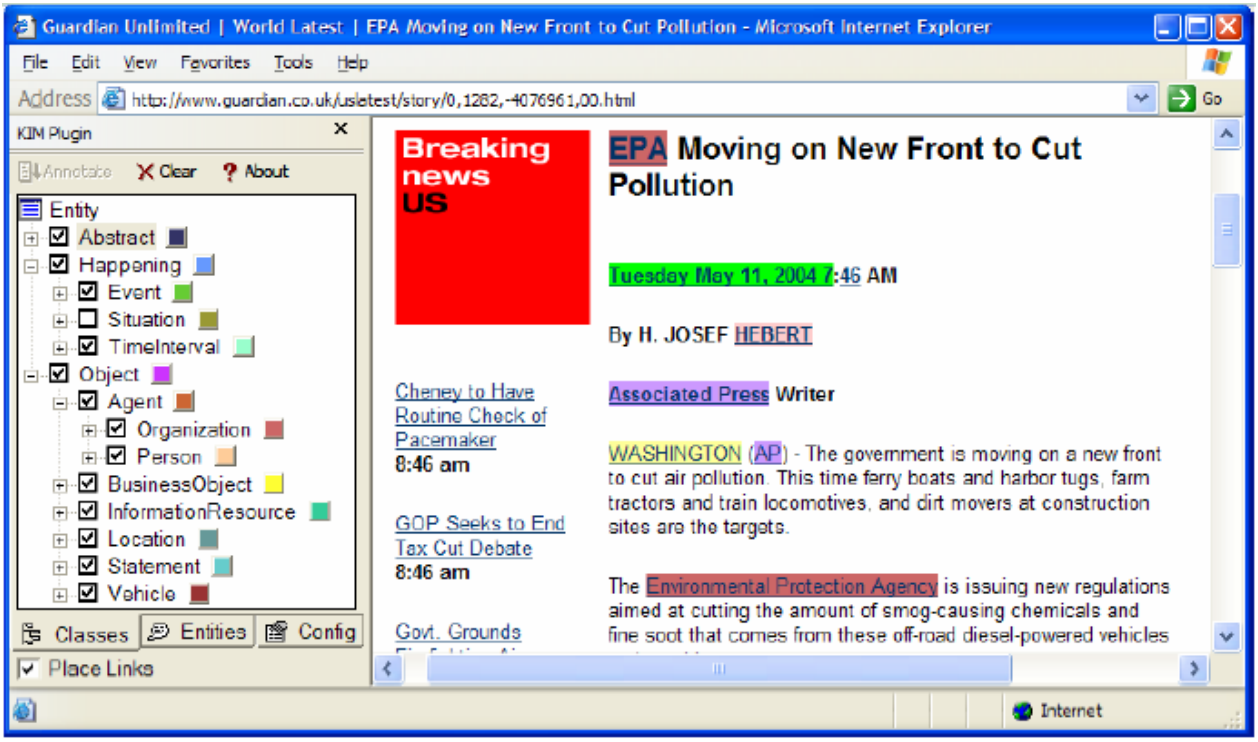


Fig. 5.1. Information extraction by KIM

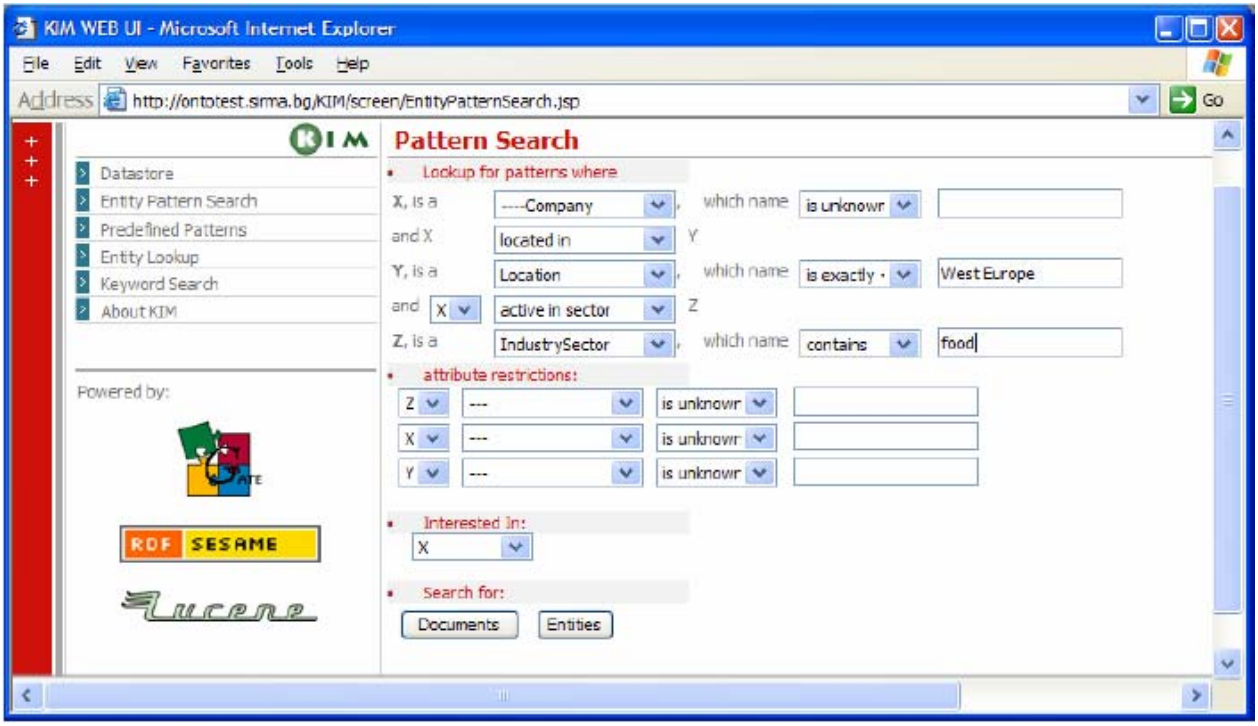


Fig. 5.2. A fixed query pattern in KIM

We are currently developing a similar system as KIM, named VN-KIM, but for Vietnamese language and domain, and being to overcome the disadvantages of KIM. One of the shortcomings of KIM is its usage of fixed patterns for knowledge querying. For example, Figure 5.2 shows such a pattern to express a query involving three entities, namely X, Y, and Z. In this pattern, one can express a relation between X and Y, and a relation between Y and Z, and the name and one property value restriction for each of the three entities, but no more.

While KIM query patterns are simple for usage and helpful in some cases, they are not sufficient and flexible enough for more expressive queries, which allow an arbitrary number of entities involved, multiple relations between them, and several property value constraints. On the other hand, SeRQL is much more expressive but not easy to use for end-users, who are not familiar to the language syntax. Below, we present our VN-KIM query editor and answering mechanism, which employs simple CGs for expressive and user-friendly knowledge querying.

5.2 Query Editor

VN-KIM query editor allows a user to pose queries in textbox patterns, CGs, or SeRQL clauses. It provides drawing tools for editing query CGs with respect to its ontology and knowledge base. Concept editing is assisted by browsing the concept type hierarchy and existing entities of a particular concept type. The relation type hierarchy can also be browsed for relation editing, and consistency of a relation and its neighbouring concepts in a CG is checked after the CG is drawn. Conditions on property values of an entity can be expressed using the constraint relations of the value domains as specified in the ontology. Query CGs are then mapped into SeRQL clauses as presented in Section 3 for knowledge and document retrieval.

The query editor is also equipped with functions for saving a query CG to an XML file, and loading a query CG from an XML file. The XML file contains both the logical and graphical information of each element in the query graph. The logical properties of one element are its category (i.e., concept, relation, or edge) and other specific properties (e.g. the type and referent of a concept element). The graphical properties such as positions and colours are used for displaying the graph. The details of such an XML file and its DTD are presented in the Appendix.

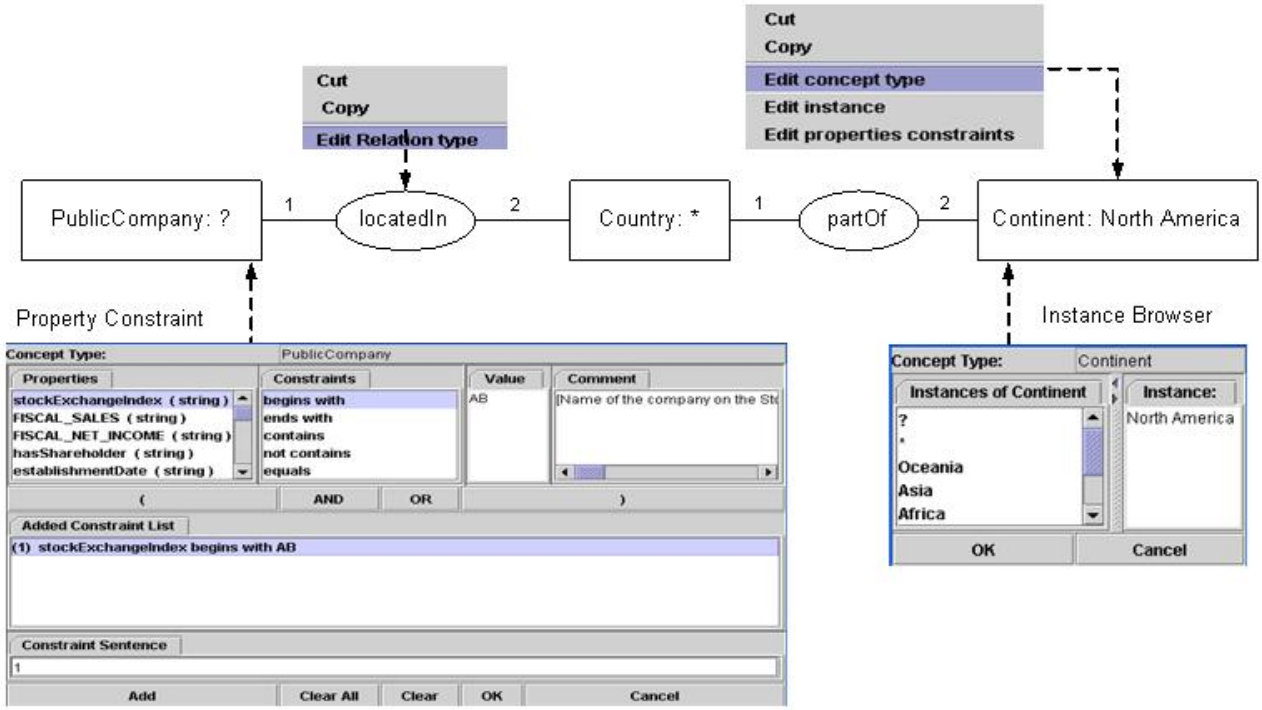


Fig. 5.3. An example query CG

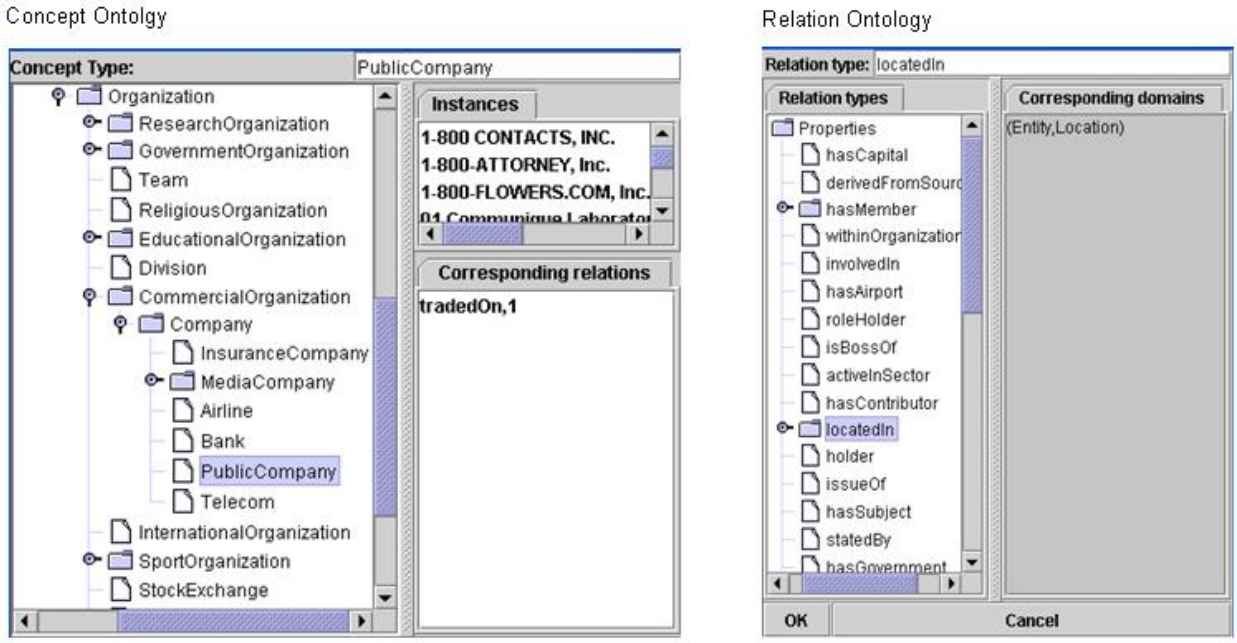


Fig. 5.4. Partial concept and relation ontologies

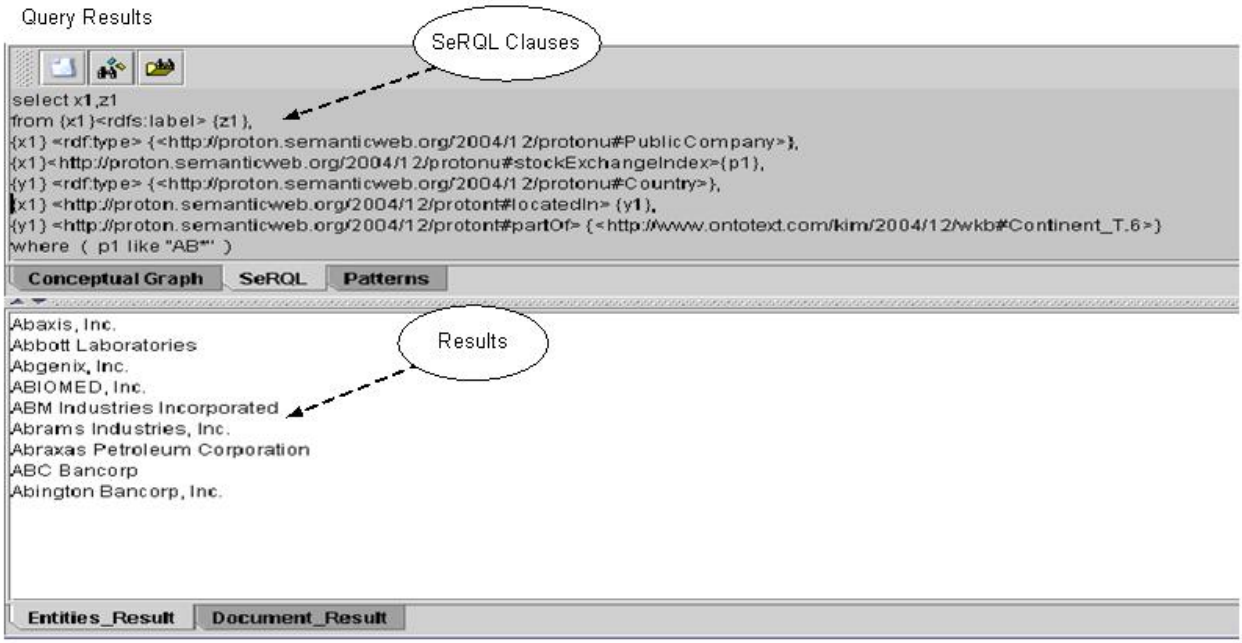


Fig. 5.5. Query results

For example, suppose that one wants to find all public companies that are located in a country of North America and have their stock exchange names begin with “AB”. This query can be posed by the CG edited using VN-KIM query editor as shown in Figure 5.3, together with assisting commands and information. The sub-hierarchies showing the concept and relation types involved are given in Figure 5.4, and the equivalent SeRQL clauses and results of the query are presented in Figure 5.5. For readability, we have run the example on KIM ontology and knowledge base, which uses English terms.

6 Conclusion

We have presented the usage of CGs as a query language in VN-KIM, a knowledge and information extraction and retrieval system for Vietnamese language and entities. We have shown that, for querying knowledge graphs, simple CGs are more expressive and flexible than fixed textbox patterns, and more readable than SeRQL clauses.

However, unlike other existing systems using CG, VN-KIM employs Sesame technology for managing knowledge in RDF and RDFS, which have been widely used. VN-KIM query module converts query CGs to SeRQL clauses to be answered by Sesame query engine. We have implemented an editor for conveniently drawing query CGs and expressing property constraints on them. A CG-SeRQL mapping algorithm and a structure of XML files for storing query CGs have also been presented.

The obtained result is two-fold. On one hand, we have exploited the advantage of CG for knowledge visualization. On the other hand, our system can be shared and incorporated with other systems on the popular RDF platform. We argue that using CGs at all levels of a system is unnecessarily, or even impossibly in certain cases, the best way to solve a problem and to popularize CG ideas. Combining the real advantages of CG with those of other languages should be more beneficial.

In this paper, only simple CGs can be used for querying and no partial matching is allowed for answering. On the theoretical basis of [2], simple query CGs can be extended to fuzzy ones with generalized quantifiers and imprecise property values. That also leads to research on partial matching of fuzzy CGs in a question answering process, whose partial result has been presented in [3]. Those are among the topics that we are currently investigating.

References

1. Berners-Lee, T: Semantic Web Roadmap. First draft available at http://www.w3.org/DesignIssues/CG.html (Last modified: February 2001).
2. Cao, T.H.: Conceptual Graphs for Modelling and Computing with Generalized Quantified Statements. In: Lawry, J., Shanahan, J., Ralescu, A. (eds.): Modelling with Words: Learning, Fusion, and Reasoning within a Formal Linguistic Representation Framework. Lecture Notes in Artificial Intelligence, Vol. 2873. Springer-Verlag (2003) 80-101.
3. Cao, T.H., Huynh, D.T.: Approximate Retrieval of Knowledge Graphs. In: Proceedings of the 11th World Congress of International Fuzzy Systems Association (2005) to appear.
4. Chein, M., Genest, D.: CGs Applications: Where are We 7 Years after the First ICCS? In: Ganter, B., Mineau, G.W. (eds.): Conceptual Structures: Logical, Linguistic, and Computational Issues. Lecture Notes in Artificial Intelligence, Vol. 1867. Springer-Verlag (2000) 127-139.
   Also at http://cogitant.sourceforge.net.
5. Chein, M., Mugnier, M.L.: Conceptual Graphs are also Graphs. Research Report No. 95003, Laboratoire d'Informatique, de Robotique et de Micro-électronique de Montpellier (1995).

6. Corby, O., Dieng-Kuntz, R., Faron-Zucker, C.: Querying the Semantic Web with Corese Search Engine. In: Proceedings of the 3rd Prestigious Applications Intelligent Systems Conference (2004).
7. Cunningham H. et al.: Developing Language Processing Components with GATE. GATE User's Guide Version 2.1 (2003).
Also at http://gate.ac.uk/.
8. Dobrev, P., Toutanova, K.: CGWorld – Architecture and Features. In: Priss, U., Corbett, D., Angelova, G. (eds.): Conceptual Structures: Integration and Interfaces. Lecture Notes in Artificial Intelligence, Vol. 2393. Springer-Verlag (2002) 261-270.
9. Gospodnetic, O.: Parsing, Indexing, and Searching XML with Digester and Lucene. IBM DeveloperWorks (2003).
Also at http://jakarta.apache.org/lucene/.
10. Kampman, A., Harmelen, F., Broekstra, J.: Sesame: a Generic Architecture for Storing and Querying RDF and RDF Schema. In: Proceedings of the 1st International Semantic Web Conference (2002).
Also at http://www.openRDF.org/ (Last visited: January 2005).
11. Martin, P., Eklund, P.: Knowledge Representation, Sharing and Retrieval on the Web. In: Zhong, N., Liu, J., Yao, Y.Y. (eds.): Web Intelligence. Springer-Verlag (2003) 243-276.
12. Popov, B. et al.: KIM – Semantic Annotation Platform. In: Proceedings of the 2nd International Semantic Web Conference (2003).
13. SeRQL Manual. Available at http://www.openRDF.org/.
14. Sowa, J.F.: Conceptual Structures - Information Processing in Mind and Machine. Addison-Wesley Publishing Company (1984).
15. Yao, H., Etzkorn, L.: Conversion from the Conceptual Graph (CG) Model to the Resource Description Framework (RDF) Model. In: Contributions of the 12th International Conference on Conceptual Structures (2004) 98-114.

Appendix: The structure and DTD of CG-XML files

The logical properties of a concept node:

- ID: the identifier of the concept.
- conceptType: the URI of the concept type.
- conceptReferentID: the URI of the concept referent ID.
- conceptReferentLabel: the label of the concept referent.
- propertyConstraints: the string containing all the property constraints of the concept.
- edgeID: the identifier of the edge attached to this concept.

The logical properties of a relation node:

- ID: the identifier of the relation.
- relationType: the URI of the relation type.
- edgeID: the identifier of the edge attached to this relation.

The logical properties of an edge:

- ID: the identifier of the edge.
- conceptID: the ID of the concept attached to this edge.
- relationID: the ID of the relation attached to this edge.
- edgeRole: the role of the concept attached to the relation via this edge.

The graphical properties of a CG element (i.e., concept, relation, or edge):

- (x1, y1): the upper-left corner position.
- (x2, y2): the lower-right corner position.
- useGradient: this specifies whether the element uses a gradient effect or not.
- startColor, endColor: the color used for the gradient effect.
- strokeSize: the size of the drawing pen.
- fill: specifies whether the element is filled or not.
- text: the displayed string of the element.

The DTD of CG-XML files:

```
<!ELEMENT shapes (shape*)>
<!ELEMENT shape (x1, y1, x2, y2, useGradient, startColor, endColor, strokeSize,
fill, text, ( (relationID, conceptID, edgeRole ) | ( ( relationType? | ( conceptType?,
conceptReferentLabel?, conceptReferentID?, propertyConstraints? ) ), edges?  ) ) )>
<!ATTLIST shape
    type CDATA #REQUIRED
    id CDATA #REQUIRED
>
<!ELEMENT x1 (#PCDATA)>
<!ELEMENT y1 (#PCDATA)>
<!ELEMENT x2 (#PCDATA)>
<!ELEMENT y2 (#PCDATA)>
<!ELEMENT useGradient (#PCDATA)>
<!ELEMENT startColor EMPTY>
```

```
<!ATTLIST startColor
     red CDATA #REQUIRED
     green CDATA #REQUIRED
     blue CDATA #REQUIRED
>
<!ELEMENT endColor EMPTY>
<!ATTLIST endColor
     red CDATA #REQUIRED
     green CDATA #REQUIRED
     blue CDATA #REQUIRED
>
<!ELEMENT strokeSize (#PCDATA)>
<!ELEMENT fill (#PCDATA)>
<!ELEMENT text (#PCDATA)>
<!ELEMENT relationType (#PCDATA)>
<!ELEMENT relationID (#PCDATA)>
<!ELEMENT conceptID (#PCDATA)>
<!ELEMENT conceptType (#PCDATA)>
<!ELEMENT conceptReferentLabel (#PCDATA)>
<!ELEMENT conceptReferentID (#PCDATA)>
<!ELEMENT propertyConstraints (#PCDATA)>
<!ELEMENT edgeRole (#PCDATA)>
<!ELEMENT edges (edge*)>
<!ELEMENT edge (edgeID)>
<!ELEMENT edgeID (#PCDATA)>
```

Difference Graphs

Harry S. Delugach and Aldo de Moor

Department of Computer Science
N300 Technology Hall
University of Alabama in Huntsville
Huntsville, AL 35899 USA
delugach@ cs.uah.edu

STAR Lab
Vrije Universiteit Brussel
Pleinlaan 2, Gebouw G-10
1050 Brussels, Belgium
ademoor@vub.ac.be

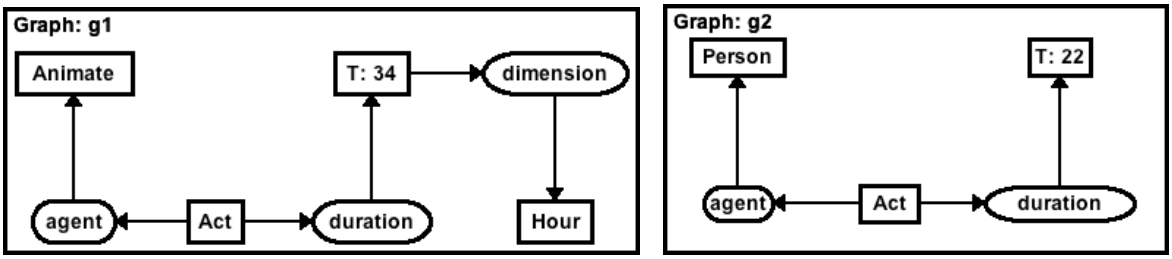
Abstract. In the conceptual structures community, much effort and attention has been focused on determining areas of commonality between two or more conceptual graphs. Less attention has been paid to another aspect of comparison, namely, what is *different* between two or more graphs. This paper explores a technique called difference graphs for determining and representing the differences between two graphs, both visually and formally. Difference graphs can be used in comparative analyses between mental models, process vs. practice models, and any domain in which multiple conceptual graphs can be obtained.

1 Introduction

In the conceptual structures community, much effort and attention has been focused on determining areas of commonality between two or more conceptual graphs; e.g., matching, similarity, overlap, etc. Less attention has been paid to another aspect of comparison, namely, what is explicitly *different* between two or more graphs. Intuitively, what is different comprises whatever *does not* match, whatever *is not* similar or whatever *does not* overlap.

There are many applications where it would be useful to understand the difference between two graphs. We could compare an observation with a model and identify where they do not match or compare two models or compare two observations and see how a situation has changed over time. Team-based large-scale development could benefit from a comparison of different developers mental models. Instructors could compare their own "right" answers with those submitted by students.

For instance, Figure 1(a) and (b) show two graphs that are different. How might the differences be characterized?



(a) (b)

Figure 1. Two different graphs.

A key feature of our comparisons is that the comparison process does not imply any "rightness" or "wrongness" with respect to one graph or the other. For example, if an observation does not match our model, it's possible that at least one of them is wrong, but merely comparing them does not tell us which one(s). The comparison processes we describe in this paper are for the purposes of determining differences between graphs, not deciding which graph to believe.

Another key feature of our comparison method is that we deliberately relinquish the claim of having a unique and deterministic solution, so that we speak of "a" difference, not "the" difference. This is to remind readers that (for any given pair of graphs) many distinct (yet correct) differences can be identified.

There are two main issues identified in this process:

- How to compute a difference? Most operations on graphs are aimed at establishing the common elements; e.g., finding the least common generalization or most common specialization. Since we are interested in the difference for human understanding and interpretation, we are satisfied (at present) with using any correct difference for our purposes.
- How to display (or otherwise convey) a difference? Semantically a difference graph (see below) may have the same constituents as its original graph but a visual editor allows coloring or other syntactic adornments to easily convey the difference.

The organization of the paper is first to explain how graph differences are computed with difference sequences; this is one technical contribution of the paper. We then explain how to display the differences through a coloring process; this is the second contribution of the paper. We then apply our technique to a real-world analysis problem involving model comparison, and offer some conclusions.

2 Computing Graph Differences

In order to compute graph differences, we first make an important pragmatic assumption:

Assumption. There need not be a provably unique difference characterization between any two graphs. Since our purpose is for humans to analyze the differences, we can allow that, for a given pair of graphs, more than one such difference is possible. It may be useful in the future to establish a formal mathematical description of what it means to be an "optimal" difference description, but for now we characterize the differences between two graphs by describing any sequence of changes needed to transform one into the other. In this way, we avoid having to justify that our approach is tractable in all cases; we need only show that it leads to a correct difference description.

Definition. A *difference operation* is any one of the transformation operations shown in Figure 2. We call the starting graph the *source* graph and the resulting graph the *result* graph. These operations transform part of one graph into part of another graph.

Definition. The *inverse* of a difference operation is an operation which will transform the result of a difference operation back into its original graph. Note that in Figure 2, each operation has an inverse operation associated with it; later, we will show how the inverse operations are used.

Difference operation	Name	Description	Inverse
Specialize concept	**S-C**	Replace with subtype, or Add new referent	**G-C**
Specialize relation	**S-R**	Replace with subtype	**G-R**
Generalize concept	**G-C**	Replace with supertype, or Delete old referent	**S-C**
Generalize relation	**G-R**	Replace with supertype	**S-R**
Insert subgraph	**I-G**	Add subgraph components	**D-G**
Delete subgraph	**D-G**	Remove subgraph components	**I-G**
Insert co-referent link	**I-L**	Link two or more concepts	**D-L**
Delete co-referent link	**D-L**	Remove link from two or more concepts	**I-L**
Move into context	**M-I**	Move a subgraph components into a context	**M-O**
Move from context	**M-O**	Move a subgraph from a context	**M-I**

Figure 2. Difference operations and their inverses.

Note that we make no claim about the truth-preserving characteristics of these steps: this clearly distinguishes them from the usual notion of a conceptual graph "transformation rule" which is intended to preserve the original graph's meaning, as in work such as [1]. Here we have explicit change steps, which we expect *will change* the intent of the original graph – that is why we are doing it.

Each difference operation represents a small change. The overall difference between two graphs is thereby captured by the accumulation of many small differences. We offer some brief definitions.

Definition. A *difference sequence DS(G, H)* between graphs *G* and *H* is a sequence of zero or more difference operations applied in order to *G* that will transform it into *H*.

Definition. The *inverse difference sequence* (or simply, *inverse*) of a difference sequence *DS(G, H)* is a sequence of zero or more difference operations applied in order to *H* that will transform it back into *G*. This sequence is shown as $DS^{-1}(G, H)$.

For any difference sequence DS consisting of steps $s_1, s_2, \ldots, s_n$, there exists an inverse sequence DS^{-1} consisting of steps $s_n^{-1}, s_n^{-1}, \ldots s_2^{-1}, s_1^{-1}$, where each s_i^{-1} represents the inverse of step s_i. We can thus completely specify the difference between two graphs by specifying either *DS* or DS^{-1}; we sometimes use both for convenience.

Notation. Two or more difference sequences can be concatenated to form a single sequence. We show concatenation between sequences with a "+" symbol, so that *DS(G, H) + DS(H, J) = DS(G, J)*.

Notation. A least common generalization between graphs *G* and *H* is denoted by *LCG(G, H)*. The least common generalization notion is the same as in the standard conceptual graph literature e.g., [2], [3].

There are several different ways to obtain a difference sequence. We partition the procedure by assuming an intermediate graph – namely an LCG. This provides a convenient starting point, where we begin by obtaining an LCG, then we generalize "up" to that LCG from the first graph, and then specialize "down" to the second graph[1], as shown in Figure 3. We use the LCG because it is a well-understood property known to be computable for any two graphs [4] [5] and therefore we do not need to justify the choice of any particular LCG. A consequence of this decision is that there may be more than one valid difference sequence for a given pair of graphs.[2]

The algorithm is organized into four steps:

- Obtain the LCG of the two graphs
- Find a sequence of difference operations that transforms the first graph into the LCG
- Find a sequence of difference operations that transforms the LCG into the second graph.
- Concatenate the two sequences.

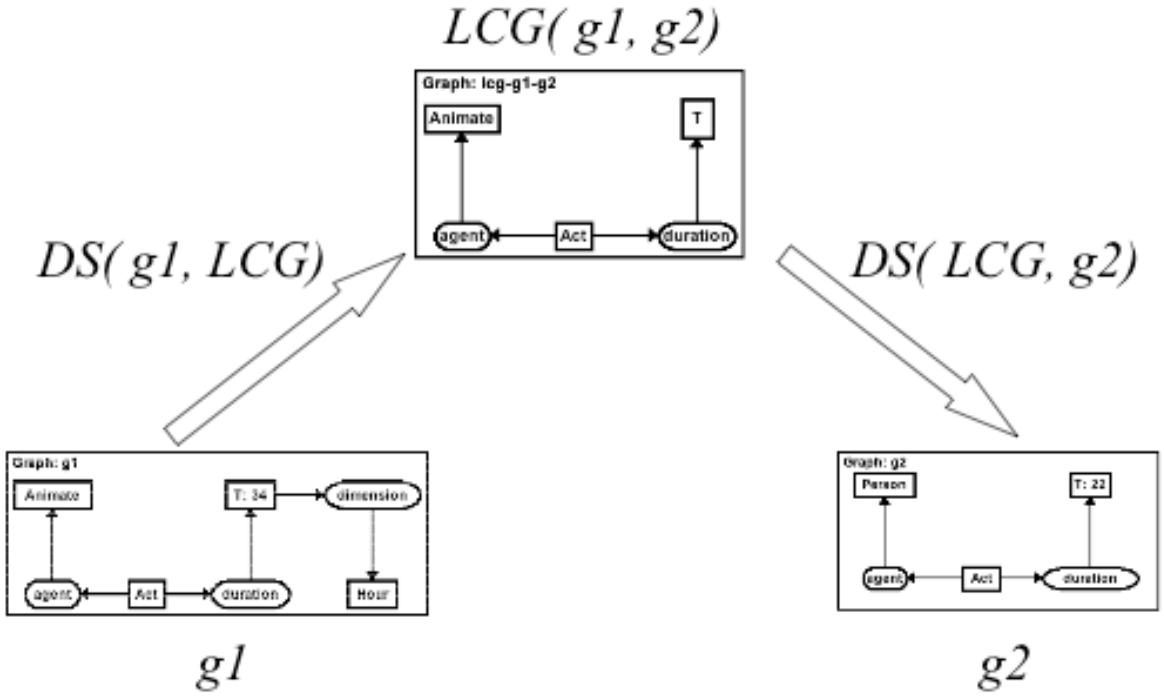


Figure 3. General form of the algorithm.

[1] We could just as easily start with the second graph, generalize to an LCG, and then specialize to the first graph. The choice is arbitrary. The important point is that for any given pair of graphs, there exists at least one LCG to serve as a "bridge".

[2] Some readers may not be comfortable that there is more than one LCG for a given pair of graphs and therefore there may be more than one correct difference sequence. Because we use the difference graphs for human analysis, we can allow for some non-determinism in obtaining the graphs.

An example of applying the algorithm is shown in the rest of this section. We obtain a set of operations to transform the first graph into one of its LCG's, then obtain a set of operations to transform that same LCG into the second graph. The concatenation of these two sequences forms an overall difference sequence.

2.1 Obtain an LCG of the two graphs

A least common generalization (*LCG*) of the graphs in Figure 1(a) and (b) is shown in Figure 4.

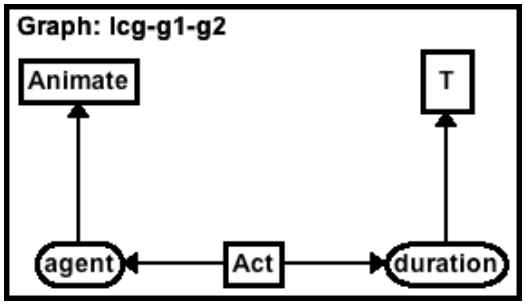


Figure 4. Least Common Generalization (LCG) of two graphs.

2.2 Transform first graph into LCG

Since we begin this part of the algorithm with the *LCG*, we will determine *DS(LCG, g1)* and then get its *inverse* DS^{-1}*(LCG, g1) = DS(g1, LCG)*. This is, we will first obtain a sequence of operations that transforms the *LCG* into *g1*. This is more convenient because we intend to find the difference sequence from the LCG to the second graph using the same process. We then invert the sequence (i.e., invert each of its operations and reverse the order). To transform the *LCG* graph into each of the two graphs, we list the following steps (each of which specializes the *LCG* graph):

DS(LCG, g1) :
S-C: Specialize **T** to **T: 34**
I-G: Insert **(dimension) -> [Hour]**

Inverting this sequence, we get DS^{-1}*(LCG, g1) = DS(g1, LCG)*.

DS(g1, LCG) :
G-C: Generalize **T:34** to **T**
D-G: Delete **(dimension) -> [Hour]**

2.3 Transform the LCG into the second graph

The second sub-sequence is somewhat easier than the first, only because we do not have to invert it as for *DS(g1, LCG)*.

DS(LCG, g2) :
S-C: Specialize **Animate** to **Person**
S-C: Specialize **T** to **T: 22**

2.4 Concatenate the two sub-sequences

Once the two sub-sequences are obtained, their concatenation is straightforward:

$$DS(g1, g2) = DS(g1, LCG) + DS(LCG, g2)$$

A complete difference sequence *DS(g1, g2)* is therefore:

DS(g1, g2) :
G-C: Generalize **T:34** to **T**
D-G: Delete **(dimension) -> [Hour]**
S-C: Specialize **Animate** to **Person**
S-C: Specialize **T** to **T:22**

This section demonstrated a straightforward method of obtaining a difference sequence between conceptual graphs that can be used to describe in a formal way how two graphs are different. While these operations are fairly simple, their combinations can be quite powerful in capturing graph differences. In particular, we intend to apply these differences to support humans in analyzing pairs of graphs (see sec. 4 below). The next section describes how we display the differences for human use and benefit.

3 Displaying Differences For Human Interpretation

While our calculation of a difference sequence may be logically adequate, we are interested in using graph differences to guide human users in analyzing the graphs. We therefore want to highlight the different parts so that human users can quickly determine what has changed between the graphs. We adopt the convention of *coloring*, i.e., using graphical shading in the displayed graphs. We show difference graphs in Figure 5 and Figure 10. We use gray for the un-changed parts of the graphs and white for the changed parts, but other highlighting strategies could be used.

Displaying the differences between two graphs can be done from the perspective of either of the two graphs. This means that two distinct difference graphs are meaningful for any pair of graphs.

Definition. A *difference graph* between two graphs *g1* and *g2* is a copy of either *g1* or *g2* with the common elements of the graphs shaded in one color, and the different elements shaded in another color. For example, given the two graphs *g1* and *g2* in Figure 5(a) and (b), two difference graphs are shown in Figure 5(c) and (d).

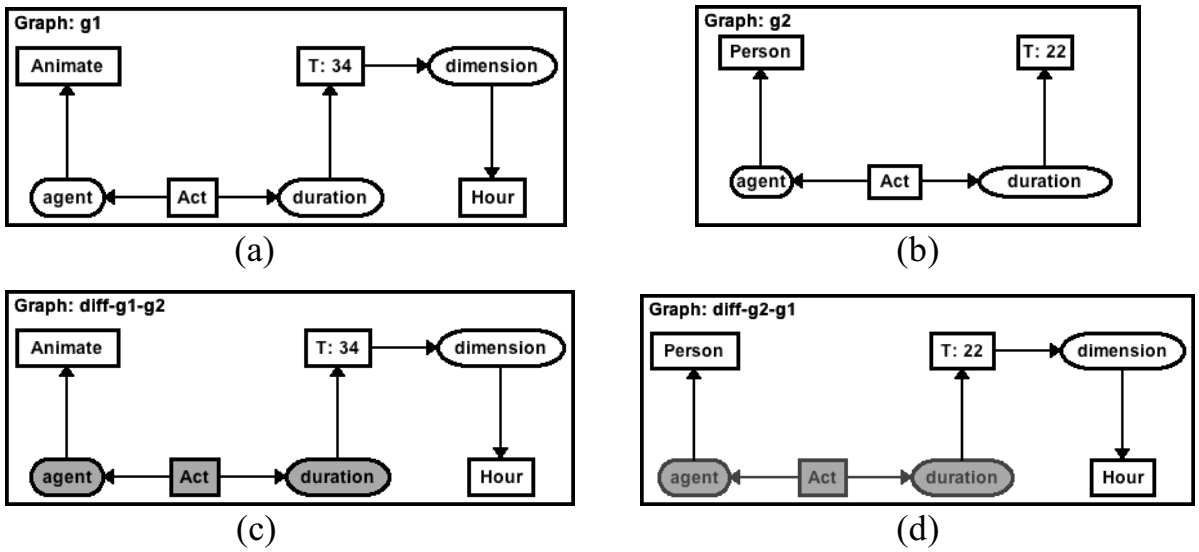


Figure 5. Displaying the differences between graphs.

We adopt the naming convention that a difference graph is named "diff" followed by the name of the original graph to be displayed, followed by the name of the second graph whose differences are to be highlighted. Therefore **diff-g2-g1** is a copy of *g2*, with the differences from *g1* highlighted.

One issue is immediately apparent: a highlighted component of a difference graph may comprise more than one kind of difference. For example, in **diff-g2-g1**, the **Person** concept is highlighted; however, it may be highlighted for any of the following reasons:

- The **Person** concept doesn't appear in graph *g1* at all.
- The **Person** concept is a (possibly indirect) subtype or supertype of a concept that does appear in graph *g1*.
- The **Person** concept in *g1* may be an individual concept (i.e., one that has a referent).
- The **Person** concept "matches" a concept in *g1*, but *g2* uses a different canonical basis and therefore the two graphs aren't comparable via subtypes or supertypes at all.

Because of the different kinds of operations, we have found it useful to derive both difference sequences (i.e., one difference sequence and its inverse) so that we can use the coloring technique from the perspective of both the graphs, as shown in Figure 5. The complete list of operations appears in Figure 6. Bear in mind that the coloring operates for difference sequences in both directions.

Difference operation	Name	To be highlighted	Where
Specialize concept	**S-C**	Concept	Both graphs
Specialize relation	**S-R**	Relation	Both graphs
Generalize concept	**G-C**	Concept	Both graphs
Generalize relation	**G-R**	Relation	Both graphs
Insert subgraph	**I-G**	Added subgraph	Result graph
Delete subgraph	**D-G**	Deleted subgraph	Source graph
Insert co-referent link	**I-L**	Linked concepts	Result graph
Delete co-referent link	**D-L**	Un-linked concepts	Source graph
Move into context	**M-I**	Moved elements	Result graph
Move from context	**M-O**	Moved elements	Source graph

Figure 6. Coloring guidelines.

We have used the coloring technique in the example of the next section. This is our first attempt at visually describing graph differences; we expect that more elaborate and revealing color schemes may be possible in the future.

4 Applying the technique

One of the motivations of our work is that we wanted to develop techniques to identify differences between software development formal process models and models of the actual practice performed by developers. Conceptual graphs seemed a reasonable choice to represent both the process model and the practice model. In this

section, we illustrate the technique with a larger example, where we first calculate the difference sequence and then analyze it.

The graphs in this section were obtained in a practical modeling experiment where we were modeling an organization's requirements development process. The purpose of the two graphs is not central here; we will simply say that Figure 7 shows a process model of how the organization wants to operate, while Figure 9 shows how the organization actually operates.

4.1 Computing a difference sequence

We base the discussion of this section on the two graphs shown in Figure 7 and Figure 9. These graphs are the actual ones obtained in a realistic knowledge capture exercise with a software project manager.

One immediate reaction when inspecting *G1* and *G2* is that they seem intuitively quite similar. Since we used the same canonical basis for obtaining both graphs from our human knowledge source, this should not be surprising.

It should be noted that for large graphs, while some differences may be "obvious", there are many reasons why it may be difficult to find all of them manually: (i) we may draw the graph with different spatial arrangements, representing the same topology, (ii) some types may have similar names, and (iii) the sheer number of concepts and relations may be overwhelming for human perception.

To find a difference sequence for this pair of graphs, we follow the same procedure as before:

First, we found a least common generalization between the two graphs; this is not shown due to space limitations. Next we found two difference sequences between the LCG and the two graphs, which were then combined to show the difference sequence in Figure 8.

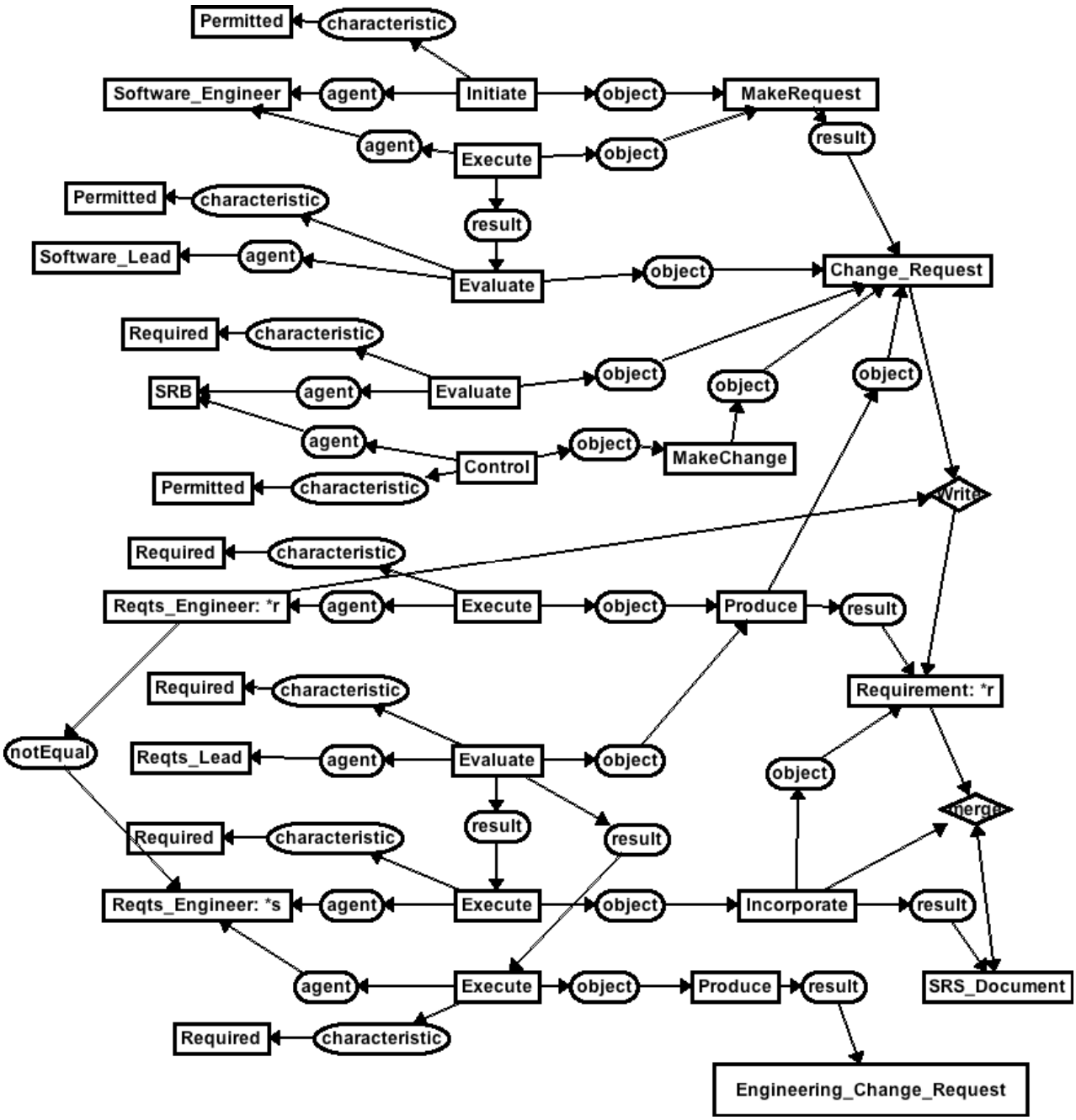


Figure 7. Example Graph *G1*, A Process Model.

DS(G1, G2) :

G-C: Generalize **Reqts_Engineer: *r** to **Reqs_Engineer**
G-C: Generalize **Reqts_Engineer: *s** to **Reqs_Engineer**
D-G: Delete **notEqual** relation
S-C: Specialize **Software_Engineer** to **Software_Engineer: Jerry**
S-C: Specialize **Reqts_Engineer** to **Reqts_Engineer: Jerry**
I-L: Insert **coreferent link** between **Software_Engineer: Jerry** and **Reqts_Engineer: Jerry** and **Reqts_Engineer: Jerry**

Figure 8. A difference sequence for two larger graphs.

We summarize the differences visually by highlighting them in the two graphs. Figure 10(a) shows a segment of Project-A's process model from Figure 7; Figure 10(b) shows a segment of its practice model from Figure 9. The white nodes are ones

where that model differs from the model to which it has been compared, i.e. it has a node the other model does not have, or the label of the node is different of the node in the same place in the other model; whereas the gray nodes are those that are the same for the two graphs.

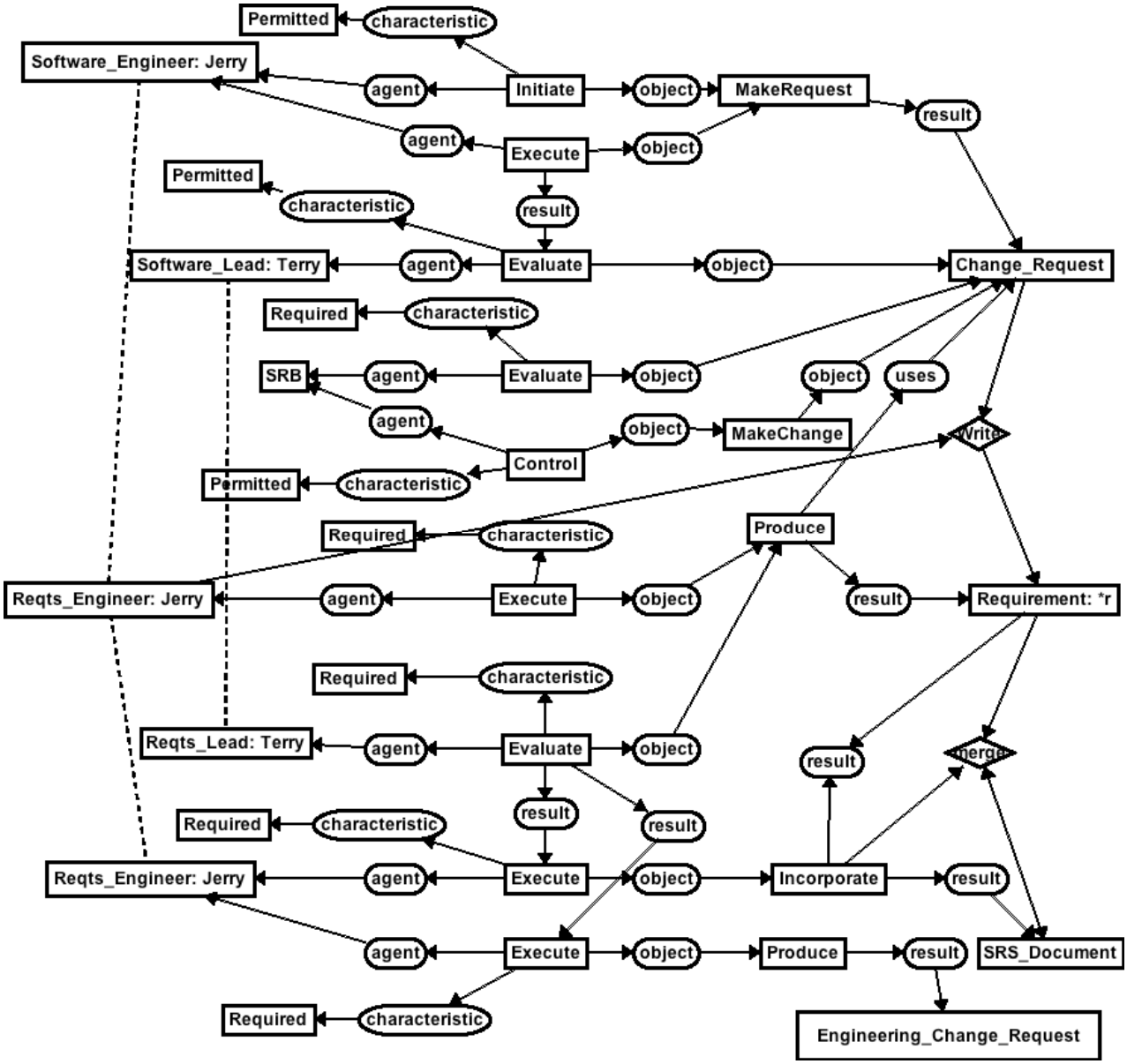


Figure 9. Example Graph *G2*, A Practice Model.

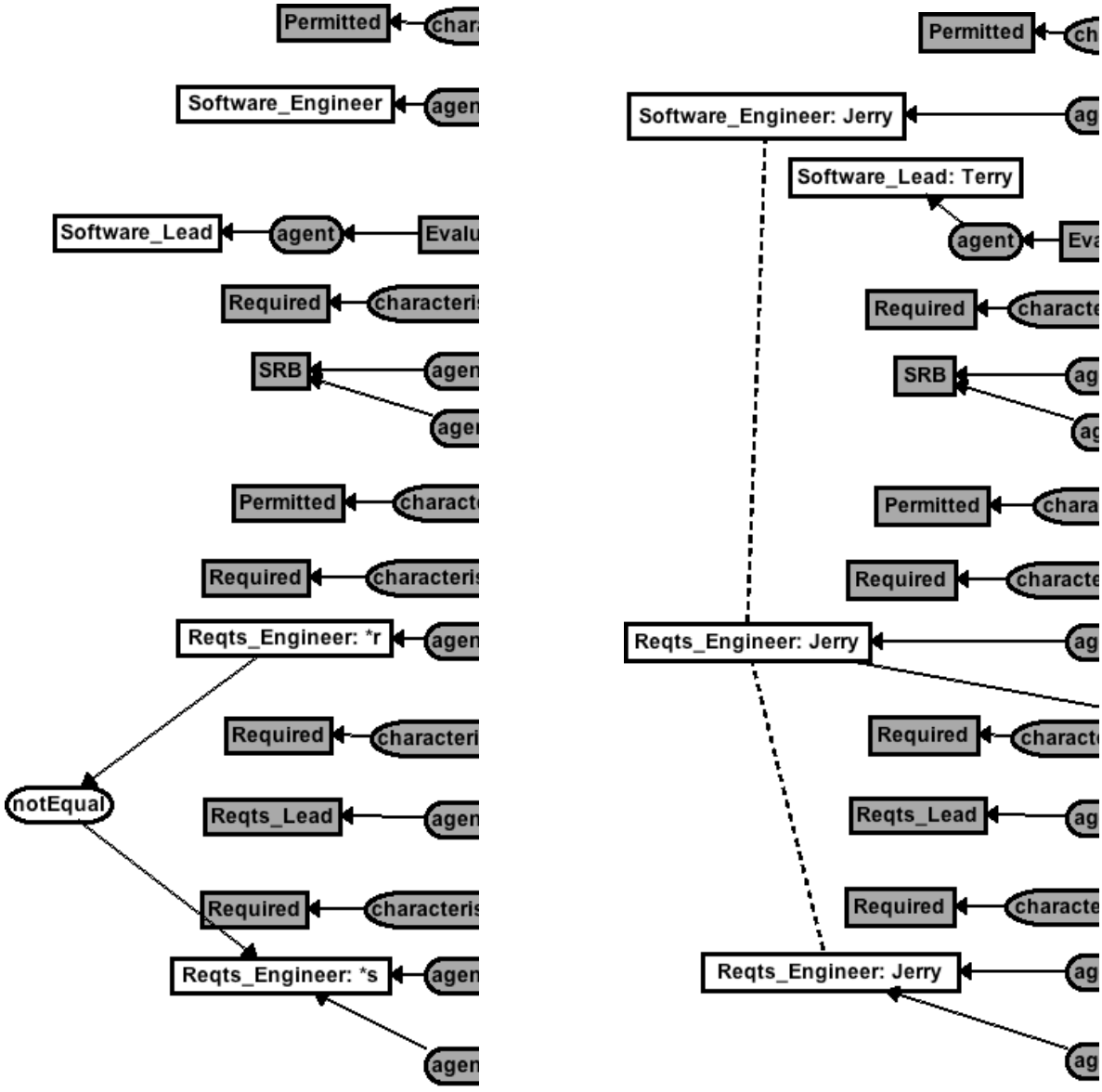
4.2 Interpreting the difference sequence

In our example, we determined two differences between the process model graph and the practice model as follows; these summarize what is shown in Figure 10.

- The process model contains no individual concepts, whereas the practice model shows individuals for some roles in the model. This reflects a natural tendency for practitioners to explain a process by identifying individual developers who are involved in the various roles.

The process model shows a **notEqual** relation between the requirements engineer who writes the requirement and the requirements engineer who incorporates the requirement into the SRS document. In the practice model, the requirements

engineer who produces the requirement (we call them ***r**) is also the same person (called ***s**) who incorporates the requirement into the SRS, a situation that is forbidden (through the **notEqual** relation) in the process model. The separation of roles (i.e., the explicit **notEqual** relation) in the process model represents an explicit prohibition, whereas in the practice model the separation between duties shown in the process model is lacking. Which model should be changed? Or should both be changed? In this final stage of the technique, it is up to actual participants such as the software managers and developers to interpret and analyze the comparisons.



(a) (b)

Figure 10. Highlighted Differences Between G1 and G2.

5 Discussion and Conclusion

For some readers, these kinds of operations take some getting used to, since they do not preserve truth characteristics. It is clear in our example that if we generalize **T:22** to **T** and then back to **T:34**, the two numbers are distinctly different and incompatible. As we mentioned earlier, our intent is to transform one graph into

another graph that is explicitly different; therefore they say different things and have different truth characteristics.

Based on the canonical graph set or existing definitions, some "differences" may in fact be merely expansions or contractions of definitions, while still preserving truth. For example, expanding a concept out to reveal its complete definition will in effect specialize a graph; however, there is no semantic change, since the definition was implied anyway by the original concept.

In general, for any given pair of graphs, there is no unique *LCG* for any pair of graphs. Among all the candidates, is one of them more useful in certain circumstances or optimal (i.e., more efficient) than the others? This is an open question. Since our purpose in developing the differences has been to support human analysis, we are satisfied for the moment with a possibly sub-optimal result with respect to its computation.

A difference sequence not only describes the differences but it also may help in measuring the degree of difference between the graphs; i.e., in general the more difference steps in the sequence, the greater the difference between the two graphs. Of course, if two graphs are completely different (i.e., their LCG is the graph **[T]**) then the difference sequence might consist of two steps – delete the first graph (a "subgraph") and then insert the second graph; which would only be two steps, yet the graphs would be completely different! The question of measuring the differences requires much more study.

One difficulty with this approach occurs if we try to compare two different graphs that were composed using differing canonical bases or ontologies. Transformations of one graph into another can therefore result in a graph that would be incorrectly formed ("nonsense") with respect to the original graph. Since we do not claim to preserve truth in performing these operations, this may not present a logical problem, but in some applications, differing ontologies may interfere with human interpretations of the results. These problems are well-known in ontology research (e.g., [6], [7]) and addressed by [8].

An interesting exercise would be to perform the graph difference algorithms on the actual canonical bases or ontologies themselves: this might lead us to discover some ontological differences between the knowledge bases they are intended to support.

There are important issues of tractability and computability which can be addressed more effectively if we relax the requirement that solutions be unique or deterministic. Of course, this means that two different analysts may derive two distinct difference sequences; our approach allows for possible multiple interpretations.

We developed our notion in order to use it in an application of conceptual graphs, not just as an interesting theoretical point. As a result, we have developed novel uses for color in conceptual graphs, in addition to demonstrating the usefulness of graph differences in aiding human analysis of conceptual graph models.

6 Bibliography

[1] E. Salvat and M.-L. Mugnier, "Sound and Complete Forward and Backward Chainings of Graph Rules," in *Conceptual Structures: Knowledge Representation as Interlingua*, vol. 1115, *Lecture Notes in Artificial*

Intelligence, P. W. Eklund, G. Ellis, and G. Mann, Eds.: Springer Verlag, 1996, pp. 248-262.

[2] D. Corbett, *Reasoning and Unification over Conceptual Graphs*. New York: Kluwer Academic, 2003.

[3] J. F. Sowa, *Conceptual Structures: Information Processing in Mind and Machine*. Reading, Mass.: Addison-Wesley, 1984.

[4] G. Ellis, "Compiling Conceptual Graphs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 7, pp. 68-81, 1995.

[5] G. Ellis and F. Lehmann, "Exploiting the Induced Order on Type-Labeled Graphs for Fast Knowledge Retrieval," in *Conceptual Structures: Current Practices*, W. M. Tepfenhart, J. P. Dick, and J. F. Sowa, Eds. Berlin: Springer-Verlag, 1994, pp. 293-310.

[6] N. Guarino, "Formal Ontology, Conceptual Analysis and Knowledge Representation," *International Journal of Human-Computer Studies*, vol. 43, 1995.

[7] A. Farquhar, R. Fikes, W. Pratt, and J. Rice, "Collaborative Ontology Construction for Information Integration," Knowledge Sharing Laboratory, Stanford University, Technical Report KSL-95-63, August 1995 1995.

[8] M.-L. Mugnier and M. Chein, "Characterization and algorithmic recognition of canonical conceptual graphs," in *Conceptual Graphs for Knowledge Representation*, vol. 699, G. W. Mineau, B. Moulin, and J. F. Sowa, Eds. Berlin: Springer-Verlag, 1993, pp. 294-311.

Conceptual Graphs and Annotated Semantic Web Pages

Pavlin Dobrev[1] and Albena Strupchanska[2]

[1] ProSyst Labs EOOD, 48 Vladaiska Str., Sofia, Bulgaria
pavlin@prosyst.com
[2] Bulgarian Academy of Sciences, Institute for Parallel Information Processing, 25A Acad. G. Bonchev Str., 1113 Sofia, Bulgaria
albena@lml.bas.bg

Abstract. Semantic Web aims at turning Internet into a machine understandable resource, which requires the existence of ontologies, methods for ontology mapping and pages annotated with semantic markup. It is clear that the manual annotation of pages is not feasible and the fully automatic one is impossible so the current trend is creation of tools for semi-automatic page annotation. Platforms like KIM and SemTag automatically annotate pages with named entities and predefined relations but deeper annotation requires human intervention. In this paper we describe a prototype that visualizes annotations of web pages and assists the user in their verification and enrichment. The ordinary people don't know terms like ontology and OWL/RDF but they can easily understand visualization of ontology terms as a hierarchy and the respective assertions as conceptual graphs. Visualization of knowledge is always an important issue and tools that make this process more intuitive contribute to better semantic annotation and domain understanding.

1. Introduction

Semantic web aims at making web resources more meaningful to computers by adding a semantic layer to the existing presentation layer of web pages. This presupposes the existence of ontologies and pages annotated with respect to them. The issues concerning ontology availability, development and evolution as well as ontology mapping are not addressed in this paper. Rather, we emphasize on the annotation of web pages and the available tools.

The annotation is considered as one of the most effort consuming tasks and needs to be performed by especially developed tools. The volume of pages available on the web and the vagueness of natural language are the main reasons that make the manual annotation of pages not feasible even with the existence of many user-friendly tools. The current trend is semi-automatic annotation of web pages where named entities (NEs) and some simple relations between them are automatically annotated but more complex entities and relations are left to human annotators.

A prototype, which is presented in this paper, gives users the opportunity to interactively exploit the results of automatic page annotations and to add more

complex annotations. Since the production of pages annotated in similar manner requires standardization of different annotation practices, which are not present globally, our prototype addresses to particular users who will annotate pages for their specific purposes and applications. We envisage that our prototype can be successfully applied in the e-learning domain.

The paper is structured as follows. Section 2 sketches related work in the Semantic Web area and some developed tools. It also mentions attempts for the usage of conceptual graphs [16] in the new web generation. Section 3 makes overview of the prototype, its functionality and features. Section 4 and Section 5 describe in more details the prototype and give some examples for its possible applications. Our future plans and the conclusion are stated in Section 6.

2. Related Work

Many tools for ontology creation, visualization and annotation appear in the Semantic Web community. We will shortly overview some of them emphasizing on their features, which are similar or related to our prototype.

Protégé[14] is a tool for ontology creation and visualization. It offers an editing environment with several third party plug-ins. With Protégé users can construct domain ontology, customize data entry forms and enter new data. The ontology editor consist of two panels: the left one shows classes of the ontology and the right one – their respective slots (attributes and relationships). The slots have a predefined structure such as name, type, cardinality etc. and the user has to fill in forms in order to add or change something. A graphical visualization is provided by the OntoViz tab plug-in where the hierarchy is again on the left-side window and the right-side window visualizes selected class and its properties as an RDF graph.

CREAM[11] is a framework for markup, knowledge acquisition, annotation and authoring. CREAM's reference implementation is OntoMat, whose working window consists of two main panels: ontology guidance and fact browser (on the left) and document/editor viewer (on the right). The user can add instances to existing ontological classes, attributes and relationships by: *(i)* typing into the generated templates the needed information; *(ii)* markup i.e. dragging a markup piece of text to the particular class or to the corresponding entry in attribute's table or to the relation of pre-selected instance. OntoMat also provides authoring with content generation. Dragging an instance of a class/attribute/relationship from the ontology guidance and fact browser and dropping it into the document editor/viewer cause an addition of simple sentences into the document. Even though OntoMat is a user-friendly tool, the manual annotation is time-consuming and not scalable.

Manual selection of text fragments and their assignment to concepts in ontology strictly depend on the individual so the results are ambiguous. The factors mentioned above lead to the creation of tools for automatic metadata extraction.

SemTag [4] is an application that aims at bootstrapping the Semantic Web with performing automated semantic tagging of large corpora. The creators of the tool introduce a new algorithm called TBD (Taxonomy- Based Disambiguation), which is used for choosing the correct ontological label of a piece of text that is ambiguous.

For disambiguation they use a 10-word window to both sides of the label. The experiments show that although the 10-word window is typically sufficient to understand the sense of the label the human judgment regarding the placement of the label into the taxonomy remains highly ambiguous.

Knowledge and Information Management (KIM) platform [15] aims at automatic annotation of web resources and ontology population. It uses Information Extraction (IE) techniques provided in GATE to automatically annotate NEs in pages and populate the ontology. The acquired instances are stored in RDF/OWL repository (Sesame) and the annotated documents - in document store (Lucence). The KIM plug-in colours the annotated parts of the text with respect to a given ontology and it also provides browsing of annotations. Another component based on the IE system Amilcare [2], for semi-automatic creation of metadata has been realized in OntoMat too, but it is not available in its download version.

Our prototype presupposes the existence of ontologies and annotated pages. It offers visualization and editing of annotations. Its editing capabilities can be viewed as a supplement to the existing ones in the mentioned above tools. The visualization is based on page annotations and it provides some semantic services to the users. An approach that is realized in the tool Magpie [9, 10] is similar to ours in the sense that it utilizes the annotations as a base for semantic services.

Magpie provides semantic-based assistance in user's web page navigation. It is a framework supporting the interpretation of web pages that is integrated in the standard web browser. Instances on the web page are automatically annotated with the chosen ontology using the ontology-based lexicon approach. User interface components visualize entities found in the page and enable users to interact with the semantic services through contextual menus. Magpie services act as an auxiliary knowledge resource, which is at users' disposal.

The prototype presented in the paper uses Conceptual Graphs (CGs) for visualization of created annotations and for some reasoning. Several attempts for using CGs in the semantic web exist. Corese [3] is one of them and it is integrated in several applications.

Corese is an ontology-based search engine, which retrieves web resources annotated in RDF(S). A query is translated into RDF then into CG. The RDF annotations of web pages are also translated into CGs. In addition, the developers of Corese proposed a RDF Rule extension to RDF and some rules are applied before the query processing. In query processing the projection operation is used for projecting a query graph into annotation graphs. The projection operation is modified so that not only concepts, which are specialization of each others, are subsumable but also concepts that are close enough. Corese provides a user with approximate answers to queries. The semantic distance and approximation operators control that process. During the query Corese enables users to define which concepts can be approximated and which ones must be found exactly. For instance, the approximation of properties is by means of the Rdfs: seeAlso property.

The authors of [18] successfully apply CGs for mining transformation rules for semantic matching. They represent knowledge as CGs and propose an algorithm for discovering transformation rules between CGs describing semantically close assertions. The developed semantic matcher plays crucial role in the transformation search. It uses taxonomic knowledge from the ontology as a baseline for measuring

the distance. The authors also report results concerning the augmentation of the matcher with transformation rules, with human authored transformations and with their combination.

3. System Overview

Our prototype relies on the experience gained from previously developed tools ViSem [5] and CGWorld [6, 7, 8]. The main features that have been developed are in the area of semantic web annotation, visualization and editing and concern also the integration of conceptual graph formalism in the semantic web.

During the years we have implemented lots of functionality in CGWorld concerning visualization and editing of CGs, their representation in different notations and realization of CG operations. The current prototype benefits from all. In addition, we have successfully explored some Natural Language Processing (NLP) scenarios for automatic extraction of CGs from either sentences in restricted natural language [1] or sentences concerning a specific domain [17]. We apply these techniques in our prototype and the extracted CGs are added to the web pages' annotations for further use in some inference.

The prototype utilizes and relies on some resources and techniques developed in the semantic web community. For example the ontology processing is built on top of Jena's API. Jena [19] is a Java framework for building Semantic Web applications, which provides a programmatic environment for RDF, RDFS and OWL. It is an open source and grown out of work with the HP Labs Semantic Web Programme. Since the mapping between RDF and CGs is easy we use the Jena API for creation and manipulation of RDF graphs and modified it to ensure consistency of RDF graphs with the CGs formalism. Jena contains other implementations of its graph interface e.g one which stores its data in a Berkley DB database, and another which uses a relational database (in our case MySQL). So it is very suitable for maintenance of large databases and in particular for a database of CGs.

The main features of the prototype are implemented in Java using Swing graphical library. Conceptual graph operations and transformation of sentences into CGs are realized using SICStus Prolog.

4. Visualization of annotations

Concerning the visualization in Semantic Web, Kimani et al. [12] classifies existing approaches, where our approach would be sorted as generated rendering with direct manipulation interaction style. Our experimental tool provides visualization and navigation support.

When choosing a concept from the hierarchy semantic indices of the uploaded web page are calculated and text parts that are semantically anchored to the concept are highlighted. One option for visualization at this point is just to highlight the parts and the concept in both windows with the same colour. However, a natural way for connecting a web page to the ontology is to draw a line between phrases/word(s) in

the text and its corresponding ontological concepts from the graphical ontology view (see Fig.1). This way of visualization is similar to the one proposed by Ted Nelson in his Cosmic Book (http://xanadu.com/cosmicbook/). We consider this option as more intuitive in cases when the parts of the texts, which are semantically anchored to the concept from the hierarchy, are dispersed in the web page. Our prototype supports both options for visualization.

We believe that showing simultaneously a concept in the ontology hierarchy and its instances on the page could be very useful in learning scenarios. Since this way of visualization shows both the language context of a concept usage as well as its ontological environment it could be applied for supporting users'comprehension while reading web pages.

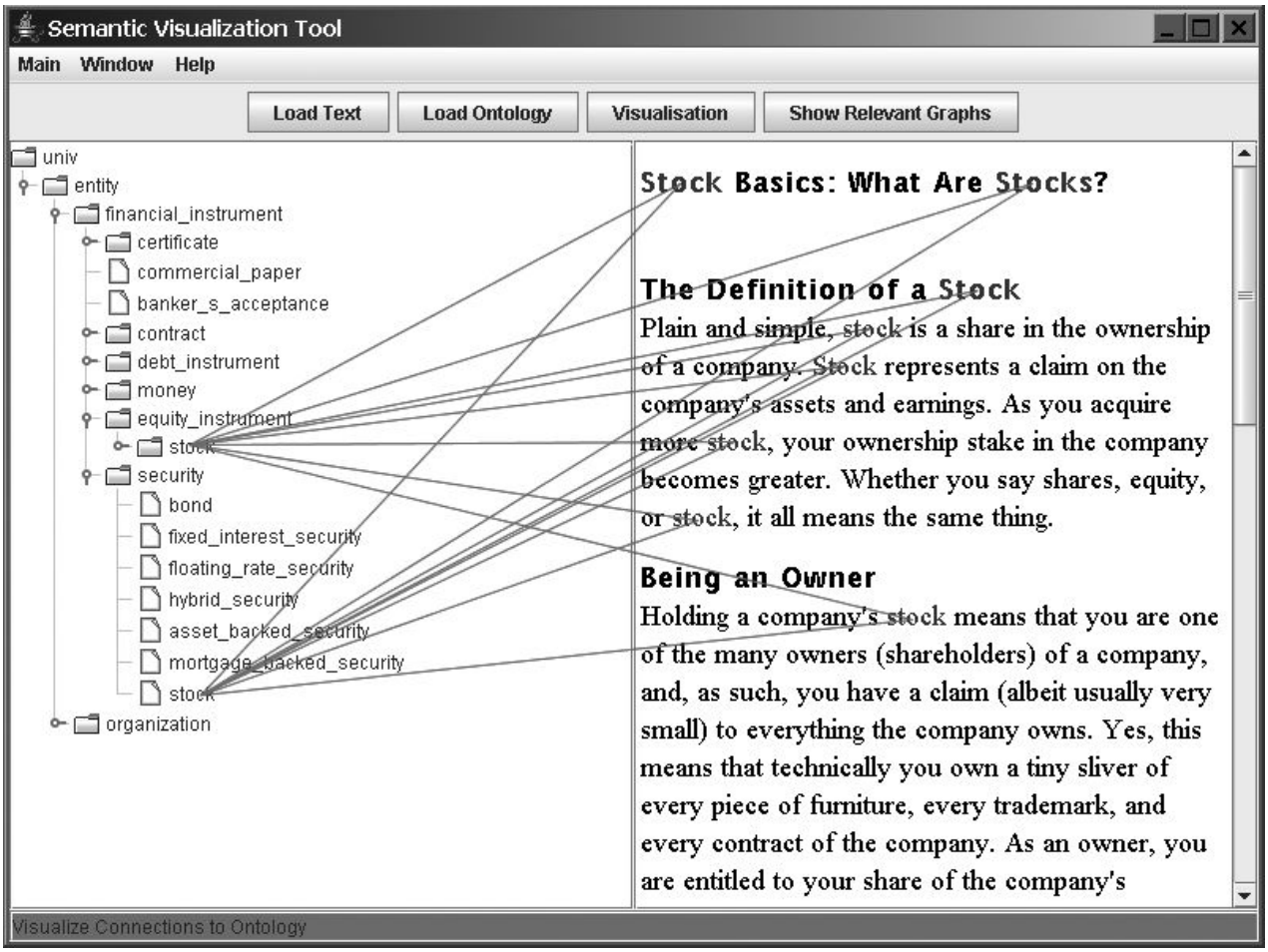


Fig. 1. Visualization of links between ontology and annotated page

Knowledge engineers would like to see visualization of concepts' properties and their relationships too. Many semantic web tools (for instance, see above the comments about Protégé) show such kind of information as RDF graph. We propose visualization as CG. On one hand the mapping between RDF/N3 and CG is relatively easy (see http://www.w3.org/DesignIssues/CG.html). Ordinary users can easily understand CGs. On the other hand CGs have querying and inference capabilities that can be further exploited. As shown in Fig. 2 a user can view assertions relevant to the chosen concept by clicking on the tab "Show Relevant Graphs". These graphs are extracted from the knowledge base of conceptual graphs, which has been developed for a previous project and extended by the usage of the current prototype.

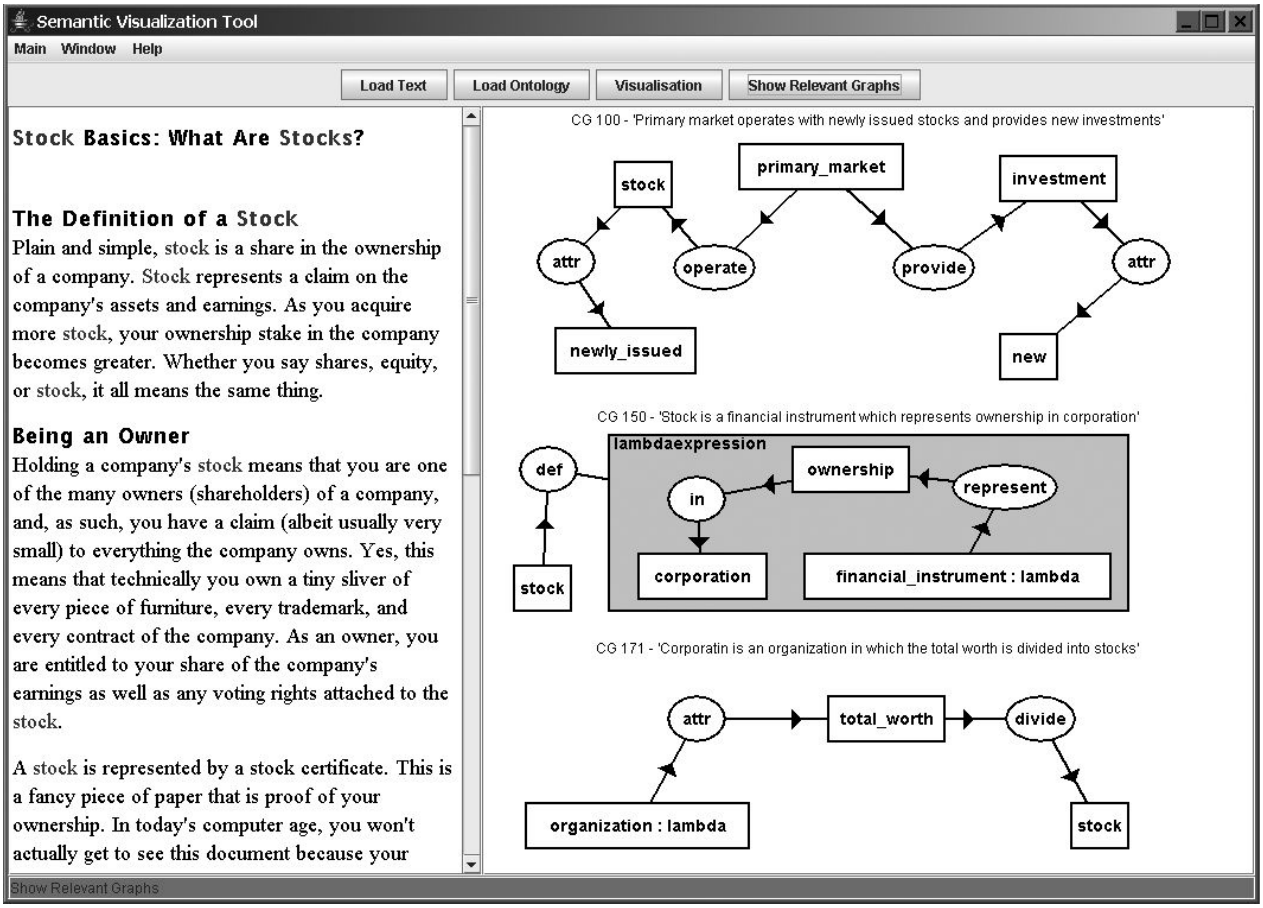


Fig. 2. Show relevant graphs

5. Annotations Editor

Users have the opportunity to edit annotations in all the tools for manual annotation of pages that are developed in the Semantic Web. They can freely choose the ontology and the assignment of words/phrases found in pages to the concepts of this ontology. The concept/individual properties are predefined and they can be included in annotation by filling in forms. The tools that use automatic processing to extract annotations of named entities allow visualization only but not editing capabilities. Our prototype presupposes the usage of an output of such tools and further provides more functional capabilities to users.

5.1. Automatic extraction of annotations

We propose the user to highlight sentence in a web page and to extract automatically a CG from it. A technique described in more details in [17] has been integrated in our prototype. We found this technique very suitable for application in a web-based content. First, it uses GATE [20] for preprocessing which has a relatively big lexicon. Second, it applies a partial parsing to produce logical forms (further converted into

CGs). For the highly varied text structure and style in the web only partial parsing can be successfully applied

A right click on the highlighted sentence triggers the appearance of a context menu with several choices. When pressing "CG Extract" a graph is extracted and it is loaded in the right window (see Fig. 3). Two possibilities are further proposed to the user.

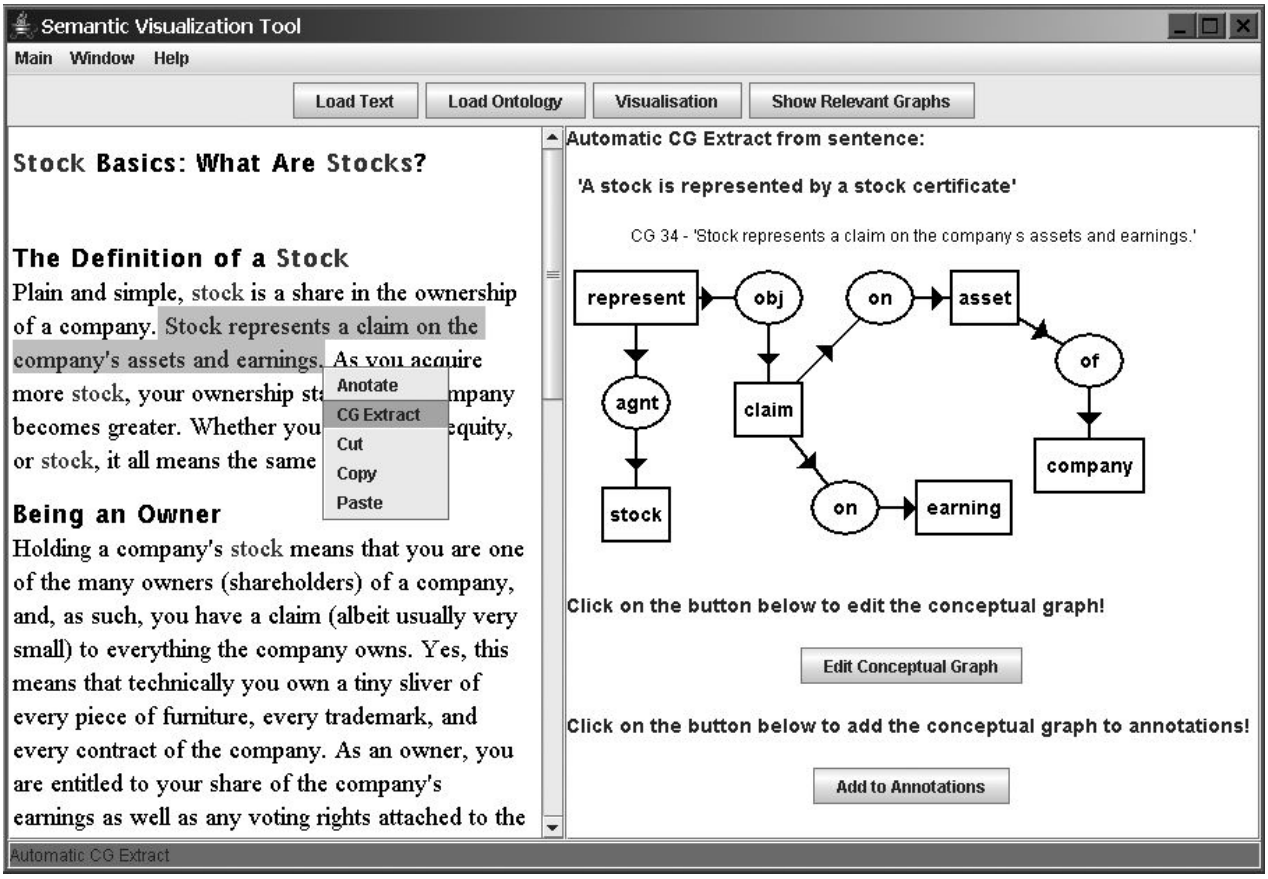


Fig. 3. Extract Conceptual Graph from a given sentence

5.2. Personalization of annotations

If a user is familiar with CG presentation formalism s/he can choose to edit the derived CG. The edit option gives him/her a possibility to add some missing parts to the CG, which are not returned as a result from the partial parsing but the user considers them as important.

Conceptual Graph Editing mostly reuses the functionality developed for the CGWorld. Concepts, relations, arcs, co-referent links and contexts are supported for editing via a simple Drag & Drop interface. Fig. 4 shows the main window of the editor. It is very easy via the simple click to add new concept, relation or to draw directly an arc between conceptual objects. An important functionality is the ability to assign any number of additional properties to the conceptual objects. These properties are related mainly to conceptual objects that represent concepts in CGs. Some possible properties are number, individual name or marker, qualifier, designator, comment etc. There is no limitation about which properties could be assigned to a

conceptual object. Such additional properties can be added and used during the annotation of web page in order to allow more knowledge information to be included and automatically processed afterwards. The editor has an excellent zooming capability. After zooming new positions of the objects are visualized i.e all dimensions are re-computed. This feature is very useful for editing large conceptual objects.

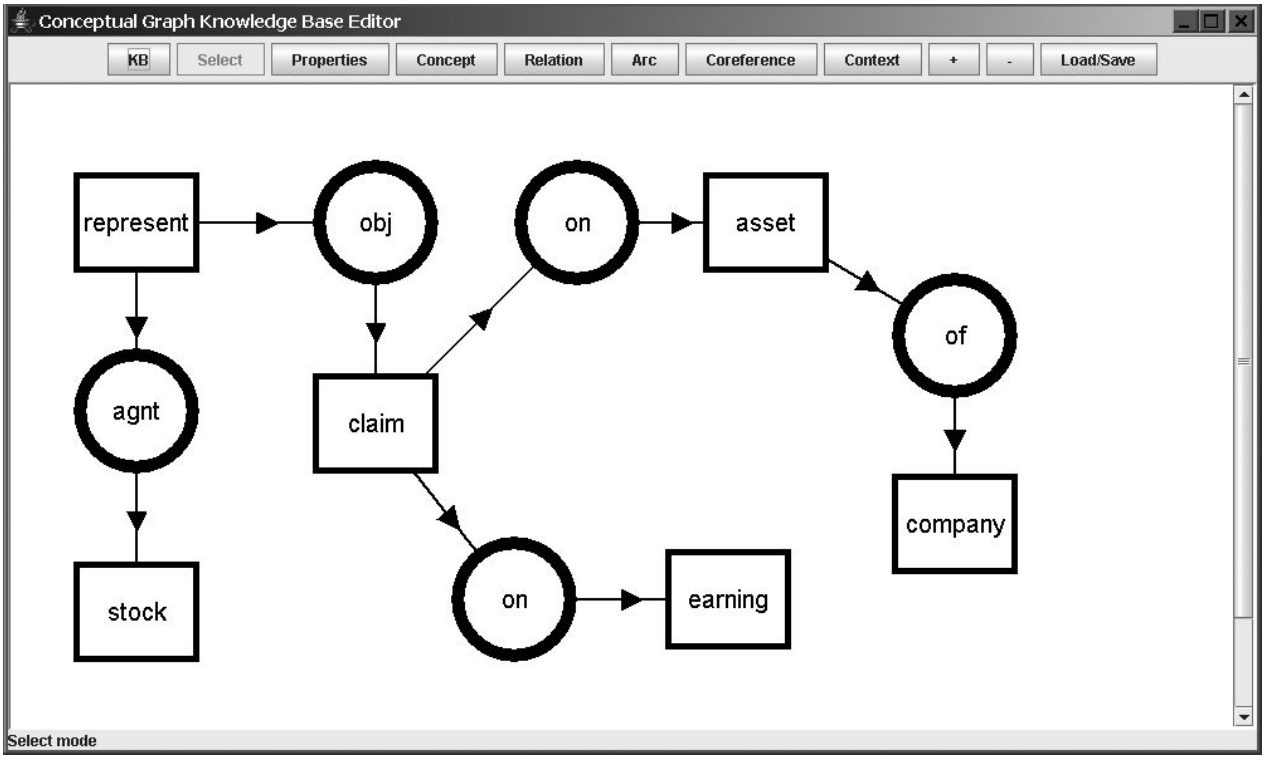


Fig. 4. Conceptual Graph editor

The application scenario we have in mind for extracting and editing of CGs in a prototype is again in the e-learning domain where teachers can choose important sentences and enrich web pages annotations.

The automatically extracted CGs might look rather complicated and not so easily understandable by a user who is non-familiar with CG formalism. As a result of parsing, all verbs are presented as concepts and all their valency roles (also called thematic roles) as relations. This kind of representation makes the reading of a CGs difficult due to the following reasons:

- a user has to know the semantics of valency roles;
- the size of CGs grows rapidly with the number of verbs and prepositions therefore capturing the meaning of a CG becomes quite effort consuming.

We have decided to use the type contraction operation in order to propose more readable CGs to the users. The previous realization of this operation takes as input a graph ID and a concept for which a type definition exists. In our case concepts are not known in advance moreover more than one concept can be replaced by respective definitions in one CG. The algorithm for type contraction has been modified to find suitable concepts and to apply contraction operation on several concepts in a CG.

Correctly identifying the semantic roles in a sentence is crucial for its interpretation. Moreover in order to develop more general natural understanding systems, broad coverage semantic resources have been created in several projects e.g. FrameNet (http://www.icsi.berkeley.edu/~framenet/) and PropBank [13]. Such resources describe a word, especially a verb by the number of arguments and their syntactic categories that can be filled by other words which are connected with the described word. Some of them contain also the semantic categories of those constituents. For instance, looking at the verb "sell" in a PropBank the following information can be found:

SELL - [*Arg0*: ***seller***]
[*Arg1*: ***thing sold***]
[*Arg2*: ***buyer***]
[*Arg3*: ***price paid***]
[*Arg4*: ***benefactive***]

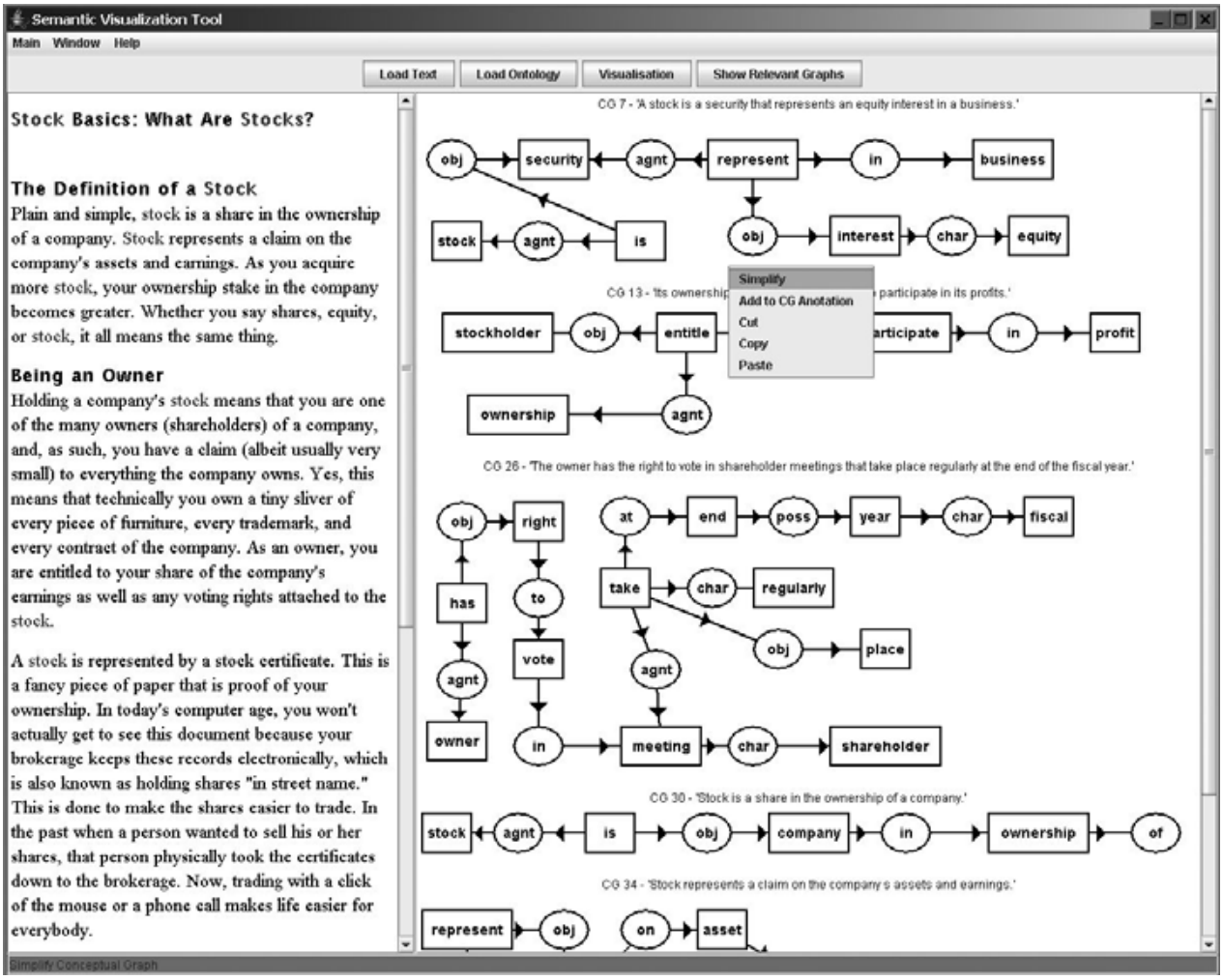


Fig. 5. Simplify Conceptual Graph

Some information that relates these roles with their syntactic categories (e.g. "*Arg0* is an *agnt")* can be found too. The example mentioned above has the purpose to show that such information about words exists. Having this information, the type definitions of words i.e their conceptual structures can be extracted. Our verbs' definitions and their roles are based on PropBank and a valency dictionary.

In [16] Sowa proposed a classification of thematic roles and types of their participants. Such classification makes reasoning more effective as specialization and generalization can be applied not only to concepts but to relations too. We plan to use this classification in our prototype when making reasoning on the CGs in the annotated pages.

Fig. 5 shows a user interface to the type contraction operation. It is very simple and intuitive. When a user clicks on a graph, a context menu appears. Choosing the option "simplify" s/he will receive a simplified version of the graph. The result is shown on Fig. 6.

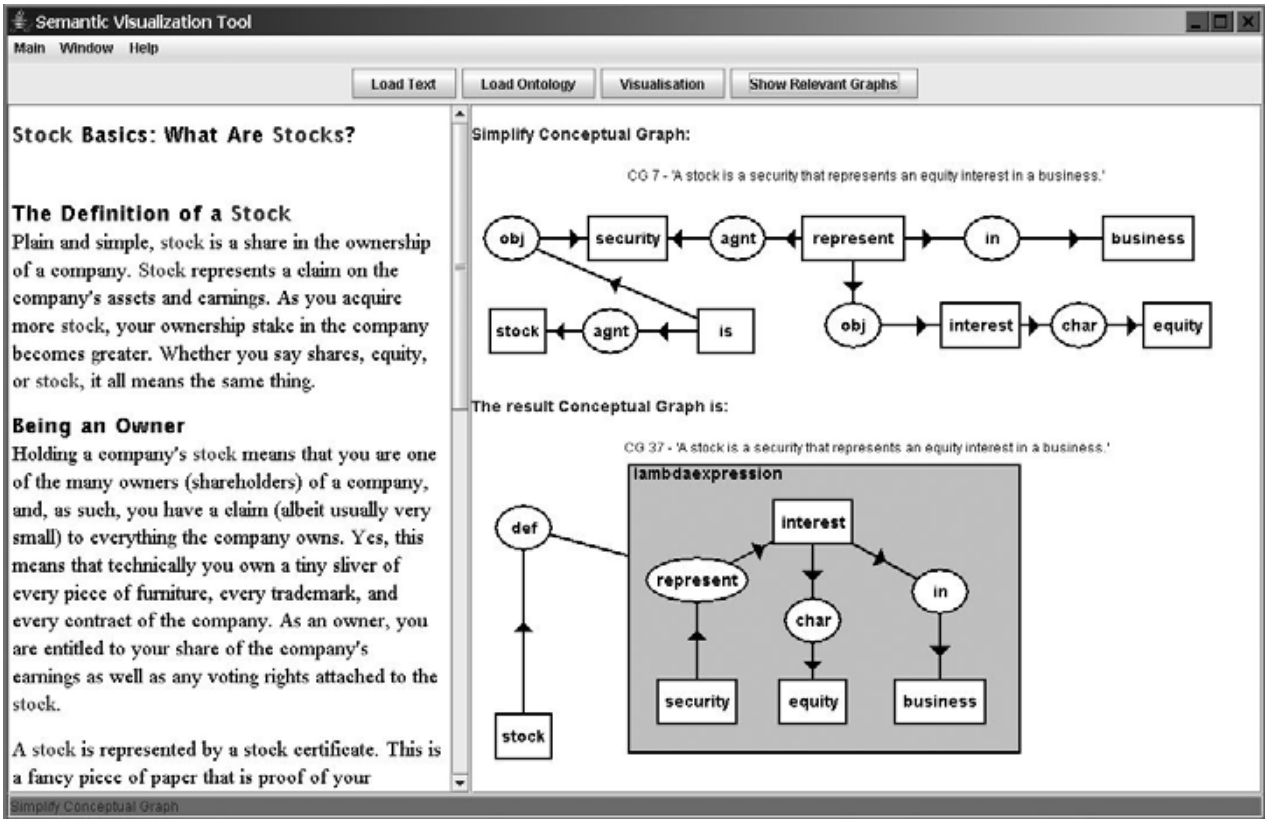


Fig. 6. Result from simplify Conceptual Graph

At the moment we have implemented type contraction on very simple definitions. Most of them have the following structure:

typedef *"verb"* is

[AgentType] <- (agnt) <- [verb] -> (obj) -> [ObjectType].

The example on Fig. 6 concerns the type contraction of the verb "be" used for definition of something. This verb and its synonyms form a special group to which additional rules are applied after performing the type contraction operation. An additional step is required because we have chosen to represent a definition of a concept as:

[Concept] - > (def) -> [Concept: CG].

The second concept is a complex one and shows particular restrictions on a genus of the defined concept.

6. Conclusion and Further Work

Manually annotating large amounts of pages is an arduous work. To master this process, techniques for semi automatic annotations are currently under development. The Semantic Web community realizes that NLP approaches are crucial for the success of the new web generation and such approaches are integrated in recently developed tools. IE techniques have been successfully applied so far for annotation of individuals and ontology population. Our prototype shows a possible subsequent step in the acquisition of knowledge structures from textual content of web pages.

The proposed way for visualization i.e. the visual connections between ontology and text parts, can be extended to cover the conceptual graph visualization. The other possible extension in this area is not only to show the connections between entities but also to allow visualizing of other relevant information like concept properties, annotation information, etc. Our research in the area of better visualization of semantic web knowledge will continue in this direction.

The integration of CGs in the web pages annotation enables better visualization and easy annotations' editing and enrichment. The inference capabilities of CG formalism together with the semantic web resources and technology can be applied to extend the current semantic search.

In contrast to the existing annotation tools, our prototype:

- supports automatic extraction of elaborated knowledge structures;
- allows annotators to edit the extracted formal structures;
- visualizes assertions about concepts in a way, which is intuitive for understanding.

The main application of the prototype is to assist annotators in the annotation process making it easier and allowing deeper annotation. We also believe that it can serve as a base for a practical application that benefits from deep semantic annotations and allows exploitation of different scenarios.

Acknowledgement

We would like to thank to our advisor Galia Angelova for her consistent support, guidance and encouragement during the years.

The work presented in this paper is partially supported by the project BIS 21++ funded by the European Commission in FP6 INCO via grant 016639/2005.

Reference

[1] Boytcheva, Sv., Dobrev, P. and G. Angelova. *CGExtract: towards Extraction of Conceptual Graphs from Controlled English*. In: G. Mineau (Ed.), Conceptual Structures: Extracting and Representing Semantics, Contributions to ICCS-2001, the 9th Int. Conference on Conceptual Structures, Stanford, California, August 2001, pp. 89-116.

[2] Ciravegna, F., Dingli, A., Petrelli, D. and Yorick Wilks: *Document Annotation via Adaptive Information Extraction*. Poster at the 25th Annual International ACM SIGIR Conference on

Research and Development in Information Retrieval August 11-15, 2002, in Tampere, Finland.

[3] Corby, O., Dieng-Kuntz, R. and Catherine Faron-Zucker: Querying the Semantic Web with Corese Search Engine. ECAI 2004: 705-709

[4] Dill, S., Eiron, N., Gibson, D., Gruhl, D., Guha, R., Jhingran,, A., Kanungo, T., Rajagopalan, S., Tomkins, A., Tomlin, J. and Jason Zien. *SemTag and Seeker: bootstrapping the semantic web via automated semantic annotation*. In Proceedings of the Twelfth International Conference on World Wide Web, 2003

[5] Dobrev, P., Strupchanska, A. and G. Angelova. *Towards a Better Understanding of the Language Content in the Semantic Web*. Lecture Notes in Artificial Intelligence, Volume 3192, Aug 2004, pp. 267-276

[6] Dobrev P.and K. Toutanova, *CGWorld - Architecture and Features*, ICCS 2002, Borovets, Bulgaria, July 2002, Lecture Notes in Computer Science 2393 Springer 2002, ISBN 3-540-43901-3

[7] Dobrev, P., Strupchanska, A. and K. Toutanova. *CGWorld - from Conceptual Graph Theory to the Implementation* , ICCS 2002 Workshop, July 2002, Borovets, Bulgaria, http://www.lml.bas.bg/iccs2002/acs/CGWorld2002.pdf.

[8] Dobrev, P., Strupchanska, A. and K. Toutanova. *CGWorld-2001 - new features and new directions*, ICCS 2001 Workshop, July 2001, Stanford University, USA http://www.cs.nmsu.edu/~hdp/CGTools/proceedings/papers/CGWorld.pdf

[9] Domingue, J.B.Dzbor, M., Motta, E. *Collaborative Semantic Web Browsing with Magpie*, In Proc. of the 1st European Semantic Web Symposium (ESWS), May 2004, Greece

[10] Dzbor, M., Domingue, J. and Motta, E. Magpie - Towards a Semantic Web Browser. 2nd International Semantic Web Conference (ISWC2003) 20-23 October 2003, Sundial Resort, Sanibel Island, Florida, USA http://kmi.open.ac.uk/projects/magpie/main.html

[11] Handschuh, S. and S. Staab. *CREAM: CREAting Metadata for the Semantic Web*. Computer Networks: The International Journal of Computer and Telecommunications Networking, Volume 42 , Issue 5, 2003, pp. 579 – 598

[12] Kimani, St., Catarci, T., and I. Cruz. *Web Rendering Systems: Techniques, Classification Criteria and Challenges*. In Vizualizing the Semantic Web Geroimenko, V.Chen Ch. (Eds.), Berlin: Springer 2002, pp. 63 – 89.

[13] Kingsbury, P. and M. Palmer. *From Treebank to PropBank*. 2002. In Proceedings of the 3rd International Conference on Language Resources and Evaluation (LREC-2002), Las Palmas, Spain.

[14] Noy, N., Sintek, F., Decker, M., S., M., R. W., Fergerson and M. Musen. *A. Creating Semantic Web Contents with Protege-2000*. IEEE Intelligent Systems 16(2):60-71, 2001.

[15] Popov, B., Kiryakov, A., Ognyanoff, D., Manov D. and A. Kirilov. *KIM - a semantic platform for information extraction and retrieval*. Journal of Natural Language Engineering, Vol. 10, Issue 3-4, Sep 2004, pp. 375-392, Cambridge University Press

[16] Sowa, J. Conceptual Structures: *Information Processing in Mind and Machine*, Addison-Wesley, Reading, MA, 1984

[17] Strupchanska, A., Yankova, M. and Sv. Boytcheva. *Conceptual Graphs Self-Tutoring System*, In Proc. ICCS 2003, July 2003, LNAI 2746, Dresden, Germany, pp. 323-336

[18] Yeh, P., Porter, B., and Ken Bakker. *Mining Transformation Rules for Semantic Matching*. Proceedings of the Workshop on Mining Graphs, Trees and Sequences (MGTS'04). Pisa, 83-94.

[19] JENA, see http://jena.sourceforge.net

[20] GATE, see http://gate.ac.uk/

Negations in Description Logic – Contraries, Contradictories, and Subcontraries

Ken Kaneiwa

National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan
kaneiwa@nii.ac.jp

Abstract. In [10], several constructive description logics were proposed as intuitionistic variants of description logics, in which classical negation was replaced by strong negation as a component to treat negative atomic information. For conceptual representation, not only strong negation but also combining it with classical negation seems to be useful and necessary due to their respective features corresponding to predicate denial (e.g., not happy) and predicate term negation (e.g., unhappy). In this paper, we propose an alternative description logic $\mathcal{ALC}_{\sim}$ with classical negation and strong negation. In particular, we adhere to the notions of contraries, contradictories, and subcontraries (in [5]), generated from conceivable statement types using predicate denial and predicate term negation. To capture these notions, our formalization provides an improved semantics that suitably interprets various combinations of classical negation and strong negation. We show that our semantics preserves contradictoriness and contrariness for $\mathcal{ALC}_{\sim}$-concepts but the semantics of constructive description logic $\mathcal{CALC}_{\sim}$ with Heyting negation and strong negation cannot preserve the property for $\mathcal{CALC}_{\sim}$-concepts.

1 Introduction

Negative information plays an important role in knowledge representation and reasoning (cf. [16, 8, 18]). Classical negation $\neg F$ represents the negation of a statement F, but a strong negation $\sim F$ may be more suitable for expressing explicit negative information (or negative facts). In other worlds, $\sim F$ indicates information that is directly opposite and exclusive to F rather than its complementary negation. Therefore the law of contradiction $\neg(F \wedge \sim F)$ holds but the law of excluded middle $F \vee \sim F$ does not hold [9, 1, 16]. For example, given the formula $rich(x)$ that represents "x is rich," the antonymous formula $poor(x)$ is defined by the strong negation $\sim rich(x)$, and not by the classical negation $\neg rich(x)$. Thus, we can logically recognize the inconsistency of $rich(x)$ and $\sim rich(x)$ (as $poor(x)$), and because $rich(x) \vee \sim rich(x)$ is not valid, we can represent "x is neither rich nor poor" as $\neg rich(x) \wedge \neg\sim rich(x)$.

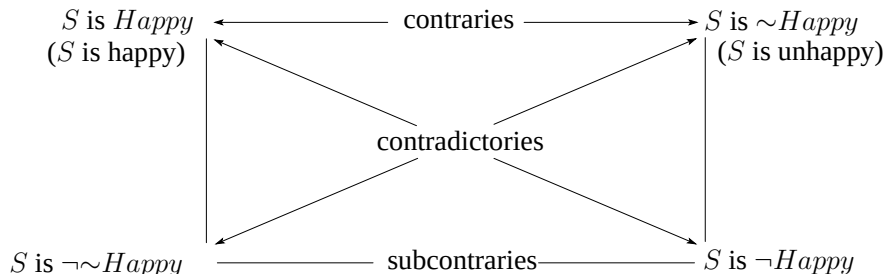
In [10], several constructive description logics were proposed as intuitionistic variants of description logics, in which classical negation was replaced by strong negation as a component to treat negative atomic information. In contrast, since basic description logics correspond to a subclass of *classical* first-order logic (i.e., they have classical negation), negative concepts are expressed by classical negation. Due to the different

features of negative information, complex negative statements using strong negation and classical negation can be usefully employed in conceptual representation. In the philosophical study of negation, there is a distinction between predicate denial (e.g., not happy) and predicate term negation (e.g., unhappy) [5, 18]. Moreover, conceivable statement types using predicate denial and predicate term negation give rise to opposition between affirmation and negation (contraries, contradictories, and subcontraries) [5]. When we logically establish classical negation and strong negation in concept languages, such philosophical notions are a rather suitable exposition of formalization.

In this paper, we propose a description logic $\mathcal{ALC}_{\sim}$ extended to include the two types of negation (classical negation $\neg$ and strong negation $\sim$), that is, an extension of the basic description logic $\mathcal{ALC}$. The following are the primary results of this paper. First, we present an improved semantics of $\mathcal{ALC}_{\sim}$, which adheres to oppositions – contraries, contradictories, and subcontraries in concept languages. We remark that the conventional semantics of strong negation [1] yields the undesirable equivalence $C \equiv \sim\neg C$ as opposed to our proposed semantics. Secondly, we show that our semantics preserves the property of contradictoriness and contrariness that there exists an element such that it belongs to the contradictory negation $\neg C$ but it does not belong to the contrary negation $\sim C$. When considering Heyting negation and classical negation in constructive description logics, the property cannot be preserved for some interpretations. The disadvantage motivates us to formalize a new semantics for contradictory negation and contrary negation. Based on the semantics, we develop a tableau-based algorithm for testing concept satisfiability in $\mathcal{ALC}_{\sim}$ and show the correctness (soundness, completeness, and termination) and complexity of the algorithm.

2 Contradictories and contraries between concepts

Strong negation can be used to describe conceptual oppositions in description logics such as the concepts $Happy$ and $Unhappy$. For example, let us denote by $Happy$, $\neg Happy$ (classical negation), and $\sim Happy$ (strong negation) the concepts "individuals that are happy," "individuals that are not happy," and "individuals that are unhappy," respectively. We can then construct the complex concepts $\exists has\text{-}child.\neg Happy$ as "Parents who have children that are not happy," $\exists has\text{-}child.\sim Happy$ as "Parents who have unhappy children," and $(\neg Happy \sqcap \neg\sim Happy) \sqcap Person$ as "Persons who are neither happy nor unhappy." Syntactically, these allow us to express concepts composed of various combinations of classical negation and strong negation, e.g., $\sim\neg C$ and $\sim\neg\sim C$. As discussed in [12], the two negations represent the following oppositions between affirmation and negation (which Horn [5] renders):



In our conceptual explanation of them, the contraries ($Happy$ and $\sim Happy$) imply that both concepts cannot contain an identical element. The contradictories ($Happy$ and $\neg Happy$) imply that one concept must contain an element when it does not belong to the other. The subcontraries ($\neg\sim Happy$ and $\neg Happy$) imply that every element belongs to either of the concepts. In order to apply the oppositions to any DL-concepts, we require to generalize them as follows [1]:

$$
\begin{array}{ll}
(\neg\sim)^i A, \sim(\neg\sim)^i A \text{ and } \neg(\sim\neg)^i A, (\sim\neg)^{i+1} A: & \text{contraries} \\
(\sim\neg)^i A, \neg(\sim\neg)^i A \text{ and } \sim(\neg\sim)^i A, (\neg\sim)^{i+1} A: & \text{contradictories} \\
(\neg\sim)^{i+1} A, \neg(\neg\sim)^i A \text{ and } \neg(\sim\neg)^{i+1} A, (\sim\neg)^i A: & \text{subcontraries}
\end{array}
$$

where A is a concept name (i.e., an atomic concept). In reasoning algorithms for description logics with the two negations, contraries and contradictories will be taken as a criterion of checking inconsistent pairs of DL-concepts.

In intuitionistic logic, Heyting negation and strong negation exist as methods of dealing with the oppositions. In general, strong negation has been formalized as a constructive negation of intuitionistic logic. Thomason [14] proposed the semantics of intuitionistic first-order logic only with strong negation, Akama [1] formalized intuitionistic logic with Heyting negation and strong negation, Wagner [15] and Herre et al.[3] defined weak negation and strong negation in the logic developed by them, and Pearce and Wagner [11] proposed logic programming with strong negation that was regarded as a subsystem of intuitionistic logic. While intuitionistic logic and strong negation allow us to represent term negation and predicate denial, we would like to propose a description logic such that:

1. It contains classical negation and strong negation since $\mathcal{ALC}$ is based on classical logic.
2. It fulfills the property that contradictoriness and contrariness are preserved for every interpretation.

Here, we observe the properties of Heyting negation ($-$) and strong negation ($\sim$). Let F be a formula (but not a concept). The law of double negation $--F \leftrightarrow F$ and the law of excluded middle $-F \vee F$ do not hold for Heyting negation, but the law of double negation $\sim\sim F \leftrightarrow F$ is valid for strong negation. The combinations of these negations lead to the valid axiom $\sim -F \leftrightarrow F$. Hence, the contradictory F and $-F$ can be replaced with $\sim -F$ and $-F$ by the equivalence $\sim -F \leftrightarrow F$; however, it should be recognized as a contrary. This is in partial disagreement with the abovementioned oppositions in which contradictories and contraries are defined differently. With regard to these features, Heyting negation and strong negation using the conventional semantics in intuitionistic logic [14, 1, 3] do not satisfy our requirements. Thus, we need to model contradictory negation and contrary negation in description logics and compare it with the constructive description logics in [10].

To incorporate strong negation into classical first-order logic and to remove the equivalence $\sim\neg F \leftrightarrow F$, we have improved the semantics by capturing the following

[1] $(\sim\neg)^i$ (resp. $(\neg\sim)^i$) denotes a chain of length i of $\sim\neg$ (resp. $\neg\sim$).

properties [7]. The law of double negation $\neg\neg F$ and the law of excluded middle $\neg F \vee F$ hold for classical negation, and the equivalence $\sim\neg F \leftrightarrow F$ is not valid. In conceptual representation based on this semantics, the strong negation $\sim C$ of a concept C is partial and exclusive to its affirmative expression C. The partiality of strong negation entails the existence of information that is neither affirmative nor strongly negative, i.e., $\neg C \sqcap \neg\sim C \not\equiv \bot$. In contrast, the classical negation $\neg C$ is complementary and exclusive to its affirmative expression C. Hence, the disjunction of affirmation and its classical negation expresses the set of all individuals, i.e., $C \sqcup \neg C \equiv \top$. Additionally, the simple double negations $\neg\neg C$ and $\sim\sim C$ are interpreted to be equivalent to the affirmation C. We can constructively define the complicated double negations $\sim\neg C$ and $\neg\sim C$ without losing the features of the two negations by the refinement of the conventional semantics. If we strongly deny the classical negation $\neg C$, then the double negation $\sim\neg C$ (called *constructive double negation*) must be partial and exclusive to $\neg C$. If we express the classical negation of a strong negation $\sim C$, then the double negation $\neg\sim C$ (called *weak double negation*) must be complementary and exclusive to $\sim C$.

3 Strong negation in description logic

In this section, we define a description logic with classical negation and strong negation that is interpreted by two different semantics and analyze the property of contradictoriness and contrariness for the proposed logic. In addition, we define a constructive description logic obtained by including Heyting negation and strong negation.

3.1 The description logic with classical negation and strong negation: $\mathcal{ALC}_\sim$

The description logic $\mathcal{ALC}_\sim$ (as an extension of $\mathcal{ALC}$ [13]) is based upon a set $\mathbf{C}$ of concept names A (including $\top$ and $\bot$), a set $\mathbf{R}$ of role names R, and a set $\mathbf{I}$ of individual names a. The concepts of the language (called $\mathcal{ALC}_\sim$-concepts) are constructed by concept names A; role names R; the connectives $\sqcap$, $\sqcup$, $\neg$ (classical negation), and $\sim$ (strong negation); and the quantifiers $\forall$ and $\exists$. Every concept name $A \in \mathbf{C}$ is an $\mathcal{ALC}_\sim$-concept. If R is a role name and C, D are $\mathcal{ALC}_\sim$-concepts, then $\neg C$, $\sim C$, $C \sqcap D$, $C \sqcup D$, $\forall R.C$, and $\exists R.C$ are $\mathcal{ALC}_\sim$-concepts.

We denote as $sub(C)$ the set of subconcepts of an $\mathcal{ALC}_\sim$-concept C. Let X be a sequence of classical negation $\neg$ and strong negation $\sim$. We denote $(X)^n$ as a chain of length n of X. For instance, $\sim(\neg\sim)^2 C_1$ and $(\sim\neg)^0 C_2$ denote $\sim\neg\sim\neg\sim C_1$ and C_2, respectively. Next, we define an interpretation of $\mathcal{ALC}_\sim$-concepts (called an $\mathcal{ALC}^2_\sim$-interpretation) by using the conventional semantics of strong negation.

Definition 1. *An $\mathcal{ALC}^2_\sim$-interpretation $\mathcal{I}$ is a tuple $(\Delta^\mathcal{I}, \cdot^{\mathcal{I}^+}, \cdot^{\mathcal{I}^-})$, where $\Delta^\mathcal{I}$ is a non-empty set and $\cdot^{\mathcal{I}^+}$ and $\cdot^{\mathcal{I}^-}$ are interpretation functions ($A^{\mathcal{I}^+} \subseteq \Delta^\mathcal{I}$, $A^{\mathcal{I}^-} \subseteq \Delta^\mathcal{I}$, $R^{\mathcal{I}^+} \subseteq \Delta^\mathcal{I} \times \Delta^\mathcal{I}$, and $a^{\mathcal{I}^+} \in \Delta^\mathcal{I}$) such that:*

1. $\bot^{\mathcal{I}^+} = \emptyset$ *and* $\top^{\mathcal{I}^+} = \Delta^\mathcal{I}$,
2. $A^{\mathcal{I}^+} \cap A^{\mathcal{I}^-} = \emptyset$.

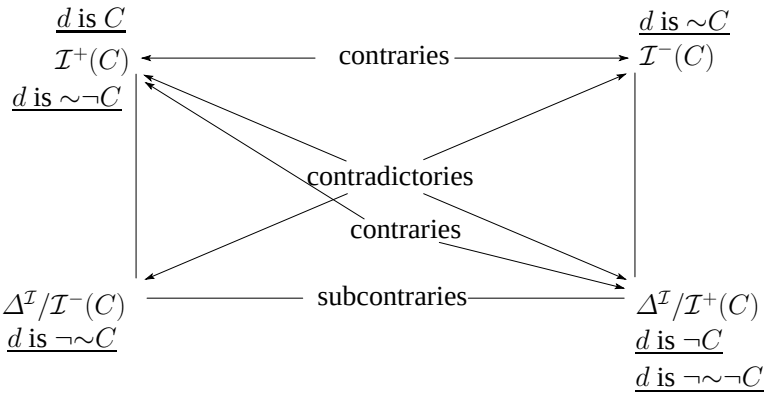


Fig. 1. Oppositions in $\mathcal{ALC}^2_\sim$-interpretations

The interpretation functions $\cdot^{\mathcal{I}^+}$ and $\cdot^{\mathcal{I}^-}$ are expanded to $\mathcal{ALC}_\sim$-concepts as follows:

$$
\begin{aligned}
(\neg C)^{\mathcal{I}^+} &= \Delta^{\mathcal{I}} \backslash C^{\mathcal{I}^+} & (\sim C)^{\mathcal{I}^+} &= C^{\mathcal{I}^-} \\
(C \sqcap D)^{\mathcal{I}^+} &= C^{\mathcal{I}^+} \cap D^{\mathcal{I}^+} & (C \sqcup D)^{\mathcal{I}^+} &= C^{\mathcal{I}^+} \cup D^{\mathcal{I}^+} \\
(\forall R.C)^{\mathcal{I}^+} &= \{d_1 \in \Delta^{\mathcal{I}} \mid \forall d_2[(d_1, d_2) \in R^{\mathcal{I}^+} \rightarrow d_2 \in C^{\mathcal{I}^+}]\} \\
(\exists R.C)^{\mathcal{I}^+} &= \{d_1 \in \Delta^{\mathcal{I}} \mid \exists d_2[(d_1, d_2) \in R^{\mathcal{I}^+} \wedge d_2 \in C^{\mathcal{I}^+}]\} \\
(\neg C)^{\mathcal{I}^-} &= C^{\mathcal{I}^+} & (\sim C)^{\mathcal{I}^-} &= C^{\mathcal{I}^+} \\
(C \sqcap D)^{\mathcal{I}^-} &= C^{\mathcal{I}^-} \cup D^{\mathcal{I}^-} & (C \sqcup D)^{\mathcal{I}^-} &= C^{\mathcal{I}^-} \cap D^{\mathcal{I}^-} \\
(\forall R.C)^{\mathcal{I}^-} &= \{d_1 \in \Delta^{\mathcal{I}} \mid \exists d_2[(d_1, d_2) \in R^{\mathcal{I}^+} \wedge d_2 \in C^{\mathcal{I}^-}]\} \\
(\exists R.C)^{\mathcal{I}^-} &= \{d_1 \in \Delta^{\mathcal{I}} \mid \forall d_2[(d_1, d_2) \in R^{\mathcal{I}^+} \rightarrow d_2 \in C^{\mathcal{I}^-}]\}
\end{aligned}
$$

An $\mathcal{ALC}^2_\sim$-interpretation $\mathcal{I}$ satisfies the contrary condition if for all concept names A, $A^{\mathcal{I}^+} \cup A^{\mathcal{I}^-} \neq \Delta^{\mathcal{I}}$. The $\mathcal{ALC}^2_\sim$-interpretation is defined by the two interpretation functions $\cdot^{\mathcal{I}^+}$ and $\cdot^{\mathcal{I}^-}$, but it causes an undesirable equation $C \equiv \sim\neg C$, i.e., $C^{\mathcal{I}^+} = (\sim\neg C)^{\mathcal{I}^+}$. Contradictoriness and contrariness are preserved in an $\mathcal{ALC}^2_\sim$-interpretation $\mathcal{I}$ of $\mathcal{ALC}_\sim$ if $\sim C^{\mathcal{I}} \subsetneq \neg C^{\mathcal{I}}$. The interpretation results in the following negative property:

Theorem 1 (Contradictoriness and contrariness for $\mathcal{ALC}^2_\sim$).
Contradictoriness and contrariness are not preserved in every $\mathcal{ALC}^2_\sim$-interpretation that satisfies the contrary condition.

Subsequently, we define an alternative interpretation of $\mathcal{ALC}_\sim$-concepts (called an $\mathcal{ALC}^n_\sim$-interpretation), which is based on the semantics [7] obtained by improving Akama's semantics [1].

Definition 2. *An $\mathcal{ALC}^n_\sim$-interpretation $\mathcal{I}$ is a tuple $(\Delta^{\mathcal{I}}, \{\cdot^{\mathcal{I}_i^+} \mid i \in \omega\}, \{\cdot^{\mathcal{I}_i^-} \mid i \in \omega\})$,* [2] *where $\Delta^{\mathcal{I}}$ is a non-empty set and $\cdot^{\mathcal{I}_i^+}$ and $\cdot^{\mathcal{I}_i^-}$ are interpretation functions ($A^{\mathcal{I}_i^+} \subseteq \Delta^{\mathcal{I}}$, $A^{\mathcal{I}_i^-} \subseteq \Delta^{\mathcal{I}}$, $R^{\mathcal{I}_0^+} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, and $a^{\mathcal{I}_0^+} \in \Delta^{\mathcal{I}}$), such that:*

[2] The symbol ω denotes the set of natural numbers. Thus, $\{\cdot^{\mathcal{I}_i^+} \mid i \in \omega\}$ is infinite as $\{\cdot^{\mathcal{I}_0^+}, \cdot^{\mathcal{I}_1^+}, \cdot^{\mathcal{I}_2^+}, \cdot^{\mathcal{I}_3^+}, \ldots\}$.

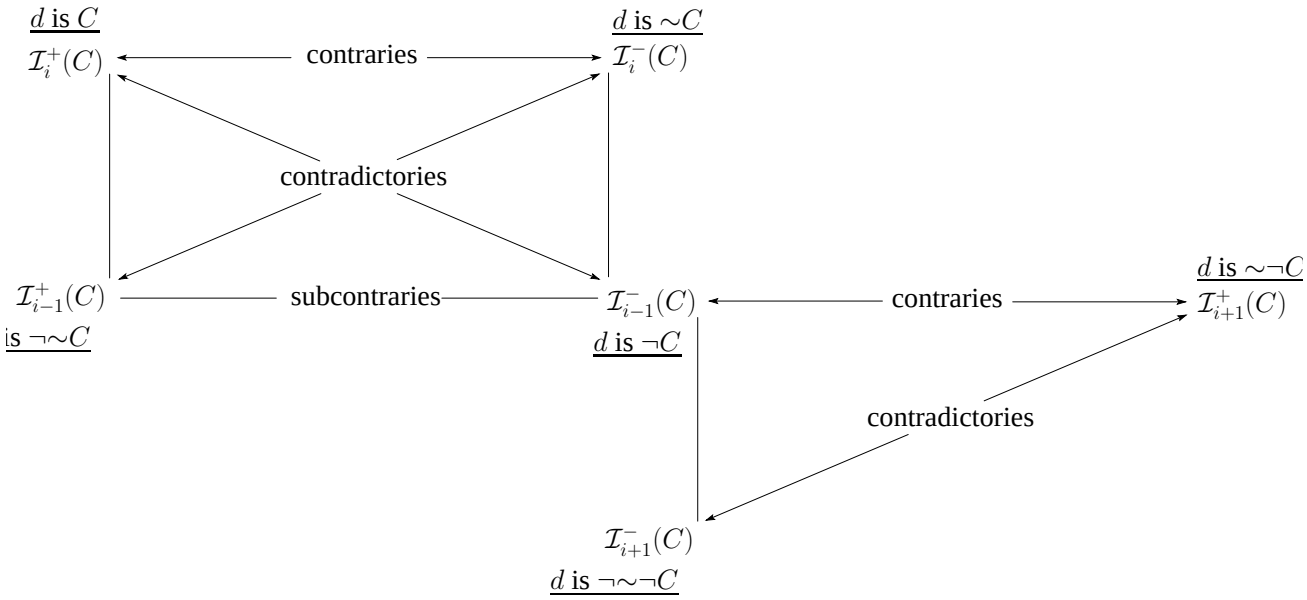


Fig. 2. Oppositions in $\mathcal{ALC}_{\sim}^{n}$-interpretations

1. $\bot^{\mathcal{I}_0^+} = \emptyset$ *and* $\top^{\mathcal{I}_0^+} = \Delta^{\mathcal{I}}$,
2. $A^{\mathcal{I}_0^+} \cap A^{\mathcal{I}_0^-} = \emptyset$,
3. $A^{\mathcal{I}_{i+1}^+} \subseteq A^{\mathcal{I}_i^+}$ *and* $A^{\mathcal{I}_{i+1}^-} \subseteq A^{\mathcal{I}_i^-}$.

The interpretation functions $\cdot^{\mathcal{I}_i^+}$ *and* $\cdot^{\mathcal{I}_i^-}$ *are expanded to* $\mathcal{ALC}_{\sim}$*-concepts as follows:*

$$
\begin{aligned}
(\neg C)^{\mathcal{I}_0^+} &= \Delta^{\mathcal{I}} \backslash C^{\mathcal{I}_0^+} \\
(\neg C)^{\mathcal{I}_i^+} &= C^{\mathcal{I}_{i-1}^-} (i > 0) & (\sim C)^{\mathcal{I}_i^+} &= C^{\mathcal{I}_i^-} \\
(C \sqcap D)^{\mathcal{I}_i^+} &= C^{\mathcal{I}_i^+} \cap D^{\mathcal{I}_i^+} & (C \sqcup D)^{\mathcal{I}_i^+} &= C^{\mathcal{I}_i^+} \cup D^{\mathcal{I}_i^+} \\
(\forall R.C)^{\mathcal{I}_i^+} &= \{d_1 \in \Delta^{\mathcal{I}} \mid \forall d_2 [(d_1, d_2) \in R^{\mathcal{I}_0^+} \rightarrow d_2 \in C^{\mathcal{I}_i^+}]\} \\
(\exists R.C)^{\mathcal{I}_i^+} &= \{d_1 \in \Delta^{\mathcal{I}} \mid \exists d_2 [(d_1, d_2) \in R^{\mathcal{I}_0^+} \wedge d_2 \in C^{\mathcal{I}_i^+}]\} \\
(\neg C)^{\mathcal{I}_i^-} &= C^{\mathcal{I}_{i+1}^+} & (\sim C)^{\mathcal{I}_i^-} &= C^{\mathcal{I}_i^+} \\
(C \sqcap D)^{\mathcal{I}_i^-} &= C^{\mathcal{I}_i^-} \cup D^{\mathcal{I}_i^-} & (C \sqcup D)^{\mathcal{I}_i^-} &= C^{\mathcal{I}_i^-} \cap D^{\mathcal{I}_i^-} \\
(\forall R.C)^{\mathcal{I}_i^-} &= \{d_1 \in \Delta^{\mathcal{I}} \mid \exists d_2 [(d_1, d_2) \in R^{\mathcal{I}_0^+} \wedge d_2 \in C^{\mathcal{I}_i^-}]\} \\
(\exists R.C)^{\mathcal{I}_i^-} &= \{d_1 \in \Delta^{\mathcal{I}} \mid \forall d_2 [(d_1, d_2) \in R^{\mathcal{I}_0^+} \rightarrow d_2 \in C^{\mathcal{I}_i^-}]\}
\end{aligned}
$$

An $\mathcal{ALC}_{\sim}^{n}$-interpretation $\mathcal{I}$ satisfies the contrary condition if for all concept names A, $A^{\mathcal{I}_0^+} \cup A^{\mathcal{I}_0^-} \neq \Delta^{\mathcal{I}}$, $A^{\mathcal{I}_{i+1}^+} \subsetneq A^{\mathcal{I}_i^+}$, and $A^{\mathcal{I}_{i+1}^-} \subsetneq A^{\mathcal{I}_i^-}$. In the two types of interpretations, conceptual oppositions are characterized as shown in Fig.1 and Fig.2. The $\mathcal{ALC}_{\sim}^{2}$-interpretation is defined as $(\sim\neg C)^{\mathcal{I}^+} = (\neg C)^{\mathcal{I}^-} = C^{\mathcal{I}^+}$, and hence, $d \in (\sim\neg C)^{\mathcal{I}}$ if and only if $d \in C^{\mathcal{I}}$. This semantically causes loss in distinction between contraries ($\sim\neg C$ and $\neg C$) and contradictories (C and $\neg C$). Instead, the $\mathcal{ALC}_{\sim}^{n}$-interpretation includes the definition $(\sim\neg C)^{\mathcal{I}_i^+} = (\neg C)^{\mathcal{I}_i^-} = C^{\mathcal{I}_{i+1}^+}$ and $(\neg C)^{\mathcal{I}_i^+} = C^{\mathcal{I}_{i-1}^-}$ $(i > 0)$, where infinite interpretation functions are required. That is, the $\mathcal{ALC}_{\sim}^{n}$-interpretation is

improved in order to capture the oppositions – contraries, contradictories, and subcontraries in the philosophical study of negation [5].

Each $\mathcal{ALC}^n_\sim$-interpretation (resp. $\mathcal{ALC}^2_\sim$-interpretation) $C^\mathcal{I}$ of each $\mathcal{ALC}_\sim$-concept is given by $C^{\mathcal{I}^+_0}$ (resp. $C^{\mathcal{I}^+}$). An $\mathcal{ALC}_\sim$-concept C (or a concept equation $C \equiv D$) is $\mathcal{ALC}^n_\sim$-satisfiable (resp. $\mathcal{ALC}^2_\sim$-satisfiable) if there exists an $\mathcal{ALC}^n_\sim$-interpretation (resp. $\mathcal{ALC}^2_\sim$-interpretation) $\mathcal{I}$, called an $\mathcal{ALC}^n_\sim$-model (resp. $\mathcal{ALC}^2_\sim$-model) of C (or $C \equiv D$), such that $C^\mathcal{I} \neq \emptyset$ (or $C^\mathcal{I} = D^\mathcal{I}$). Otherwise, it is $\mathcal{ALC}^n_\sim$-unsatisfiable (resp. $\mathcal{ALC}^2_\sim$-unsatisfiable). In particular, if an $\mathcal{ALC}_\sim$-concept C is $\mathcal{ALC}^n_\sim$-satisfiable (resp. $\mathcal{ALC}^2_\sim$-satisfiable) and the $\mathcal{ALC}^n_\sim$-model (resp. $\mathcal{ALC}^2_\sim$-model) satisfies the contrary condition, then it is $\mathcal{ALC}^n_\sim$-satisfiable (resp. $\mathcal{ALC}^2_\sim$-satisfiable) under the contrary condition. Otherwise, it is $\mathcal{ALC}^n_\sim$-unsatisfiable (resp. $\mathcal{ALC}^2_\sim$-unsatisfiable) under the contrary condition. A concept equation $C \equiv D$ is $\mathcal{ALC}^n_\sim$-valid (resp. $\mathcal{ALC}^2_\sim$-valid) if every $\mathcal{ALC}^n_\sim$-interpretation ($\mathcal{ALC}^2_\sim$-interpretation) $\mathcal{I}$ is an $\mathcal{ALC}^n_\sim$-model ($\mathcal{ALC}^2_\sim$-model) of $C \equiv D$. We can derive the following fact from these interpretations:

Proposition 1. *If C is an $\mathcal{ALC}_\sim$-concept, then the concept equation $C \equiv \sim\neg C$ is not $\mathcal{ALC}^n_\sim$-valid, but it is $\mathcal{ALC}^2_\sim$-valid.*

In addition, since $\mathcal{ALC}_\sim$-concepts do not contain the negation of roles, each role is interpreted only by the interpretation function $\cdot^{\mathcal{I}^+_0}$ (or $\cdot^{\mathcal{I}^+}$). Let us give an example of an $\mathcal{ALC}^n_\sim$-interpretation $\mathcal{I} = (\Delta^\mathcal{I}, \{\cdot^{\mathcal{I}^+_i} \mid i \in \omega\}, \{\cdot^{\mathcal{I}^-_i} \mid i \in \omega\})$ such that $\Delta^\mathcal{I} = \{John, Mary, Tom\}$, $\mathit{Happy}^{\mathcal{I}^+_0} = \{John\}$, $\mathit{Happy}^{\mathcal{I}^-_0} = \{Mary, Tom\}$, $\mathit{Happy}^{\mathcal{I}^+_1} = \emptyset$, $\mathit{Happy}^{\mathcal{I}^-_1} = \{Tom\}$, $\mathit{Happy}^{\mathcal{I}^+_2} = \emptyset$, …, $\mathit{has\text{-}child}^{\mathcal{I}^+_0} = \{(John, Tom)\}$ with $\mathit{Happy}^{\mathcal{I}^+_0} \cap \mathit{Happy}^{\mathcal{I}^-_0} = \emptyset$, $\mathit{Happy}^{\mathcal{I}^+_{i+1}} \subseteq \mathit{Happy}^{\mathcal{I}^+_i}$, and $\mathit{Happy}^{\mathcal{I}^-_{i+1}} \subseteq \mathit{Happy}^{\mathcal{I}^-_i}$. Then, we obtain the interpretation functions $\cdot^{\mathcal{I}^+_i}$ and $\cdot^{\mathcal{I}^-_i}$ expanded to the $\mathcal{ALC}_\sim$-concepts $\exists \mathit{has\text{-}child}.{\sim}\mathit{Happy}$, $\neg{\sim}\neg{\sim}\mathit{Happy}$, and $\neg{\sim}{\sim}\neg\mathit{Happy}$ as below:

$$\begin{aligned}(\exists \mathit{has\text{-}child}.{\sim}\mathit{Happy})^{\mathcal{I}^+_0} &= \{d_1 \in \Delta^\mathcal{I} \mid \exists d_2[(d_1, d_2) \in \mathit{has\text{-}child}^{\mathcal{I}^+_0} \wedge d_2 \in \mathit{Happy}^{\mathcal{I}^-_0}]\} \\ &= \{John\}\end{aligned}$$

$$\begin{aligned}(\neg{\sim}\neg{\sim}\mathit{Happy})^{\mathcal{I}^+_0} &= \Delta^\mathcal{I} \backslash (\sim\neg{\sim}\mathit{Happy})^{\mathcal{I}^+_0} \\ &= \Delta^\mathcal{I} \backslash (\neg{\sim}\mathit{Happy})^{\mathcal{I}^-_0} \\ &= \Delta^\mathcal{I} \backslash (\sim\mathit{Happy})^{\mathcal{I}^+_1} \\ &= \Delta^\mathcal{I} \backslash \mathit{Happy}^{\mathcal{I}^-_1} \\ &= \{John, Mary\}\end{aligned} \qquad \begin{aligned}(\neg{\sim}{\sim}\neg\mathit{Happy})^{\mathcal{I}^+_0} &= \Delta^\mathcal{I} \backslash (\sim{\sim}\neg\mathit{Happy})^{\mathcal{I}^+_0} \\ &= \Delta^\mathcal{I} \backslash (\sim\neg\mathit{Happy})^{\mathcal{I}^-_0} \\ &= \Delta^\mathcal{I} \backslash (\neg\mathit{Happy})^{\mathcal{I}^+_0} \\ &= \{John\}\end{aligned}$$

Remark. Semantically, the three conditions $A^{\mathcal{I}^+_0} \cap A^{\mathcal{I}^-_0} = \emptyset$, $A^{\mathcal{I}^+_{i+1}} \subseteq A^{\mathcal{I}^+_i}$, and $A^{\mathcal{I}^-_{i+1}} \subseteq A^{\mathcal{I}^-_i}$ in the $\mathcal{ALC}^n_\sim$-interpretation $\mathcal{I}$ define the inconsistency of contraries between $\mathcal{ALC}_\sim$-concepts. Syntactically, the conditions lead to the inconsistent pairs $\langle A, {\sim}A\rangle$, $\langle \neg(\sim\neg)^i A, (\sim\neg)^{i+1} A\rangle$, and $\langle (\neg\sim)^{i+1} A, {\sim}(\neg\sim)^{i+1} A\rangle$ of $\mathcal{ALC}_\sim$-concepts. Each pair consists of a concept C and its strong negation $\sim C$ (i.e., $\langle C, \sim C\rangle$) where C is of the form A, $\neg(\sim\neg)^i A$, or $(\neg\sim)^{i+1} A$. For example, $\neg Red$ and $\sim\neg Red$ are inconsistent. In the next lemma, these conditions are generalized to any $\mathcal{ALC}_\sim$-concept.

Lemma 1. *Let $\mathcal{I} = (\Delta^{\mathcal{I}}, \{\cdot^{\mathcal{I}_i^+} \mid i \in \omega\}, \{\cdot^{\mathcal{I}_i^-} \mid i \in \omega\})$ be an $\mathcal{ALC}_{\sim}^n$-interpretation. For any $\mathcal{ALC}_{\sim}$-concept C, the following statements hold:*

1. $C^{\mathcal{I}} \cap {\sim}C^{\mathcal{I}} = \emptyset$,
2. $({\sim}\neg)^{i+1}C^{\mathcal{I}} \subseteq ({\sim}\neg)^{i}C^{\mathcal{I}}$,
3. ${\sim}(\neg{\sim})^{i+1}C^{\mathcal{I}} \subseteq {\sim}(\neg{\sim})^{i}C^{\mathcal{I}}$.

This lemma will be used to prove the correspondence between a tableau for an $\mathcal{ALC}_{\sim}$-concept and the satisfiability of the concept.

Any $\mathcal{ALC}_{\sim}$-concept is transformed into an equivalent one in a normal negation form (that is more complicated than the normal negation form in $\mathcal{ALC}$) using the following equivalences from left to right:

$$
\begin{aligned}
(\neg)^k(\sim\neg)^i{\sim}{\sim}C &\equiv (\neg)^k(\sim\neg)^iC \\
(\sim)^k(\neg\sim)^i\neg\neg C &\equiv (\sim)^k(\neg\sim)^iC \\
(\neg)^k(\sim\neg)^i{\sim}(C \sqcap D) &\equiv (\neg)^k(\sim\neg)^i({\sim}C \sqcup {\sim}D) \\
(\neg)^k(\sim\neg)^i{\sim}(C \sqcup D) &\equiv (\neg)^k(\sim\neg)^i({\sim}C \sqcap {\sim}D) \\
(\neg)^k(\sim\neg)^i{\sim}(\forall R.C) &\equiv (\neg)^k(\sim\neg)^i(\exists R{\sim}C) \\
(\neg)^k(\sim\neg)^i{\sim}(\exists R.C) &\equiv (\neg)^k(\sim\neg)^i(\forall R{\sim}C) \\
(\sim)^k(\neg\sim)^i\neg(C \sqcap D) &\equiv (\sim)^k(\neg\sim)^i(\neg C \sqcup \neg D) \\
(\sim)^k(\neg\sim)^i\neg(C \sqcup D) &\equiv (\sim)^k(\neg\sim)^i(\neg C \sqcap \neg D) \\
(\sim)^k(\neg\sim)^i\neg(\forall R.C) &\equiv (\sim)^k(\neg\sim)^i(\exists R.\neg C) \\
(\sim)^k(\neg\sim)^i\neg(\exists R.C) &\equiv (\sim)^k(\neg\sim)^i(\forall R.\neg C)
\end{aligned}
$$

where $k \in \{0, 1\}$ and $i \in \omega$. The form of the concepts obtained by this transformation is called a *constructive normal negation form,* where the four types of negation forms $({\sim}\neg)^{i+1}$, $\neg({\sim}\neg)^i$, $(\neg{\sim})^{i+1}$, and ${\sim}(\neg{\sim})^i$ occur only in front of a concept name. For example, $({\sim}\neg A_1 \sqcup \neg{\sim}\neg A_2) \sqcap {\sim}(\neg{\sim})^4 A_3$ is in the constructive normal negation form.

Proposition 2. *Every concept equation $C \equiv D$ in the translation is $\mathcal{ALC}_{\sim}^n$-valid.*

Next, we will discuss an important property of $\mathcal{ALC}_{\sim}^n$-interpretations that is derived from the contrary condition.

Lemma 2. *Let $\mathcal{I} = (\Delta^{\mathcal{I}}, \{\cdot^{\mathcal{I}_i^+} \mid i \in \omega\}, \{\cdot^{\mathcal{I}_i^-} \mid i \in \omega\})$ be an $\mathcal{ALC}_{\sim}^n$-interpretation that satisfies the contrary condition. For any $\mathcal{ALC}_{\sim}$-concept C, the following statements hold:*

1. $C^{\mathcal{I}} \cup {\sim}C^{\mathcal{I}} \neq \Delta^{\mathcal{I}}$,
2. $({\sim}\neg)^{i+1}C^{\mathcal{I}} \subsetneq ({\sim}\neg)^{i}C^{\mathcal{I}}$,
3. ${\sim}(\neg{\sim})^{i+1}C^{\mathcal{I}} \subsetneq {\sim}(\neg{\sim})^{i}C^{\mathcal{I}}$.

This lemma guarantees that the proposed semantics characterizes the differences between contradictories and contraries in every interpretation. The following theorem states the property of contradictoriness and contrariness for $\mathcal{ALC}_{\sim}^n$-interpretations. Contradictoriness and contrariness are preserved in an $\mathcal{ALC}_{\sim}^n$-interpretation $\mathcal{I}$ of $\mathcal{ALC}_{\sim}$ if ${\sim}C^{\mathcal{I}} \subsetneq \neg C^{\mathcal{I}}$.

Theorem 2 (Contradictoriness and contrariness for $\mathcal{ALC}^n_\sim$).
If an $\mathcal{ALC}^n_\sim$-interpretation satisfies the contrary condition, then contradictoriness and contrariness are preserved in the $\mathcal{ALC}^n_\sim$-interpretation.

We would like to apply the replacement property [17] to conceptual representation and strong negation in $\mathcal{ALC}_\sim$. Knowledge base designers rewrite concepts by their equivalent concepts in the context of (conceptual) knowledge representation (e.g., re-building ontologies in the Semantic Web). However, the replacement property provides a limitation such that concepts can only be replaced by strongly equivalent concepts when various combinations of the two types of negation are used. Let $C,\ D$ be $\mathcal{ALC}_\sim$-concepts. C and D are equivalent if, for every $\mathcal{ALC}^n_\sim$-interpretation $\mathcal{I}$, $C^{\mathcal{I}} = D^{\mathcal{I}}$. C and D are strongly equivalent if, for every $\mathcal{ALC}^n_\sim$-interpretation $\mathcal{I}$, $C^{\mathcal{I}_i^+} = D^{\mathcal{I}_i^+}$ and $C^{\mathcal{I}_i^-} = D^{\mathcal{I}_i^-}$. The concept E_C denotes a (complex) concept that contains the concept C as a subconcept of E_C and E_D denotes the concept obtained by replacing C in E_C by D.

Theorem 3 (Replacement for $\mathcal{ALC}_\sim$). *Let $C,\ D$ be $\mathcal{ALC}_\sim$-concepts. If C and D are strongly equivalent, then E_C and E_D are also equivalent.*

It should be noted that the replacement property under strong equivalence is natural in the presence of strong negation.

3.2 The constructive description logic with Heyting negation and strong negation: $\mathcal{CALC}_\sim$

We define the description logic $\mathcal{CALC}_\sim$ (as an extension of the constructive description logic $\mathcal{CALC}^{N4}$ [10]) by combining Heyting negation and strong negation. The concepts in the language (called $\mathcal{CALC}_\sim$-concepts) are constructed by concept names A; role names R; the connectives $\sqcap$, $\sqcup$, $-$ (Heyting negation), and $\sim$ (strong negation); and the quantifiers $\forall$, $\exists$. Every concept name $A \in \mathbf{C}$ is a $\mathcal{CALC}_\sim$-concept. If R is a role name and C, D are $\mathcal{CALC}_\sim$-concepts, then $-C$, $\sim C$, $C \sqcap D$, $C \sqcup D$, $\forall R.C$, and $\exists R.C$ are $\mathcal{CALC}_\sim$-concepts. We give an interpretation of $\mathcal{CALC}_\sim$-concepts (called a $\mathcal{CALC}^2_\sim$-interpretation) as follows:

Definition 3. *A $\mathcal{CALC}^2_\sim$-interpretation $\mathcal{I}$ is a tuple $(W, \preceq, \Delta^{\mathcal{I}}, \{\cdot^{\mathcal{I}_t^+} \mid t \in W\}, \{\cdot^{\mathcal{I}_t^-} \mid t \in W\})$, where W is a set of worlds, $\Delta^{\mathcal{I}} = \{\Delta^{\mathcal{I}_t} \mid t \in W\}$ is the family of non-empty sets and $\cdot^{\mathcal{I}_t^+}$ and $\cdot^{\mathcal{I}_t^-}$ are interpretation functions for each world $t \in W$ ($A^{\mathcal{I}_t^+} \subseteq \Delta^{\mathcal{I}}$, $A^{\mathcal{I}_t^-} \subseteq \Delta^{\mathcal{I}}$, $R^{\mathcal{I}_t^+} \subseteq \Delta^{\mathcal{I}_t} \times \Delta^{\mathcal{I}_t}$, and $a^{\mathcal{I}_t^+} \in \Delta^{\mathcal{I}_t}$) such that:*

1. $\bot^{\mathcal{I}_t^+} = \emptyset$ *and* $\top^{\mathcal{I}_t^+} = \Delta^{\mathcal{I}_t}$,
2. $A^{\mathcal{I}_t^+} \cap A^{\mathcal{I}_t^-} = \emptyset$,
3. *if* $t, t' \in W$ *and* $t \preceq t'$, *then* $\Delta^{\mathcal{I}_t} \subseteq \Delta^{\mathcal{I}_{t'}}$, $A^{\mathcal{I}_t^+} \subseteq A^{\mathcal{I}_{t'}^+}$, $A^{\mathcal{I}_t^-} \subseteq A^{\mathcal{I}_{t'}^-}$, *and* $R^{\mathcal{I}_t^+} \subseteq R^{\mathcal{I}_{t'}^+}$.

For every world $t \in W$, the interpretation functions $\cdot^{\mathcal{I}_t^+}$ and $\cdot^{\mathcal{I}_t^-}$ are expanded to $\mathcal{CALC}_\sim$-concepts as follows:

$$
\begin{aligned}
(-C)^{\mathcal{I}_t^+} &= \{d \,|\, d \in \Delta^{\mathcal{I}_{t'}} \backslash C^{\mathcal{I}_{t'}^+} \text{ s.t. } t \preceq t'\} & (\sim C)^{\mathcal{I}_t^+} &= C^{\mathcal{I}_t^-} \\
(C \sqcap D)^{\mathcal{I}_t^+} &= C^{\mathcal{I}_t^+} \cap D^{\mathcal{I}_t^+} & (C \sqcup D)^{\mathcal{I}_t^+} &= C^{\mathcal{I}_t^+} \cup D^{\mathcal{I}_t^+} \\
(\forall R.C)^{\mathcal{I}_t^+} &= \{d_1 \in \Delta^{\mathcal{I}_t} \,|\, \forall t'[t \preceq t' \rightarrow \forall d_2 \in \Delta^{\mathcal{I}_{t'}}[(d_1, d_2) \in R^{\mathcal{I}_{t'}^+} \rightarrow d_2 \in C^{\mathcal{I}_{t'}^+}]]\} \\
(\exists R.C)^{\mathcal{I}_t^+} &= \{d_1 \in \Delta^{\mathcal{I}_t} \,|\, \exists d_2 \in \Delta^{\mathcal{I}_{t'}}[(d_1, d_2) \in R^{\mathcal{I}_t^+} \wedge d_2 \in C^{\mathcal{I}_t^+}]\} \\
(-C)^{\mathcal{I}_t^-} &= C^{\mathcal{I}_t^+} & (\sim C)^{\mathcal{I}_t^-} &= C^{\mathcal{I}_t^+} \\
(C \sqcap D)^{\mathcal{I}_t^-} &= C^{\mathcal{I}_t^-} \cup D^{\mathcal{I}_t^-} & (C \sqcup D)^{\mathcal{I}_t^-} &= C^{\mathcal{I}_t^-} \cap D^{\mathcal{I}_t^-} \\
(\forall R.C)^{\mathcal{I}_t^-} &= \{d_1 \in \Delta^{\mathcal{I}_t} \,|\, \exists d_2 \in \Delta^{\mathcal{I}_{t'}}[(d_1, d_2) \in R^{\mathcal{I}_t^+} \wedge d_2 \in C^{\mathcal{I}_t^-}]\} \\
(\exists R.C)^{\mathcal{I}_t^-} &= \{d_1 \in \Delta^{\mathcal{I}_t} \,|\, \forall t'[t \preceq t' \rightarrow \forall d_2 \in \Delta^{\mathcal{I}_{t'}}[(d_1, d_2) \in R^{\mathcal{I}_{t'}^+} \rightarrow d_2 \in C^{\mathcal{I}_{t'}^-}]]\}
\end{aligned}
$$

An $\mathcal{ALC}_\sim^n$-interpretation $\mathcal{I}$ satisfies the contrary condition if $A^{\mathcal{I}^+} \cup A^{\mathcal{I}^-} \neq \Delta^{\mathcal{I}}$, where $A^{\mathcal{I}^+} = \bigcup_{t \in W} A^{\mathcal{I}_t^+}$ and $A^{\mathcal{I}^-} = \bigcup_{t \in W} A^{\mathcal{I}_t^-}$. The $\mathcal{CALC}_\sim^2$-interpretation $C^{\mathcal{I}}$ of each $\mathcal{CALC}_\sim$-concept is given by $\bigcup_{t \in W} C^{\mathcal{I}_t^+}$. A $\mathcal{CALC}_\sim$-concept C (or a concept equation $C \equiv D$) is $\mathcal{CALC}_\sim^2$-satisfiable if there exists an $\mathcal{CALC}_\sim^2$-interpretation $\mathcal{I}$, called a $\mathcal{CALC}_\sim^2$-model of C (or $C \equiv D$), such that $C^{\mathcal{I}} \neq \emptyset$ (or $C^{\mathcal{I}} = D^{\mathcal{I}}$); otherwise, it is $\mathcal{CALC}_\sim^2$-unsatisfiable. In particular, if a $\mathcal{CALC}_\sim$-concept C is $\mathcal{CALC}_\sim^2$-satisfiable and the $\mathcal{CALC}_\sim^2$-model satisfies the contrary condition, then it is $\mathcal{CALC}_\sim^2$-satisfiable under the contrary condition. Otherwise, it is $\mathcal{CALC}_\sim^2$-unsatisfiable under the contrary condition. A concept equation $C \equiv D$ is $\mathcal{CALC}_\sim^2$-valid if every $\mathcal{CALC}_\sim^2$-interpretation $\mathcal{I}$ is a $\mathcal{CALC}_\sim^2$-model of $C \equiv D$. Contradictoriness and contrariness are preserved in an $\mathcal{CALC}_\sim^2$-interpretation of $\mathcal{CALC}_\sim$ if $\sim C^{\mathcal{I}} \subsetneq -C^{\mathcal{I}}$.

Theorem 4 (Contradictoriness and contrariness for $\mathcal{CALC}_\sim^2$).
Contradictoriness and contrariness are not preserved in some $\mathcal{CALC}_\sim^2$-interpretations that satisfy the contrary condition.

Table 1 shows the contradictoriness and contrariness for $\mathcal{ALC}_\sim$ and $\mathcal{CALC}_\sim$. The $\mathcal{CALC}_\sim$-concepts can be used to represent predicate denial and predicate term negation that capture conceptual models or describe a certain domain of interest; however, the contradictoriness and contrariness are not preserved in some $\mathcal{CALC}_\sim^2$-interpretations. For the $\mathcal{ALC}_\sim$-concepts, the contradictoriness and contrariness are preserved in every $\mathcal{ALC}_\sim^n$-interpretation since strong negation is suitably added to the classical description logic $\mathcal{ALC}$ without the undesirable equivalent $C \equiv \sim\neg C$ in the semantics. In the next section, the tableau-based satisfiability algorithm for $\mathcal{ALC}$ is extended to $\mathcal{ALC}_\sim$. This extension is based on the contradictoriness and contrariness for the $\mathcal{ALC}_\sim^n$-interpretations.

Additionally, we show that the replacement property holds for strongly equivalent $\mathcal{CALC}_\sim$-concepts. Let $C,\ D$ be $\mathcal{CALC}_\sim$-concepts. C and D are equivalent if, for every $\mathcal{CALC}_\sim^2$-interpretation $\mathcal{I}$, $C^{\mathcal{I}} = D^{\mathcal{I}}$. Let $C^{\mathcal{I}^+}$ denote $\bigcup_{t \in W} C^{\mathcal{I}_t^+}$ and $C^{\mathcal{I}^-}$ denote $\bigcup_{t \in W} C^{\mathcal{I}_t^-}$. C and D are strongly equivalent if, for every $\mathcal{CALC}_\sim^2$-interpretation $\mathcal{I}$, $C^{\mathcal{I}^+} = D^{\mathcal{I}^+}$ and $C^{\mathcal{I}^-} = D^{\mathcal{I}^-}$.

Table 1. Contradictoriness and contrariness for $\mathcal{ALC}_{\sim}$ and $\mathcal{CALC}_{\sim}$

DL Syntax	Semantics	Contradictoriness and contrariness
$\mathcal{ALC}_{\sim}$	$\mathcal{ALC}^{2}_{\sim}$	not preserved for every interpretation
	$\mathcal{ALC}^{n}_{\sim}$	preserved for every interpretation
$\mathcal{CALC}_{\sim}$	$\mathcal{CALC}^{2}_{\sim}$	not preserved for some interpretations

Theorem 5 (Replacement for $\mathcal{CALC}_{\sim}$). *Let $C,\ D$ be $\mathcal{CALC}_{\sim}$-concepts. If C and D are strongly equivalent, then E_C and E_D are (strongly) equivalent.*

4 Tableau-based algorithm for $\mathcal{ALC}_{\sim}$

We denote $rol(C)$ as the set of roles occurring in an $\mathcal{ALC}_{\sim}$-concept C. For instance, $rol(\neg\forall R_1.\exists R_2.C_1 \sqcup {\sim}C_2) = \{R_1, R_2\}$. To prove the soundness and completeness of the tableau-based satisfiability algorithm, a tableau for an $\mathcal{ALC}_{\sim}$-concept is created by adding conditions for the forms ${\sim}C$ and $({\sim}\neg)^i C$ to a tableau for an $\mathcal{ALC}$-concept (as in [6]).

Definition 4. *Let D be an $\mathcal{ALC}_{\sim}$-concept in the constructive normal negation form. A tableau T for D is a tuple (S, L, E), where S is a set of individuals, $L\colon S \to 2^{sub(D)}$ is a mapping from each individual into a set of concepts in $sub(D)$, and $E\colon rol(D) \to 2^{S\times S}$ is a mapping from each role into a set of pairs of individuals. There exists some $s_0 \in S$ such that $D \in L(s_0)$, and for all $s, t \in S$, the following conditions hold:*

1. *if $C \in L(s)$, then ${\sim}C, \neg C \notin L(s)$,*
2. *if $C_1 \sqcap C_2 \in L(s)$, then $C_1 \in L(s)$ and $C_2 \in L(s)$,*
3. *if $C_1 \sqcup C_2 \in L(s)$, then $C_1 \in L(s)$ or $C_2 \in L(s)$,*
4. *if $\forall R.C \in L(s)$ and $(s,t) \in E(R)$, then $C \in L(t)$,*
5. *if $\exists R.C \in L(s)$, then there exists $t \in S$ such that $(s,t) \in E(R)$ and $C \in L(t)$,*
6. *for every $i \in \omega$, if $({\sim}\neg)^{i+1}C \in L(s)$, then $({\sim}\neg)^{i}C \in L(s)$.*

In particular, it is called a C-tableau if the the following conditions hold:

7. *for every $i \in \omega$, there exists $s \in S$ such that $({\sim}\neg)^{i}C \in L(s)$ and $({\sim}\neg)^{i+1}C \notin L(s)$,*
8. *there exists $s \in S$ such that $C \notin L(s)$ and ${\sim}C \notin L(s)$.*

Conditions 1 and 6 reflect the $\mathcal{ALC}^{n}_{\sim}$-interpretation of $\mathcal{ALC}_{\sim}$-concepts combining classical and strong negations. Condition 1 states that $C \in L(s)$ implies ${\sim}C \notin L(s)$ (in addition to $\neg C \notin L(s)$) to satisfy the semantic condition $A^{\mathcal{I}^+_0} \cap A^{\mathcal{I}^-_0} = \emptyset$. Moreover, Condition 6 is imposed for the semantic conditions $A^{\mathcal{I}^+_{i+1}} \subseteq A^{\mathcal{I}^+_i}$ and $A^{\mathcal{I}^-_{i+1}} \subseteq A^{\mathcal{I}^-_i}$. For example, by Condition 6, if ${\sim}\neg{\sim}\mathit{Happy} \in L(s)$, then ${\sim}\mathit{Happy} \in L(s)$. In the corresponding semantics, if $d \in ({\sim}\neg{\sim}\mathit{Happy})^{\mathcal{I}^+_0}$, then $d \in ({\sim}\mathit{Happy})^{\mathcal{I}^+_1}$. Hence, by the condition $A^{\mathcal{I}^+_{i+1}} \subseteq A^{\mathcal{I}^+_i}$, we obtain $d \in ({\sim}\mathit{Happy})^{\mathcal{I}^+_0}$. Conditions 7 and 8 correspond to the contrary condition for the $\mathcal{ALC}^{n}_{\sim}$-interpretation, i.e., $A^{\mathcal{I}^+_0} \cup A^{\mathcal{I}^-_0} \neq \Delta^{\mathcal{I}}$, $A^{\mathcal{I}^+_{i+1}} \subsetneq A^{\mathcal{I}^+_i}$, and $A^{\mathcal{I}^-_{i+1}} \subsetneq A^{\mathcal{I}^-_i}$. The next lemma shows the correspondence between the existence of a tableau for an $\mathcal{ALC}_{\sim}$-concept and its satisfiability.

$\sqcap$**-rule:** $L(x) = L(x) \cup \{C_1, C_2\}$
if $C_1 \sqcap C_2 \in L(x)$ and $\{C_1, C_2\} \not\subseteq L(x)$

$\sqcup$**-rule:** $L(x) = L(x) \cup \{C_1\}$ or $L(x) = L(x) \cup \{C_2\}$
if $C_1 \sqcup C_2 \in L(x)$ and $\{C_1, C_2\} \cap L(x) = \emptyset$

$\forall$**-rule:** $L(y) = L(y) \cup \{C\}$
if $\forall R.C \in L(x)$, $(x, y) \in E(R)$ and $C \notin L(y)$

$\exists$**-rule:** $S = S \cup \{y\}$ with $y \notin S$, $E(R) = E(R) \cup \{(x, y)\}$ and $L(y) = \{C\}$
if $\exists R.C \in L(x)$ and $\{z \mid (x, z) \in E(R), C \in L(z)\} = \emptyset$

$(\sim\neg)^i$**-rule 1:** $L(x) = L(x) \cup \{(\sim\neg)^i C\}$
if $(\sim\neg)^{i+1} C \in L(x)$ and $(\sim\neg)^i C \notin L(x)$

$(\sim\neg)^i$**-rule 2:** $S = S \cup \{y\}$ with $y \notin S$ and $L(y) = \{(\sim\neg)^i C\}$
if $(\sim\neg)^i C \in L(x)$ and
there exists no $z \in S$ such that $(\sim\neg)^{i+1} C \notin L(z)$ and $(\sim\neg)^i C \in L(z)$

$\sim$**-rule:** $S = S \cup \{y\}$ with $y \notin S$ and $L(y) = \emptyset$
if $A \in L(x)$ or $\sim A \in L(x)$ and
there exists no $z \in S$ such that $\{A, \sim A\} \cap L(z) = \emptyset$.

Fig. 3. Completion rules for $\mathcal{ALC}_\sim$-concepts

Lemma 3. *Let D be an $\mathcal{ALC}_\sim$-concept. There exists a tableau for D if and only if it is $\mathcal{ALC}^n_\sim$-satisfiable. In particular, there exists a C-tableau for D if and only if it is $\mathcal{ALC}^n_\sim$-satisfiable under the contrary condition.*

Note that while every $\mathcal{ALC}^n_\sim$-interpretation $\mathcal{I}$ consists of infinite interpretation functions $\cdot^{\mathcal{I}^+_i}$ and $\cdot^{\mathcal{I}^-_i}$ for $i \in \omega$, the tableau corresponding to an $\mathcal{ALC}^n_\sim$-model of an $\mathcal{ALC}_\sim$-concept is finitely defined. Each $\mathcal{ALC}_\sim$-concept can be satisfied by finite interpretation functions because the number of connectives occurring in it is finite. For example, $(\sim\neg)^m A$ can be satisfied by the maximum $2m+1$ of interpretation functions $\cdot^{\mathcal{I}^+_0}, \ldots, \cdot^{\mathcal{I}^+_m}, \cdot^{\mathcal{I}^-_0}, \ldots, \cdot^{\mathcal{I}^-_{m-1}}$. Lemma 3 indicates that given a tableau for an $\mathcal{ALC}_\sim$-concept D, we can define an $\mathcal{ALC}^n_\sim$-interpretation $\mathcal{I}$ satisfying it (i.e., its $\mathcal{ALC}^n_\sim$-model). The model is constructed in such a manner that for every constructive double negation $(\sim\neg)^i A$ (resp. $\sim(\neg\sim)^i A$) in $sub(D)$, $A^{\mathcal{I}^+_i}$ (resp. $A^{\mathcal{I}^-_i}$) is defined by the set of individuals $\{s \mid (\sim\neg)^i A \in L(s)\}$ (resp. $\{s \mid \sim(\neg\sim)^i A \in L(s)\}$). Thus, the finiteness of a tableau for an $\mathcal{ALC}_\sim$-concept leads to the termination of its satisfiability algorithm which we will present.

In order to determine the satisfiability of $\mathcal{ALC}_\sim$-concepts, the tableau-based algorithm for $\mathcal{ALC}$ will be extended by introducing three new completion rules ($(\sim\neg)^i$-rule 1, $(\sim\neg)^i$-rule 2, and $\sim$-rule) and clash forms with respect to strong negation and constructive double negation. In Fig.3, the completion rules for $\mathcal{ALC}_\sim$-concepts are presented (as in [4, 13]). $(\sim\neg)^i$-rule 1 is applied to $\mathcal{ALC}_\sim$-concepts of the forms $(\sim\neg)^i A$ and $\sim(\neg\sim)^i A$. $(\sim\neg)^i$-rule 2 and $\sim$-rule introduce new variables if there exists no $z \in S$ such that $(\sim\neg)^{i+1} C \notin L(z)$ and $(\sim\neg)^i C \in L(z)$; or $\{A, \sim A\} \cap L(z) = \emptyset$.

Remark. The algorithm has to recognize additional clash forms besides $\{A, \neg A\}$ and $\{\bot\}$. That is, $L(x)$ contains a clash if it contains $\{C_1, \neg C_1\}$, $\{C_2, \sim C_2\}$, or $\{\bot\}$, where

C_1 is of the form $(\sim\neg)^i A$ or $\sim(\neg\sim)^i A$ and C_2 is of the form $(\neg\sim)^i A$ or $\neg(\sim\neg)^i A$. For example, if $\{\neg\sim\neg A_1, \sim\neg\sim\neg A_1\} \subseteq L(x_1)$, then it contains a clash.

We present a tableau-based satisfiability algorithm for $\mathcal{ALC}_\sim$. Given an $\mathcal{ALC}_\sim$-concept D, the following procedure constructs a forest $ST = (S, E_{rol(D)} \cup E_\sim \cup E_{(\sim\neg)^i}, x_0)$ for D, where S is a set of individuals, each node $x \in S$ is labeled as $L(x)$, $E_{rol(D)} = \{(x,y) \in E(R) \mid R \in rol(D)\}$ (each edge $(x,y) \in E(R)$ is labeled as R), $(x,y) \in E_\sim \Leftrightarrow y$ is introduced for $A \in L(x)$ or $\sim A \in L(x)$ in $\sim$-rule, $(x,y) \in E_{(\sim\neg)^i} \Leftrightarrow y$ is introduced for $(\sim\neg)^i C \in L(x)$ in $(\sim\neg)^i$-rule 2, and x_0 is the root. First, set the initial forest $ST = (\{x_0\}, \emptyset, x_0)$, where $S = \{x_0\}$, $L(x_0) = \{D\}$, $E(R) = \emptyset$ for all $R \in rol(D)$, and $E_\sim = E_{(\sim\neg)^i} = \emptyset$. Then, apply completion rules in Fig.3 to ST until none of the rules are applicable. A forest ST is called complete if any completion rule is not applicable to it. If there is a clash-free complete forest ST, then return "satisfiable," and otherwise return "unsatisfiable."

We show the correctness of the tableau-based satisfiability algorithm under the contrary condition (soundness, completeness and termination)[3] and the complexity of the satisfiability problem. We sketch the behavior of the new completion rules in the algorithm. Unlike the other completion rules, each application of the new completion rules does not subdivide a concept. However, since $(\sim\neg)^i$-rules 1 and 2 add only a subconcept to each node, $\sim$-rule creates an empty node, and the number of variables introduced in $(\sim\neg)^i$-rule 2 and $\sim$-rule is bounded by polynomial, the termination can be established.

Theorem 6 (Satisfiability under the contrary condition).
Let D be an $\mathcal{ALC}_\sim$-concept. The following statements hold:

1. *The tableau-based algorithm terminates.*
2. *The tableau-based algorithm constructs a clash-free complete forest for an $\mathcal{ALC}_\sim$-concept D if and only if D is $\mathcal{ALC}^n_\sim$-satisfiable under the contrary condition.*
3. *Satisfiability of $\mathcal{ALC}_\sim$-concepts is PSPACE-complete.*

5 Conclusion

We have presented an extended description logic $\mathcal{ALC}_\sim$ that incorporates classical negation and strong negation for representing contraries, contradictories, and subcontraries between concepts. The important specification of the description logic is that strong negation is added to the classical description logic $\mathcal{ALC}$ and the property of contradictoriness and contrariness holds for every interpretation. The two negations are adequately characterized by $\mathcal{ALC}^n_\sim$-interpretations, unlike $\mathcal{ALC}^2_\sim$- and $\mathcal{CALC}^2_\sim$-interpretations. Technically, the semantics of strong negation is adapted to the oppositions in the philosophical study of negation [7]. Furthermore, we have developed a satisfiability algorithm for $\mathcal{ALC}_\sim$ that is extended to add three new completion rules to the tableau-based algorithm for $\mathcal{ALC}$ [13]. It constructs an $\mathcal{ALC}^n_\sim$-model satisfying the contrary condition, in which a constructive normal negation form and various

[3] The tableau-based satisfi ability algorithm is complete in the class of $\mathcal{ALC}^n_\sim$-interpretations where $(\sim\neg)^i$-rule 2 and $\sim$-rule are used to construct a model satisfying the contrary condition.

clash forms are defined to treat complex negative concepts. The description logic provides a decidable fragment (precisely, PSPACE-complete) of classical first-order logic with classical negation and strong negation (but not constructive description logic with Heyting negation and strong negation).

References

1. S. Akama. Constructive predicate logic with strong negation and model theory. *Notre Dame Journal of Formal Logic*, 29(1):18–27, 1988.
2. J. Y. Halpern and Y. Moses. A guide to completeness and complexity for model logics of knowledge and belief. *Artificial Intelligence*, 54(3):319–379, April 1992.
3. H. Herre, J. Jaspars, and G. Wagner. Partial logics with two kinds of negation as a foundation for knowledge-based reasoning. In D.M. Gabbay and H. Wansing, editors, *What is Negation ?*, pages 121–159. Kluwer Academic Publishers, 1999.
4. B. Hollunder, W. Nutt, and M. Schmidt-Schauß. Subsumption algorithms for concept description languages. In *Proceedings of ECAI-90, 9th European Conference on Artificial Intelligence*, pages 348–353, 1990.
5. L. R. Horn. *A Natural History of Negation*. University of Chicago Press, 1989.
6. I. Horrocks and U. Sattler. Ontology reasoning in the SHOQ(D) description logic. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, 2001.
7. K. Kaneiwa. On the semantics of classical first-order logic with constructive double negation. NII Technical Report NII-2004-010E, National Institute of Informatics.
8. K. Kaneiwa and S. Tojo. An order-sorted resolution with implicitly negative sorts. In *Proceedings of the 2001 Int. Conf. on Logic Programming*, pages 300–314. Springer-Verlag, 2001. LNCS 2237.
9. D. Nelson. Constructible falsity. *The Journal of Symbolic Logic*, 14(1):16–26, 1949.
10. S. P. Odintsove and H. Wansing. Inconsistency-tolerant description logic. motivation and basic systems. In V. Hendricks and J. Malinowski, editors, *Trends in Logic. 50 Years of Studia Logica*, pages 301–335. Kluwer Academic Publishers, 2003.
11. D. Pearce and G. Wagner. Logic programming with strong negation. In P. Schroeder-Heister, editor, *Proceedings of the International Workshop on Extensions of Logic Programming*, volume 475 of *Lecture Notes in Artificial Intelligence*, pages 311–326, Tübingen, FRG, December, 8–10 1989. Springer-Verlag.
12. M. La Palme Reyes, J. Macnamara, G. E. Reyes, and H. Zolfaghari. Models for non-boolean negations in natural languages based on aspect. In D.M. Gabbay and H. Wansing, editors, *What is Negation ?*, pages 241–260. Kluwer Academic Publishers, 1999.
13. M. Schmidt-Schauss and G. Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48:1–26, 1991.
14. R. H. Thomason. A semantical study of constructible falsity. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 15:247–257, 1969.
15. G. Wagner. A database needs two kinds of negation. In B. Thalheim, J. Demetrovics, and H-D. Gerhardt, editors, *Mathematical Foundations of Database Systems*, pages 357–371. LNCS 495, Springer–Verlag, 1991.
16. G. Wagner. *Vivid Logic: Knowledge-Based Reasoning with Two Kinds of Negation*. Springer-Verlag, 1994.
17. H. Wansing. *The logic of information structures*, volume 681 of *LNAI*. Springer-Verlag, 1993.
18. H. Wansing. Negation. In L. Goble, editor, *The Blackwell Guide to Philosophical Logic*, pages 415–436. Basil Blackwell Publishers, 2001.

Hanging Monadic Relations in Conceptual Graphs

Pavel Kocura

Department of Computer Science
Loughborough University
Loughborough, Lecestershire LE11 3TU
Englend
P.Kocura@lboro.ac.uk

Abstract. Constructing the inheritance hierarchy for a non-trivial ontology is an iterative, and essentially non-terminating process. Multiple inheritance may lead to problems, which can be alleviated by dismantling compound concept types into hanging monadic relations. This provides more flexibility. We show that the conventional projection operation on monadic relations yields less information than in graphs with compound types. We suggest an augmented, configurable projection operation, which also takes into account temporal information. The principle extends to n-adic relations. The new projection leads to changes in the operation of Join. Monadic relations are useful for compiling knowledge. We discuss canonical models in this context, and suggest a methodology for the generation of canonical models from predicate definitions.

1 Introduction

An ontology has to rest on the firm foundations of an inheritance hierarchy. Once the ontology domain is well understood, its generalization structure will remain static, hardly ever changing, or so we thought. A major counterexample is provided by the recent developments in the oldest and largest system of categorization - systematic biology. Increasingly, the 250-years-old Linnaean taxonomy structure cannot accommodate our current understanding of evolution. Systematic biologists had to augment the Linnaean nomenclature, and refine it with new ranks, e.g. phalanxes, infracohorts, supertribes, and innumerable other new categories. Occasionally, a small piece of evolutionary evidence would force them to rename hundreds of species and restructure their interconnections. As a result, a growing number of scientists support the development of a completely new system, the *PhyloCode* e.g. [1]. While the Linnaean taxonomy is based on shared characteristics, the PhyloCode models the branches of evolution. The two systems will run in parallel for some time. We could also use a more formal biological hierarchy based on species DNA. Even with these new coding systems, the process of constructing biological ontologies (and those of any other non-trivial domains) is unlikely to terminate soon, as new evidence will emerge and a better understanding will continue developing for a long time.

A conventional taxonomy is defined strictly as a tree: a single inheritance scheme categorizing a single aspect of the domain. In biological classification there are now at least three separate categorization systems, each generalizing over a different view of the same domain. Current computer-oriented knowledge representation systems, such as CGs, favour multiple inheritance. Formal Concept Analysis develops the ultimate generalization structure, which collects all known views of the domain, and generates an exhaustive set of all their possible combinations.

Taxonomies are 'accelerating' structures, both cognitively and computationally. An explicit inheritance hierarchy replaces slow, piecemeal inference that re-creates the implicit lines of generalization with fast, hard-wired lookup or, in encoded systems, simple comparison of codes. At first sight, multiple inheritance appears particularly attractive, but there might be a catch or two. For example, it leads to a combinatorial explosion of predicates. We could argue that, in a large and complex domain, multiple inheritance confounds cognitive organization by mixing orthogonal views of the domain. Also, it is notoriously difficult to implement, as there are few good coding systems, if any, for dense lattices. Thus, multiple inheritance may negatively affect both cognitive transparency and computational performance. Also, we may need to avoid top-down methods of building large, complex inheritance structures. As in systematic biology, a large part of the system might have to be radically restructured after a few local changes. We will go on to suggest ways of minimizing the undesirable effects of multiple inheritance in CGs, and explore the consequences for CGs representation and operations, in particular for projection.

2 Hanging Monadic Relations

In the original specification CGs [2], and the conventional interpretation of the semantics of types and relations, any domain property corresponding to a monadic predicate would be represented as a concept type. Multiple predicates applying to a single individual, e.g. [FEMALE:Max], [RICH:Max], [FAMOUS:Max], [BEAUTIFUL:Max], would be commonly joined into *compound types*, in this case giving [RICH_FAMOUS_BEAUTIFUL_FEMALE:Max], which requires multiple inheritance. This approach may result in staggeringly complicated types, such as RICH_FAMOUS_HANDSOME_BALD_OBESE_BESPECTACLED_BEARDED_BOHEMIAN- BORN_COMPUTER-SCIENCE-LECTURER ..., etc. In the typical CGs approach (and in FCA), this type would be represented by a single node in the lattice. The problems with compound types - cognitive, logical and of their implementation - stem from the large numbers of predicates, and from their internal complexity. Although they combine distinct properties, the labels themselves are atomic and un-analyzable. Some compound types do have dedicated labels in the natural language, and thus are part of our cognitive repertoire, but many do not have a linguistic or cognitive mapping (e.g. abstract, 'hidden' types). As for the combinatorially defined predicates (see the multi-faceted academic above), the user will think about them

only in terms of their real atomic components, never as an undivided whole. In fact, the system interface will have to reconstruct the compound type label from its components, entered individually and by the user. The interface will then search for the internal code of the node, replicating the FCA process that generated them in the first place. Because the compound labels are atomic, they cannot incorporate negation internally, as in ¬RICH_FAMOUS_BEAUTIFUL or RICH_¬FAMOUS_BEAUTIFUL, etc. Moreover, once we introduce temporal information into the individual type components, the whole scheme falls apart, because each predicate may have a different temporal validity. Finally, implementing efficient multiple inheritance coding schemes is notoriously difficult. All such problems reduce the usefulness of multiple inheritance, and suggest that we might benefit from eliminating complex type labels altogether, or at least reducing their number. We may attempt to do that by converting most or all of the compound types into *hanging monadic relations*.

We notice that a compound type usually consists of a natural type, e.g. FEMALE, linked to a chain of role or attribute types, e.g. RICH_FAMOUS_BEAUTIFUL. We can avoid compound types, and thus reduce multiple inheritance, by putting only natural types into the concepts, disassembling the rest of the compound type, and attaching its constituent predicates to the natural concept as hanging monadic relations, as in Fig. 1:

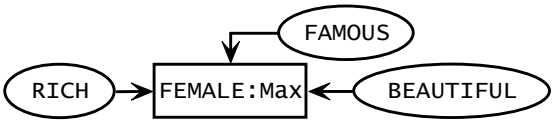


Fig. 1. Max is a rich, famous and beautiful female.

2.1 Knowledge Compilation

We may also generate hanging monadic relations by factoring out discrete categories from continuous or many-valued ranges of attribute values [6]. Such role types are defined from binary attribute relations. For example, assume that our knowledge base contains information about people's body measurements (height, chest, waist and collar circumference, arm and inside leg length, etc.). Combinations of value intervals for all these attributes are then mapped onto discrete categories, e.g. the 'standard' sizes used by the clothing industry. While the measurements are general and 'persistent', the size categories can be re-defined for different systems (US/UK/Euro, etc.) and purposes. Then we use only the hanging monadic relation, as in (UK-48-SHORT-WEARER)→[MALE:Pavel], instead of the complex graph describing Pavel's ample proportions. Similarly, we may store income information, for example, [MALE:Jim]→(INCOME)→[£@Year:90,000]. Then we may define various monadic relations that would categorize each person

as, for example, HIGHER-TAX-PAYER, MEDICAL-BILL-EXEMPT, TUITION-FEES-EXEMPT, etc. It is clear how monadic relations, such as TALL or RICH, can be defined, respectively, from the dyadic INCOME and HEIGHT. It is less obvious how we could define the monadic BEAUTIFUL, or similarly behaving predicates, from a dyadic relation. 'Aesthetic value' or 'attractiveness' is a legitimate attribute associated with a number of types. Beauty is in the eye of the beholder, but also subject to consensus, represented by some actual or virtual scoring system of preference rules (one such system can be found at http://www.hotornot.com/). Thus, we might define human beauty as
type BEAUTIFUL(x) is
[PERSON:*x]→(ATTRACTIVENESS)→[8<x≤10-HOTORNOT-SCORE].
This says that a beautiful person is one who scores over 8 points in the Hot-or-Not attractiveness rating. Assigning scores is subjective and loses information. A better method of 'beauty assessment', or of any other similar attribute, would be based on a more rigorous measurement of symmetry, proportionality, 'golden ratios', etc. The format of the definition would be the same, though. The criterion is whether it is more efficient to look up what category an individual belongs into rather than to calculate it each time from the underlying definitions and detailed data. We could view the use of monadic relations as 'knowledge compilation'.

The full semantics of an attribute relation label may depend on the type of object it is applied to. For example, the interpretation of the relation RICH, taken in isolation, is ambiguous. RICH has multiple different meanings, albeit related by metaphor, e.g., 'rich sauce', 'rich hair', 'rich colour', 'that is rich coming from you', etc. We could try to eliminate ambiguity by rigorously defining new, exclusive types that would have only one meaning, e.g. 'RICH-IN-VALUABLE-ASSETS', which would apply only to so endowed people or organizations. The 'richnesses' of sauces and hair would be defined in different ways. Alternatively, we may retain the same label for all the different uses, but include in the ontology rules that provide a different interpretation for each type that could appear in the relation's argument. The choice between the two methods depends on the trade-off between semantic transparency and computational cost.

2.2 Compounding of Hanging Monadic Relations

Isolating natural types only postpones compounding. We still need to dismantle any remaining compound role types, and have to prevent any further compounding of the now separate predicates. We can do so by adopting a rule that bans subtype formation by the conjunction of orthogonal[1] role or attribute predicates. For example, we will not accept a definition that compounds TALL and RICH: type TALL_RICH(x) is (TALL)→[PERSON:*x]←(RICH). Thus, the only way to specialize a monadic relation is by further qualifying the given property, not by adding a completely new quality.

[1] Orthogonality of predicates, i.e. their mutual independence, is a matter of degree. For example, there is a physical relation between TALL and HEAVY, and a statistical one between TALL and RICH. For this discussion, we assume that this makes no difference to the structure of the type lattice.

Eschewing combinations of orthogonal predicates yields hierarchies with only single or, at worst, limited multiple inheritance. The question remains whether we can avoid multiple inheritance altogether. Only detailed analysis of the semantics, pragmatics and representation of inheritance structures in actual domains will provide the answer.

Examples in Fig. 2 and Fig. 3 illustrate an interesting property of categories that map onto progressive sequences of subsets, rather than on disjoint sets of values. The shallow hierarchy in Fig. 2 appears to represent categories of asset ownership as being disjoint; each monadic predicate is defined from a discrete interval of monetary values. The top node, 'ASSETTED', say, represents an abstract relation that would be compulsory in the given domain (everybody has assets, even if they are zero or negative).

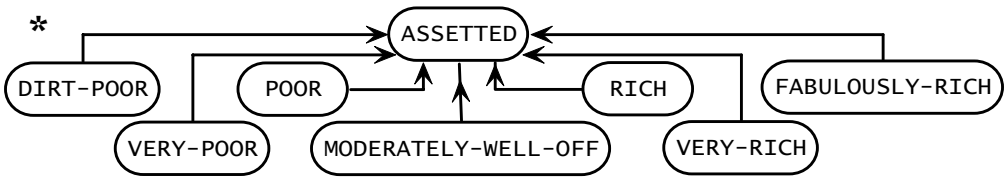


Fig. 2. Discrete categories of asset ownership

However, if we make a query, such as [PERSON:*x] ¬((POOR)→[PERSON:*x]), we would still get, apart from all the rich, also the very poor and dirt-poor. However, the monetary interval associated with DIRT-POOR is a sub-interval of VERY-POOR, which is a sub-interval of POOR. The stratified subsumption of such predicates is shown in Fig. 3.

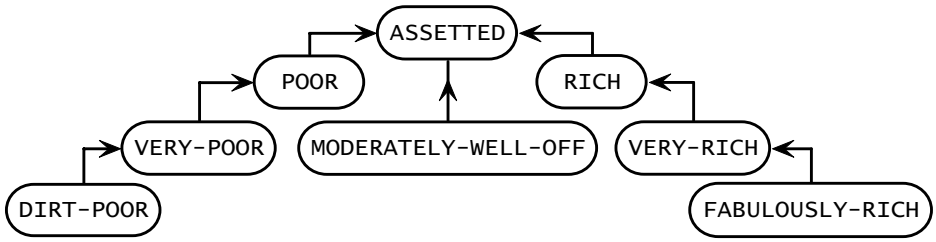


Fig. 3. The poor people subsume the very poor, who subsume the dirt-poor, etc.

Interestingly, the hierarchy in Fig. 3 seems to run counter to the pragmatic rule of taxonomy construction that warns against inheritance nodes with a branching factor of 1. The relation ASSETTED would not be used in a declarative sense, but only in a query. If we had no idea how well-off Max is, but wanted to know, projecting (ASSETTED)→[PERSON] into the knowledge base would return (FABULOUSLY-RICH)→[FEMALE:Max]. However, for such queries we have to extend the definition of CGs projection.

3 Projection for Hanging Monadic Relations

The original definition of projection in [2] did not formally include hanging monadic relations, nor relational inheritance. We may interpret projection on monadic relations in a conservative way, so that a single monadic relation in the query returns a single relation in the target. For example, assume that the graph in Fig. 4 is part of our knowledge base:

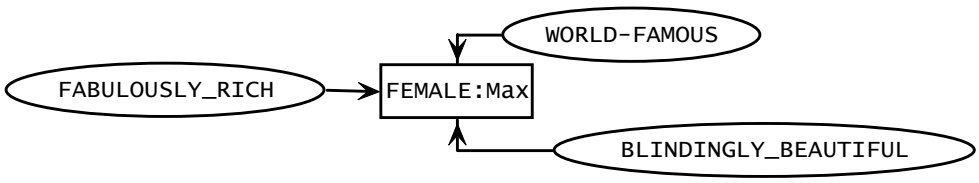


Fig. 4. Max is a fabulously rich, blindingly beautiful, world-famous female.

Then the conservative query (RICH)→[PERSON] would return (FABULOUSLY-RICH)→[FEMALE:Max], i.e. one source relation – one projection image. However, if our system did not have hanging monadic relations, but used compound types and multiple inheritance, the query [RICH-PERSON] would return the concept [FABULOUSLY-RICH_WORLD-FAMOUS_BLINDINGLY-BEAUTIFUL_FEMALE:Max].
This means that a conservative projection on a cluster of monadic relations would return less information than traditional projection would on an equivalent compound-type concept. We could say that each different combination of hanging relations attached to a given concept forms a virtual compound type. The query (RICH)→[PERSON] should really return not only the graph (FABULOUSLY-RICH)→[FEMALE:Max], but also (FABULOUSLY-RICH)→[FEMALE:Max]←(WORLD-FAMOUS), (FABULOUSLY-RICH)→[FEMALE:Max]←(BLINDINGLY-BEAUTIFUL), and the whole graph of Fig. 4. The new projection not only returns the direct sub-relation of each monadic relation in the query, but also all its combinations with all the other relations in the cluster of the target concept. This provides much more scope for the fine-tuning of queries, ranging from the minimum answer, through all the intermediate stages, to all the retrievable information. The maximalist approach might be desirable, for example, in a Criminal Intelligence Analysis system, where we want to know all there is about our suspects.

We may want to restrict our query in a fairly precise way, for example, to give different users different levels of access to the knowledge base, or avoid being swamped by irrelevant information. There are two basic approaches to query refinements: by *inclusion* – return just what we ask for and nothing else, and by *exclusion* – return everything but what we say we do not want. The query interface will expect control information specifying which type of query to perform. Thus, an inclusive query will return only direct specializations of the relations actually in the query, e.g. (RICH)→[PERSON] will return (FABULOUSLY-

RICH)→[FEMALE:Max], but say nothing about her fame and/or attractiveness, or any other possible attributes.

In an exclusive query we specify which classes of attributes we do not want to know about, but remain receptive to any other information that might be present in the knowledge base. For this the query interface will ask which relations to omit from the answer. To formulate such queries, a well-structured inheritance hierarchy for monadic relations is needed, because the superfluous relations in the answer will be pruned out by marking their top-level super-relations in the query. For example, assume that our ontology includes, apart from the hierarchy of wealth-related predicates, also inheritance lines for appearance-related and fame-related predicates. The local tops of these hierarchies might be labeled 'LOOKS' and 'FAMILIARITY'. They are analogous to ASSETTED, in that LOOKS subsumes UGLY, PLAIN, PRETTY, etc., and FAMILIARITY subsumes INFAMOUS, OBSCURE, UNKNOWN, FAMOUS, etc. Then, if we want to know how well off Max is, but are not interested in her glamour or notoriety, we would ask, imposing a procedural constraint on the query:

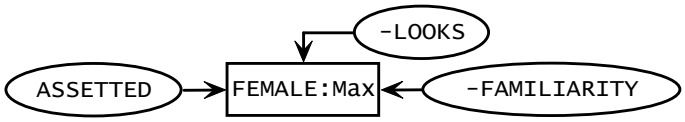


Fig. 5. How well off is Max (and don't bother about looks and fame)?

The minus signs in (-LOOKS) and (-FAMILIARITY) tell the system not to return any of their subrelations that might be found. Assuming that the knowledge base also includes information that Max is super-intelligent and a top athlete, which we did not know or ask about, the query in Fig. 5 will return the graph in Fig. 6:

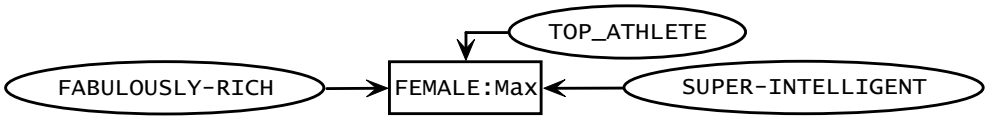


Fig. 6. Max is a fabulously rich, super-intelligent, top female athlete

Restricting a query in this way, or pruning what it brings back, requires good knowledge of the inheritance hierarchies of the ontology. However, as whole branches of eligible relations will be eliminated by the local top node, the user will only need to actively know the local top nodes, or find them using the help system. We can negate the individual hanging monadic relations, which would be impossible if we used compound types. The query in Fig. 7 will return all the females who are rich (or even super-rich, etc.), but not famous (let alone world-famous).

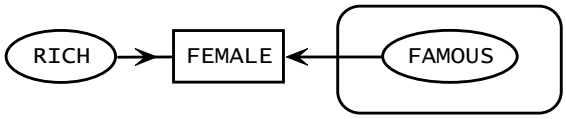


Fig. 7. Which females are rich, but not famous?

3.1 Virtual Compounding of N-Adic Relations

The disadvantages of compounded predicates, and the principles of decomposing them into individual atomic relations, also apply to CGs dyadic relations. This concerns primarily independent relations between individuals, not so much those between an individual and an attribute value. This is a situation where two different dyadic relations link the same object and the same attribute value. An exception to this would be a case where an individual has the same value for two attributes, e.g. the same width and length. For example, the three relations in Fig. 8 could be combined, using λ-abstraction or equivalence, into the single relation of Fig. 9.

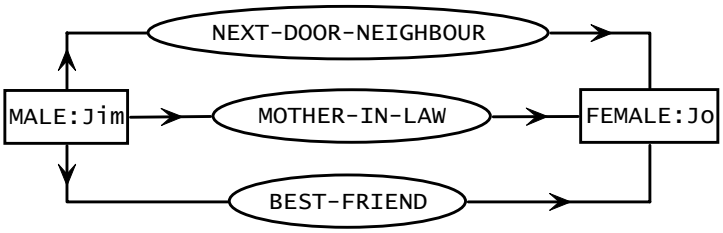


Fig. 8. Jo is Jim's next-door neighbour, mother-in-law and best friend (separately).

Fig. 9. Jo is Jim's next-door neighbour, mother-in-law and best friend (one relation).

With a multiple inheritance hierarchy for dyadic relations, any query that would use the relations CO-RESIDENT, ACQUAINTANCE, or RELATIVE, say, instead of the R in [PERSON]→(R)→[PERSON], would return the graph of Fig. 9, and therefore the whole graph, with all three dyadic relations, of Fig. 8 . This means that the projection operation for dyadic relations, and for certain types of triadic and tetradic relations, will require the same query control mechanism as that suggested earlier for monadic hanging relations.

3.2 Relations with Temporal Validity

By introducing temporal validity, monadic relations turn into triadic, and dyadic into tetradic, the two extra arguments being the time points when the relation started and ceased to be valid for the given argument koc03. The introduction of temporal information makes compound types even more objectionable, as each constituent predicate may be temporally independent of its neighbours in the compound type. If a CGs-based system uses temporal information, the projection algorithm will have to use it as well. Interestingly, there can be two different interpretations of projection between the time intervals of temporally qualified predicates: a) the time interval of the target graph (or rather its candidate) subsumes that of the query, and b) the time interval of the query subsumes that of the target. To use an example from Criminal Intelligence Analysis, assume that a crime occurred during the interval 17:00 – 18:00 (on a given day). Int_{Query} and Int_{Target} are the interval of the query and that of the retrieved graph, respectively. The query is in Fig. 10, LOC_{spc} stands for 'spatial location of'.

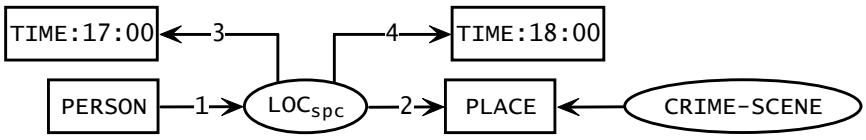


Fig. 10. What person was at the crime scene between 17:00 and 18:00?

$\text{Int}_{Query} \subseteq \text{Int}_{Target}$: the time interval of the query is a subset of the time interval associated with the events that took place at the crime scene. This corresponds to a situation where we are verifying the alibi of a potential suspect. Int_{Query} = 17:00 -18:00; the potential suspect will be cleared if she can prove to have been elsewhere during, at least, the interval Int_{Target} = 16:59 - 18:01.
$\text{Int}_{Target} \subseteq \text{Int}_{Query}$: the time interval associated with the crime events is a subset of the interval of our query. This would apply when searching for potential culprits. Anyone present at the crime location, for any sub-interval of the query interval, would be suspect.

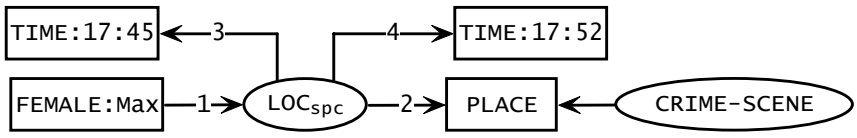


Fig. 11. Max was at the crime scene 17:45 - 17:52, and so could have done it.

If we interpret a temporal sub-interval as being a 'specialization' of its enclosing interval(s), then, interestingly, case *b* is more in the spirit of conventional projection: for any point in time that the 'evidence' graph is valid, the query is

valid. For case *a*, there may be time-points associated with the evidence graph that lie outside the query interval.

3.3 Joining Conceptual Graphs

In a system using compound types and relations, a CGs knowledge base is constructed by the maximal joins of valid, fully instantiated graphs, as entered, to the valid, fully instantiated graph of the previously entered data. In fact, this is a straightforward merge interlinking the union of two sets of individual concepts by means of the union of their respective sets of n-adic relations. During this process, the types of some concepts, and some relations, may have to be restricted to their sub-type/relation before they can be joined. In a system that uses clusters of hanging monadic relations and bundles of atomic n-adic relations, the merge-like nature of maximal join is even more pronounced, as the type/relation restriction is largely replaced by the addition of more relations into the cluster. In either system, for fully instantiated graphs, there is only one, unique result.

The conventional maximal join of uninstantiated graphs operates on a maximal, non-unique 'typed intersection' of its input graphs. This strict intersection rule maximizes commonality between the input graphs, and thus relevance of the newly formed graph. However, we see the main purpose of joining (or maximally joining) uninstantiated graphs as the main mechanism of generating hypotheses. In that case, in a system with clustered atomic relations, we may want to regulate or maximize the amount of information that goes into the newly formed graph.

In our criminal domain, the system would join together graphs representing partial or unreliable evidence, while constantly checking for violation of semantic constraints. Then, using domain-specific preference rules, it would identify parts of the newly formed graph in need of further specialize by type and/or referent restriction, or by adding new relations (monadic or n-adic) to the relevant clusters. On the basis of such identified gaps, the system will ask for specific evidence that would help it to further populate and specialize the graph. The hypothesis is proved when its graph is fully instantiated, its predicates maximally restricted, and all the required relations added into the relevant clusters: the case solved. For example, in Fig. 12 graphs *a* and *b* represent the testimonies of two witnesses.

In Fig. 13 we have the hypothesis, formed by joining the two testimonies, namely, that the two people so described are the same person. Note that the result of a conventional join would be only (SMELLY)→[MALE], linked by the (AGNT) and (PTNT) relations to the rest of the newly formed graph.

In a system with multiple inheritance, a generic concept on which graphs A and B are joining may be 'outside' either graph; it is a common specialization of its respective sources in A and B, but not type-identical to either of them. Fig. 14 shows an extreme case where the type of every concept in the join is a proper common subtype of either of its parents.

Hypothesis formation is a flexible, incremental process guided by preference rules, and guarded by the Canonical Models [4]. The preference rules determine in what order the validity of concepts, of the hanging monadic and n-adic relations is tested. The Canonical Models are used to check for semantically incompatible combinations of both monadic and n-adic relations, preventing, for example, the join of graphs such as (TALL)→[PERSON] and (SHORT)→[PERSON].

4 Canonical Models

Canonical Models, together with the projection operation, are the building blocks of a CGs ontology. (Canonical models are not related to the Canonical Formation Rules presented in [2] and [5]). They enforce the general semantic and domain-specific rules and patterns during the construction of the knowledge base, hypothesis generation, and reasoning. Associated with every single predicate (concept type, monadic, or general n-adic relation), there is a set of canonical models that define the conditions under which the given predicate/relation can be used. Canonical models are projected into candidate graphs to test their semantic correctness. There are two types of canonical models: positive and negative.

The *Positive Canonical Model* for a given predicate represents the least common generalization of all the occurrences of the predicate in the knowledge base. It can be also defined prescriptively, like laws and regulations. In principle, for any given true or otherwise semantically correct graph, all the positive canonical models for all its predicates must successfully project into it. Each graph entered into, or generated by the system is thus checked by all the relevant positive canonical models. If a positive canonical model does not project into the candidate graph, the graph is rejected, or offered for modification. However, positive canonical models may also project into graphs that contain exceptions to the general semantic rules.

The *Negative Canonical Model* will detect any semantically flawed uses of predicates, as described by the exceptions to the positive models. Accordingly, every predicate of the domain ontology is associated with a set (possibly empty) of negative canonical models, each being the least common generalization of all the occurrences of one type of semantic exception. Alternatively, negative models may be prescriptively defined. Thus, if a negative canonical model projects into a graph that has passed the positive model test, the graph is rejected, or a change suggested.

Canonical models should detect any semantically malformed and therefore false graphs. The logical definition of a predicate, together with all the canonical models, provides a comprehensive description of the predicate's structure, behaviour, and relation to the rest of the ontology.

In knowledge compilation, we replace a complex graph with a single hanging monadic, or n-adic relation. For every defined predicate, its canonical models are generated, recursively, from the body of its definition (and from the definitions of the predicates in the body, etc.), and passed on to the predicate, possibly in a modified form. Thus, a predicate inherits its canonical models from its logical

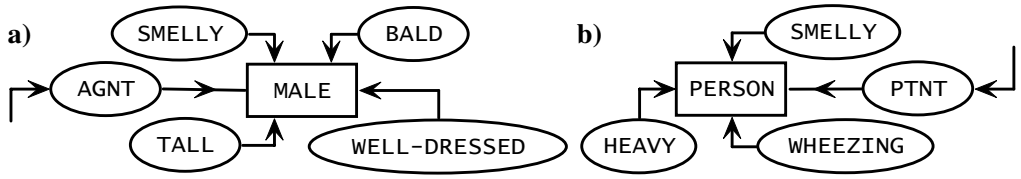


Fig. 12. a) Witness A saw a tall, bald, smelly yet well-dressed male being the agent of some act; b) Witness B (a blind person) reported a heavy, wheezing and malodorous person being subject to some event.

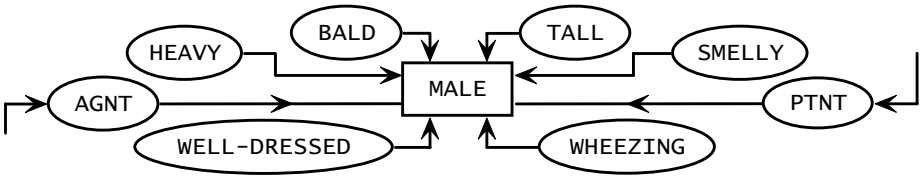


Fig. 13. A join (merge) of graphs a and b of Fig. 12.

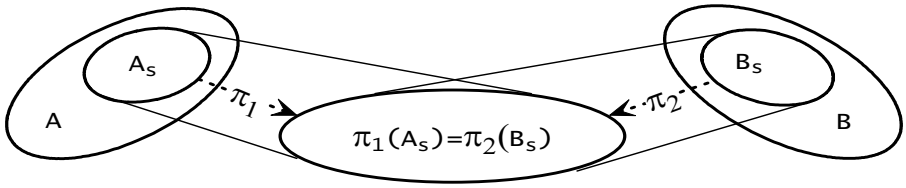


Fig. 14. No concept or relation of $\pi_1(A_s)=\pi_2(B_s)$ actually lies in A or B.

In a system lacking multiple inheritance, the projection image of a concept in A is always in B, and vice versa, as shown in Fig. 15. Thus, when constructing a graph to join on, we search for two projections: one from A to B, the other from B to A. Then we recursively 'consolidate' the subraphs by substituting more general types or relations with their subtypes or relations from the projection images, and eventually merge the images to produce the join 'core', and the (maximally) joined graph itself.

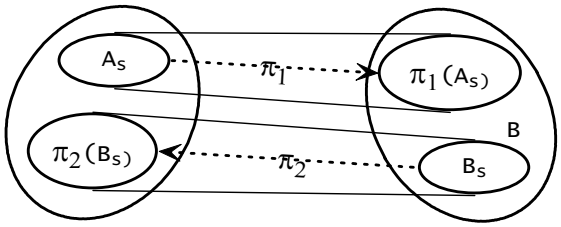


Fig. 15. Joining graphs under exclusively single inheritance

definition. Canonical models are not necessarily applied through conventional theorem proving, but by means of direct projection. Assume our ontology contains the monadic relation WIFE, which might be defined as relation WIFE(x) is [FEMALE:*x]←(SPOUSE)←[MALE]. Going back in the chain of definitions, we have defined SPOUSE from the event of the wedding, including various conditions that make a wedding legally possible.

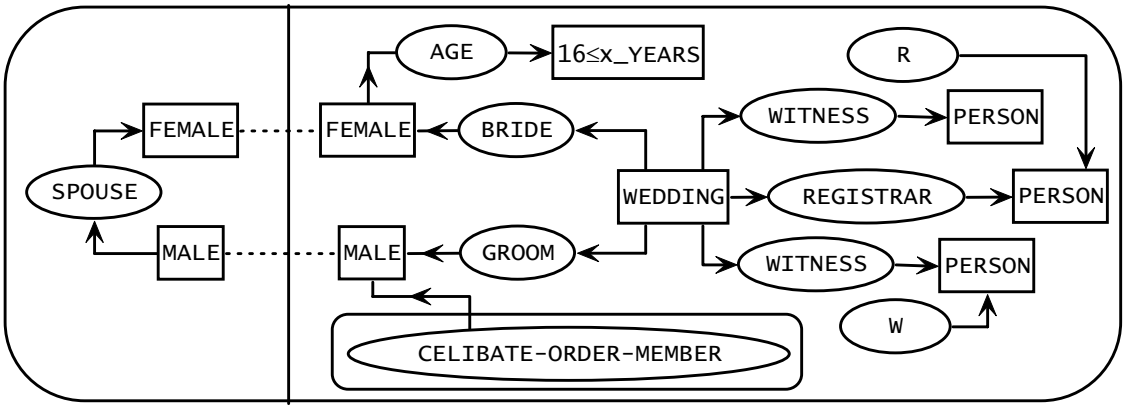


Fig. 16. Partial definition: A male and a female are spouses if they had a wedding, such that ...

This is a partial definition, omitting the place and time of the wedding, constraints on polygamy or polygyny, legal incompetence, consanguinity, etc. It only specifies the genders of the bride and groom, their age limit, and constraints on members of religious orders. The positive canonical model for SPOUSE is, say, [16≤x_YRS]←(AGE)←[MALE]→(SPOUSE)→[FEMALE]→(AGE)→ [16≤x_YRS]. It will detect same-sex and child marriages, non-human partners or inanimate objects. The relations R and W in Fig. 16 specify the qualifications for becoming a registrar and a wedding witness, respectively. The relation labels REGISTRAR, WITNESS, and the type WEDDING are not present in the definiendum, so none of their conditions (canonical models), such as those contained in the relations R and W, will be propagated to the relation SPOUSE. We know that SPOUSE is a one-to-one relation, so we formulate a negative canonical model that must not project: [PERSON]←(SPOUSE)←[PERSON]→(SPOUSE)-→[PERSON]. We may ignore gender in this, as we have already enforced gender constraints by the positive model. Another negative canonical model represents the constraint on celibate orders: (CELIBATE-ORDER-MEMBER)-→[PERSON]→(SPOUSE)→[PERSON], which must not project. The monadic hanging relation WIFE is defined from SPOUSE. Because only the type FEMALE is present in the definiendum, WIFE inherits only the canonical models that applied to the bride/married female in the previous definitions. Any canonical models for the groom/married male may be dropped. This outline encapsulates our methodology for the construction of canonical models, based on their link to the logical definitions of predicates.

5 Conclusion

Using as examples the oldest and most extensive 'real' inheritance structure, namely, the Linnean taxonomy, and the future-oriented PhyloCode, we have pointed out the potential instability in the development of any non-trivial ontology, and the impermanent nature of the resulting systems. We have emphasized the problems of multiple inheritance, and outlined the advantages of hanging monadic relations. We have suggested extensions to the CGs projection operation that cater for the behaviour of not only hanging monadic relations, but also that of clustering atomic n-adic relations. We suggested and discussed modifications to the operations of join or maximal join on monadic and clustering n-adic relations under single inheritance, and discussed and illustrated the use of joins in hypothesis generation. To provide a solid basis for tools and methodologies for the development of ontology-based systems, the theory of knowledge representation by CGs requires further refinement, clarification, and possible extensions. This goal subsumes several major tasks, concerning theory, implementation, and empirical validation. The theoretical part requires full treatment, possibly along the lines and scope of [7], but that is outside the confines of this paper. We must stress that only extensive empirical analysis, as part of the construction of real-world ontologies, will test the proposed theory and methodology, and that will be made feasible by a principled, robust computer system based on these ideas.

References

1. http://www.ohiou.edu/phylocode/
2. Sowa, John F.: Conceptual Structures: Information Processing in Mind and Machine, Addison-Wesley. (1984), ISBN 0-201-14472-7.
3. Kocura, P.: Representing Temporal Ontology in Conceptual Graphs, In: Conceptual Structures for Knowledge Creation and Communication, Proc. 11th Int. Conf. on Conceptual Structures, ICCS 2003, Dresden, Germany, July 2003, De Moor, A., Lex, W., Ganter, B. (eds.), LNAI 2746, Springer, ISSN 0302-9743, ISBN 3-540-40576-3pp 174-187.
4. Kocura, P.: Conceptual Graphs Canonicity and Semantic Constraints, In: Proc. 4^{th} Int. Conf. on Conceptual Structures (b), ICCS'96, Sydney, Australia, ISBN 0 7334 1387 0, pp 133-145.
5. The updated version of the draft conceptual graph standard developed by the NCITS.T2 Committee on Information Interchange and Interpretation, http://www.jfsowa.com/cg/cgstandw.htm
6. Kocura, P.: Semantics of Attribute Relations in Conceptual Graphs, Conceptual Structures: Logical, Linguistic and Computational Issues, Proc. 8th Int. Conf. on Conceptual Structures, ICCS 2000, Darmstadt, August 2000, Ganter, B., Mineau, G.W. (eds.), LNAI 1867, Springer, ISBN 3-540-67859-X, pp 235-248.
7. Mugnier, M.-L.: A Graphs-based Approach to Knowledge Representation and Reasoning, 2003, http://www.lirmm.fr/∼mugnier/.

The Semantics of Confusion in Hierarchies: Theory and Practice

Serguei Levachkine[1], Adolfo Guzman-Arenas[1,2], and Victor-Polo de-Gyves[2]

[1]Centre for Computing Research (CIC) - National Polytechnic Institute (IPN)
UPALMZ, CIC Building, 07738, Mexico City, MEXICO
[2]SoftwarePro International
sergei@cic.ipn.mx, a.guzman@acm.org, degyves@gmail.com

Abstract. Conceptual knowledge embedded in symbolic values (such as `software`, `computer systems`, `Asia`, `Japan`) is formally represented in a tree of sets known as a *hierarchy*. Similarities (or rather, errors) among values belonging to a hierarchy are gauged by a function: conf (*r*, *s*) measures the degree of confusion when (symbolic) value *r* is used instead of (the correct, or intended) value *s*. Intuitively, the error in using `nurse` instead of `physician` is smaller than the error if `hospital` or `avocado` were used. In addition to being easy to use, *conf* allows the definition of other functions: *Identical*, when two symbolic values are identical. *Substitute*, when a symbolic value may replace another, with confusion 0. *Very similar*. *Similar*. *Somewhat similar*. *Equality up to a given confusion*: $r=_\varepsilon s$ (*r* is equal to *s* up to confusion ε). More over, it is possible to extend a predicate P(*o*) over an object *o*, which is *true* if *o* fulfils P, to "P_ε (o)", read "P holds for *o* within confusion ε", which is *true* if *o* fulfils P up to confusion ε. Once we have allowed *confusion* to enter predicates, we can extend a relational database, granting it the capability to retrieve objects within a given confusion. Hence, the exposed theory acquires practical dimensions. This is done by introducing *extensions* to SQL, so that a user can express queries involving confusion. Later, these *extensions* are removed by a parser that takes an extended SQL sentence and produces "pure" SQL, but where confusion is handled satisfactorily, through tables in memory. Our implementation shows that these conceptual structures and formal representations can support human communication and use; in fact, we give (simple) examples of inexact retrieval.

1 Introduction

What wearing apparel do we wear for rainy days? *Raincoat* is a correct answer; *umbrella* is a close miss; *belt* a fair error, and *typewriter* a gross error. What is closer to an *apple,* a *pear* or a *caterpillar?* Can we measure these errors and similarities? How related or close are these words? Some preliminary definitions follow.

Element set. *(Definition)* A set E whose elements are explicitly defined. ♦[1] *Example*: {*red, blue, white, black, pale*}.
Ordered set. *(Definition)* An element set whose values are ordered by a < ("less than") relation. ♦ *Example*: {*very_cold, cold, warm, hot, very_hot*}.

Covering. *(Definition)* K is a covering for set E if K is a set of subsets $s_i \subseteq E$, such that $\cup s_i = E$. ♦[1] Every element of E is in some subset $s_i \in K$. If K is not a covering of E, we can make it so by adding a new s_j to it, named "others", that contains all other elements of E that do not belong to any of the previous s_i.

Exclusive set. *(Definition)* K is an exclusive set if $s_i \cap s_j = \varnothing$, for every $s_i, s_j \in K$. ♦ Its elements are mutually exclusive. If K is not an exclusive set, we can make it so by replacing every two overlapping $s_i, s_j \in K$ with three: $s_i - s_j$, $s_j - s_i$, and $s_i \cap s_j$.
Partition. *(Definition)* P is a partition of set E if it is both a covering for E and an exclusive set.
Qualitative variable. *(Definition)* A single-valued variable that takes symbolic values. ♦ Its value cannot be a set.[2] By symbolic we mean qualitative, as opposed to numeric, vector or quantitative variables.
(Definition) A symbolic value v **represents** a set E, written $v \propto E$, if v can be considered a name or a depiction of E. ♦ *Example*: *pale* $\propto$ {*white, yellow, orange, beige*}.

1.1 Hierarchy

(Definition). For an element set E, a **hierarchy** H of E is another element set where each element e_i is a symbolic value that represents either a single element of E or a partition, and $\cup_i \{r_i \mid e_i \propto r_i\} = E$ (The union of all sets represented by the e_i is E). ♦ *Example* (Hierarchy H_1): for E = {*Canada, USA, Mexico, Cuba, Puerto_Rico, Jamaica, Guatemala, Honduras, Costa_Rica*}={a, b, c, d, e, f, g, h, i}, a hierarchy H_1 is {*North_America, Caribbean_Island, Central_America*}={H_1^1, H_1^2, H_1^3}, where *North_America* $\propto$ {*Canada, USA, Mexico*}; *Caribbean_Island* $\propto$ {*Spanish_Speaking_Island, English_Speaking_Island*}={H_1^{21}, H_1^{22}}; *Spanish_Speaking_Island* $\propto$ {*Cuba, Puerto_Rico*}; *English_Speaking_Island* $\propto$ {*Jamaica*}; *Central_America* $\propto$ {*Guatemala, Honduras, Costa_Rica*}.

Hierarchies make it easier to compare qualitative values belonging to it (§3).
(Def.) A **hierarchical variable** is a qualitative variable whose values belong to a hierarchy (The data type of a hierarchical variable is hierarchy). ♦ *Example*: *place_of_origin* that takes values from H_1. Note: hierarchical variables are single-valued.

---

[1] The symbol ♦ means: end of definition.
[2] Variable, attribute and property are used interchangeably. An object may have an attribute (Ex: weight) while others do not: the weight of blue *does not make sense*, as opposed to saying that the weight of blue *is unknown* or not given. A variable (*color*, *height*) describes an aspect of an object; its value (*blue*, *2 Kg*) is such description or measurement.

Thus, a value for *place_of_origin* can be *North_America* or *Mexico,* but not {*Canada, USA, Mexico*}, although *North_America* ∝ {*Canada, USA, Mexico*}.

1.2 Notation

The sets represented by each element of a hierarchy form a tree under the relation subset. *Example*: for H_1, such tree is given in Figure 1.

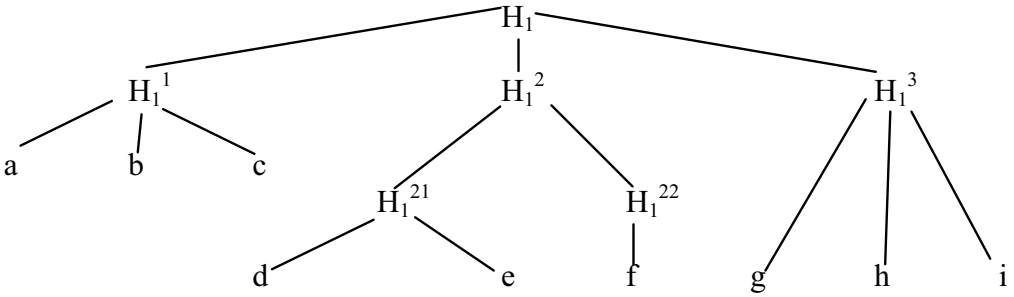


Fig. 1. The tree induced by hierarchy H_1.

We will also write a hierarchy such as H_1 thus: {*North_America* ∝ {*Canada USA Mexico*} *Caribbean Island* ∝ {*Spanish_Speaking_Island* ∝ {*Cuba Puerto_Rico*} *English_Speaking_Island* ∝{*Jamaica*} } *Central_America* ∝ {*Guatemala Honduras Costa_Rica*} }.

(Definition) **father_of** (*v*). In a tree representing a hierarchy (such as H_1), the father_of a node is the node from which it hangs. ♦ Similarly, *(Definition)* the **sons_of** (v) are the values hanging from *v*. The nodes with the same father are **siblings**. ♦
(Definition) Similarly, **grand_father_of**, **brothers_of, aunt**, **ascendants**, **descendants**... are defined, when they exist. ♦
(Definition) The **root** is the node that has no father.♦

2 Previous Work

Measures of the similarity between variables [5] have been used, using Kendall, Hamming, Russel & Rao, Jaccard, Kuzlinsky, Yule and other distances, and a contingency table. Being *distances,* they are symmetric: d(*a*,*b*)=d(*b*,*a*), while we will see that in a hierarchy the "similarity" between *a* and *b* is not the same as that between *b* and *a*. For these reasons, we introduce a new term, *confusion.*

Non-symmetrical comparison on a tree (an ontology) occurs in [1], where each agent possesses its own ontology of concepts. Each agent uses an ontology comparator COM, that maps a concept from one ontology (through comparison of words in a natural language) into the closest corresponding concept of another ontology. COM achieves communication without need of a common or *standard ontology*.

A datum makes sense only within a context. Intuitively, we know that "computer" is closer to "office" than to "ocean" or to "dog." A "cat" is closer to "dog" than to "bus station." "Burning" is closer to "hot" than to "icy." How can we measure these similarities?

A hierarchy describes the structure of qualitative values in a set *S*. A **(simple, normal) hierarchy** is a tree with root *S* and if a node has children, these form a partition of the father [2]. A simple hierarchy describes a hierarchy where *S* is a set (thus its elements are not repeated, not ordered). For example, live being{animal{mammal, fish, reptile, other animal}, plant{tree, other plant}}. In a **percentage hierarchy**, the size of each set is known[3]. For instance, AmericanContinent(640M){North America(430M) {USA(300M), Canada(30M), Mexico(100M)} Central America (10M), South America(200M)}. In an **ordered hierarchy** [3], the nodes of some partitions obey an ordering relation: object{tiny, small, medium, large}* [4]. Finally, a **mixed hierarchy** combines the three former types. Other works related to retrieval of approximate answers are referenced in [4].

For these four types of hierarchies we define conf(*r*, *s*) as the confusion or error in using value *r* instead of *s*, the intended or correct value. These definitions agree with the human sense of estimation in closeness for several wrong but approximate answers to a given question; each is applicable to particular endeavors.

Then, we define an enriched SQL syntax that deals with approximate queries on elements in a database holding qualitative values hierarchically structured. This enriched SQL uses precision-controlled predicates. Finally, we explain how the extension (to precision-controlled retrieval) of *any* database is possible.

3 Properties and Functions on Hierarchies

I ask for a *European car*, and I get a *German car*. Is there an error? Now, I ask for a *German car*, and a *European car* comes. Can we measure this error? Can we systematize or organize these values? Hierarchies of symbolic values allow measuring the similarity between these values, and the error when one is used instead of another.

3.1 Confusion in using r instead of s, for simple hierarchies

(Definition) If $r, s \in H$, then the **confusion** in using *r* instead of *s*, written conf(*r*, *s*), is:

- conf (*r*, *r*) = conf (*r*, *s*) = 0, where *s* is any ascendant of *r*; **(1)**
- conf (*r*, *s*) = 1 + conf (*r*, father_of(*s*)) . ♦ **(2)**

[3] The size of each set is written in parenthesis after the set. Here is the number of inhabitants.
[4] Notation: an * is placed at the end of the partition, to signify that it is an *ordered* partition.

To measure conf, count the *descending* links from *r* to *s*, the replaced value. The function conf is not a *distance*, nor *ultradistance*, nor it is symmetric. To differentiate from other known distances, symmetric measures, linguistic terms like relatedness, closeness, similarity or semantic distance, we introduce a new term: **confusion**.

Example (Hierarchy H_2): conf(*r*, *s*) for H_2 of Figure 2 is given in Table 1:

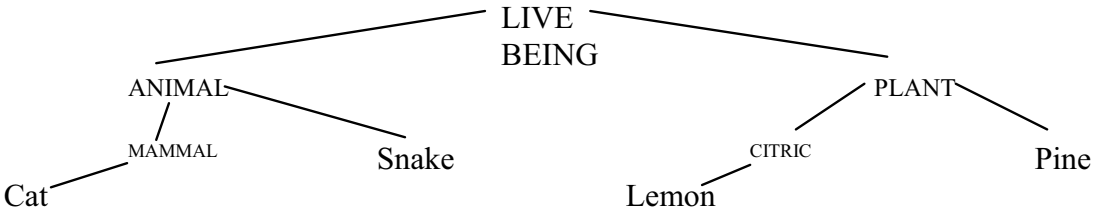


Fig. 2. A hierarchy H_2 of live beings. conf(cat, mammal)= 0; conf(mammal, cat)= 1.

Table 1. conf(*r*, *s*): Confusion in using *r* instead of *s* for the live beings of H_2.

r \ *s*	Live b.	Animal	Plant	Mam.	Snake	Citric	Pine	Cat	Lemon
Live b.	0	1	1	2	2	2	2	3	3
Animal	0	0	1	1	1	2	2	2	3
Plant	0	1	0	2	2	1	1	3	2
Mam.	0	0	1	0	1	2	2	1	3
Snake	0	0	1	1	0	2	2	2	3
Citric	0	1	0	2	2	0	1	3	1
Pine	0	1	0	2	2	1	0	3	2
Cat	0	0	1	0	1	2	2	0	3
Lemon	0	1	0	2	2	0	1	3	0

The confusion thus introduced *resembles reality* and *catches the hierarchy semantics*. For example, conf(animal, live_being) = 0: if they ask you for a live being and you give them an animal, the error of using animal instead of live being is 0, since all animals are live beings. Giving a live being when asked for an animal has error 1; conf(live_being, animal) = 1. The confusion among two brothers (say, dog and cat) is 1; using a son instead of the father produces conf=0; using the father instead of the son makes conf = 1. conf obeys the triangle law: conf(*a*,*c*) ≤ conf(*a*,*b*) + conf(*b*, *c*).

3.1.1 Confusion in using r instead of s, for ordered hierarchies

(Definition) For hierarchies formed by sets that are lists (ordered sets; example *Temp*={icy, cold, normal, warm, hot, burning}), the **confusion** in using *r* instead of *s*, conf'' (*r*, *s*), is defined as:

- conf'' (*r*, *r*) = conf (*r*, any ascendant of *r*) = 0;
- If *r* and *s* are distinct brothers, conf'' (*r*, *s*) = 1 if the father is not an ordered set; else, conf'' (*r*, *s*) = the relative distance from *r* to *s* = the number of steps

needed to jump from *r* to *s* in the ordering, divided by the cardinality-1 of the father; **(3)**

- conf'' (*r*, *s*) = 1 + conf''(*r*, father_of(*s*)). ♦

This is like conf for *hierarchies formed by sets*, except that there the error between two brothers is 1, and here it is a number ≤ 1. *Example*: in the list *Temp*, conf'' (icy, cold) = 1/5, while conf'' (icy, burning) = 5/5.

3.1.2 Confusion in using *r* instead of *s*, for percentage hierarchies

Now consider a hierarchy *H* (of an element set *E*) but composed of bags (unordered collection where repetitions are allowed) instead of sets.

(Definition) For bags, the **confusion** in using *r* instead of *s*, conf' (*r*, *s*), is:

- conf' (*r*, *r*) = conf'(*r*, *s*) = 0, when *s* is any ascendant of *r*;
- conf' (*r*, *s*) = 1 – relative proportion of *s* in *r*. ♦[5] **(4)**

Example: If *baseball_player* = {*pitcher catcher base_player* ∝ {*baseman baseman baseman*} *field_player* ∝ {*fielder fielder fielder*} *shortstop*} then (a) conf' (*fielder, baseball_player*) = 0; (b) conf' (*baseball_player, fielder*) = 1 – 1/3 = 2/3; (c) conf' (*baseball_player, left_fielder*) = 8/9 (a *left_fielder* is one of those three fielders); (d) conf' (*base_player, fielder*) = 2/3.

3.1.3 Confusion in using *r* instead of *s*, for mixed hierarchies

(Definition) To compute conf(*r*, *s*) in a mixed hierarchy, a simple-minded way[6] is:

- apply rule (1) to the *ascending* path from *r* to *s*;
- in the descending path, use rule (3) instead of rule (2), if p is an ordered set[7]; or use rule (4) instead of (2), when sizes of p and q are known. ♦ That is, use (4) instead of (2) for percentage hierarchies.

This definition is consistent with and reduces to previous definitions for simple, ordered, and percentage hierarchies. The paper will derive results for conf; those for conf' and conf'' can be similarly derived.

3.2 The set of values that are equal to another, up to a given confusion

(Def.) A **value *u* is equal to value *v*, within a given confusion ε**, written $u =_\varepsilon v$, iff conf(*u*, *v*) ≤ ε (It means that value *u* can be used instead of *v*, within error ε). ♦

---

[5] Number of elements of *S* that are in *r* and that also occur in *s* / number of elements of *S* that are also in *r* = relative popularity or percentage or proportion of *s* in *r*.

[6] In certain sense, there is no "proper" way to mix apples and oranges.

[7] Here, *p* and *q* are two consecutive elements in the path from *r* to *s*, where *q* immediately follows *p*. That is, *r* → …*p*→*q*… →*s*.

Example: If v = lemon (Figure 2), then (a) the set of values equal to v with confusion 0 is {lemon}; (b) the set of values equal to v with confusion 1 is {citric lemon}; (c) the set of values equal to v with confusion 2 is {plant citric pine lemon}. Notice that $=_\varepsilon$ is neither *symmetric* nor *transitive*.

3.2.1 Queries

Objects possessing several properties (or variables), some of them perhaps hierarchical variables, can best be stored as rows of a table in a relational database. We now extend the notion of queries to tables with hierarchical variables,[8] by defining the set S of objects that satisfy predicate P within a given confusion ε.

(Definition) ***P* holds for object *o* with confusion ε**, or P holds for o within ε,
(1) when P is formed by non-hierarchical variables, iff P is true for o;
(2) when pr is a hierarchical variable and P is of the form (pr = c), iff for value v of property pr in object o, $v =_\varepsilon$ c (if the value v of the object can be used instead of c with confusion ε);
(3) when P is of the form P_1 P_2, iff P_1 holds for o within ε or P_2 holds for o within ε;
(4) when P is of the form $P_1 \wedge P_2$, iff P_1 holds for o within ε and P_2 holds for o within ε;
(5) when P is of the form P_1, iff P_1 does not hold for o within ε. ♦

Example 1 (refer to hierarchies H_1 and H_2 above): Let us define predicates P = (lives_in = USA) (pet = cat), Q = (lives_in = USA) ∧ (pet = cat), $R = \neg$ (lives_in = Spanish_Speaking_Island); and objects (*Ann* (lives_in USA) (pet snake)), (*Bill* (lives_in English_Speaking_Island) (pet citric)), (*Fred* (lives_in USA) (pet cat)), (*Tom* (lives_in Mexico) (pet cat)), (*Sam* (lives_in Cuba) (pet pine)). Then we have the following results (Table 2): represents

Table 2. How the predicates P, Q and R of example 1 hold for several objects.

	***P* holds within ε for:**	***Q* holds within ε for:**	***R* holds within ε for:**
ε = 0	Ann, Fred, Tom	Fred	Ann, Bill, Fred, Tom
ε = 1	Ann, Fred, Tom	Fred, Tom	Ann, Fred, Tom
ε = 2	Ann, Fred, Tom, Sam	Ann, Fred, Tom	Nobody

3.2.2 The smallest ε for which P(*o*) is true

How close is Tom to be like Ann in Example 1? Ann lives in the USA and her pet is a snake, while Tom lives in Mexico and his pet is a cat. When we apply S = (lives_in = USA)∧(pet=snake) to Tom,[9] we see that S starts holding for ε=1. The answer to "How close is Tom to Ann?" is 1. Now we ask: How close is Ann to Tom? Ann[9] is

[8] For non-hierarchical variables, a match in value means conf = 0; a mismatch means conf = ∞

[9] Here predicate S = (lives_in=USA)∧(pet=snake) represents "to be like Ann." That is, from the definition of Ann (in example 1), we construct predicate *S.* Similarly, predicate (lives_in= Mexico)∧(pet=cat) embodies "to be like Tom" (see Tom in example 1).

close to Tom starting from ε=2; that is, (lives_in=Mexico)∧(pet=cat) does not hold for Ann at ε=1, but it starts holding for her at ε=2. This hints how to define the "*closeness to,*" a number (a confusion) between an object *o* and a predicate *P*.
(Definition) Object *o* **ε-fulfills** predicate *P* at threshold ε, if ε is the smallest number for which *P* holds for *o* within ε. ♦
(Definition) Such smallest ε is the **closeness** of *o* to *P*. ♦ It is not symmetric.
Closeness is an integer number defined between an object and a predicate. The closer is ε to 0, the "tighter" *P* holds. Compare with the *membership function* for fuzzy sets.

3.3 Confusion between variables (not values) that form a hierarchy

What could be the error in "Sue directed the thesis of Fred", if all we know is "Sue was in the examination committee of Fred"? Up to now, the *values* of a hierarchical variable form a hierarchy (Cf. §1.1). Now, consider the case where the *variables* (or relations) form a hierarchy. For instance, relative and brother, in a universe of kinship relations E = {sister, aunt…}. Consider hierarchies H_3 and H_4: (H_3) relative ∝ {close_relative ∝ {father mother son daughter brother sister} mid_relative ∝ {aunt uncle niece cousin} far_relative ∝ {grandfather grandmother grandson granddaughter grandaunt granduncle grandcousin grandniece} }; (H_4) player ∝ {soccer_player ∝ {John Ed} basketball_player ∝ {Susan Fred} }.

In hierarchy H_3, conf (son, relative) = 0; conf (relative, son) = 2. We know that, for object (*Kim* (close_relative Ed) (pet cat)), the predicate *V* = (close_relative Ed) holds with confusion 0. It is reasonable to assume that *W* = (son Ed) holds for Kim with confusion 1[10]; that *X* = (relative Ed) holds for Kim with confusion 0. Moreover, since Ed is a member of hierarchy H_4, it is reasonable to assume that for object (*Carl* (close_relative soccer_player) (pet pine)) the predicate *V* holds with confusion 1, *X* holds with confusion 0+1 = 1 and *W* holds with confusion 1+1 = 2. Thus, we can extend the definition of queries to variables that are members of a hierarchy, by adding another bullet to the definition of §3.2.1, thus:

« (6) When *P* is of the form (*var* = c), for *var* a variable member of a hierarchy, iff ∃ variable var_2 for which (var_2=c) holds for *o* within ε – conf (var_2, *var*), where var_2 also belongs to the hierarchy of *var*.♦ »

The confusion of the variables *adds* to the confusion of the values. *Example*: For (*Burt* (relative basketball_player) (pet cat)), *V* holds with confusion 1+2=3, *W* with confusion 2+2=4, and *X* with confusion 2+0=2.

3.4 Objects' similarity and accumulated confusion

The three hierarchies of figures 3-5 will allow us to introduce other new concepts such as identical, substitute, similar, etc. and accumulated confusion.

[10] We are looking for a person that is a son of Ed, and we find Kim, a close relative of Ed.

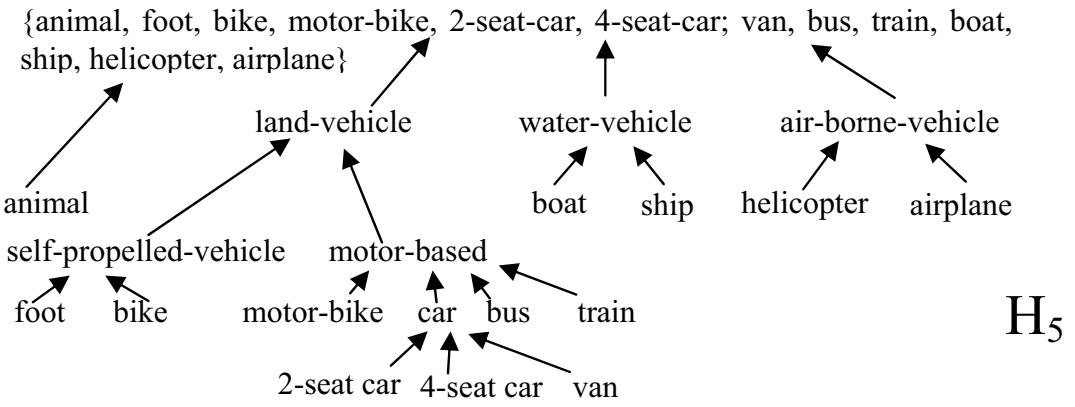


Fig. 3. A hierarchy H_5 of transportation vehicles. Some qualitative values, like air-borne-vehicle, represent sets: {helicopter, airplane} in our example

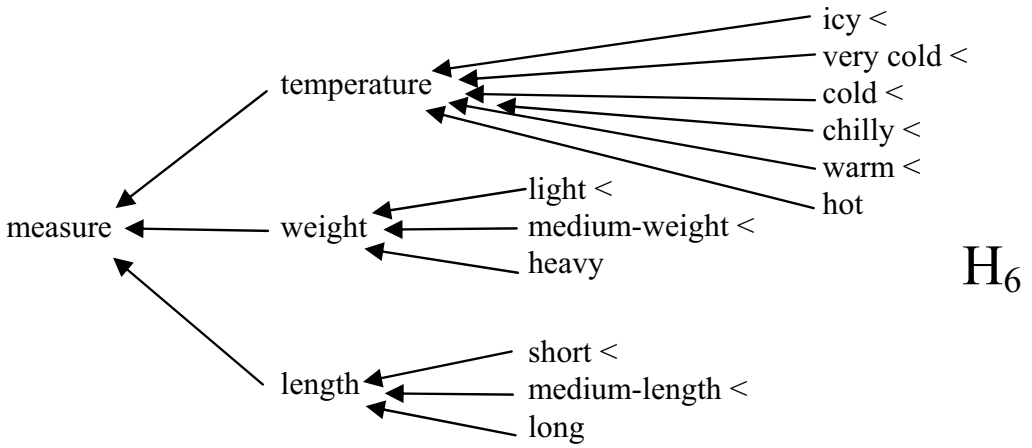


Fig. 4. H_6. A hierarchy having some ordered sets: (short < medium-length < long), (light < medium-weight < heavy), (icy < very cold < cold < chilly < warm < hot)

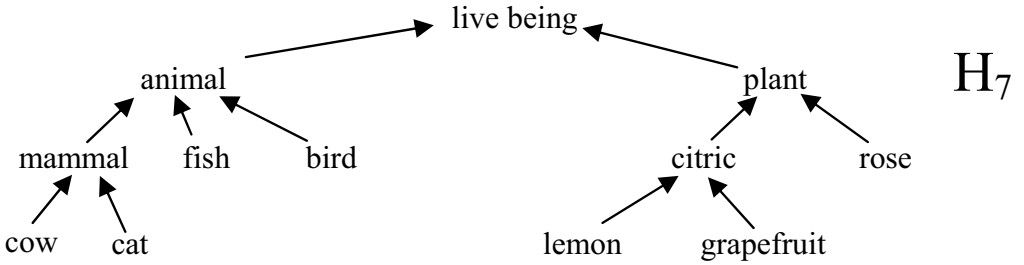


Fig. 5. A hierarchy H_7 of living creatures.

3.4.1 Identical, very similar, somewhat similar objects.

(Definition) **Objects** are entities described by k (property, value) pairs, which in our notation we refer to as (variable, value) pairs. They are also called (relationship, attribute) pairs in databases. An object o with k (variable, value) pairs is written as (o (v_1 a_1) (v_2 a_2)... (v_k a_k)). ♦

Example: (*Bob* (travels-by boat) (owns bird) (weighs heavy)).

We want to estimate the error in using object *o'* instead of object o. For an object o with k (perhaps hierarchical) variables v_1, v_2,.., v_k and values a_1, a_2, ..., a_k, we say about another object *o'* with same variables v_1...v_k but with values a_1', a_2',... a_k', the following statements:

- *(Definition) o'* is **identical** to *o* if $a_i' = a_i$ for all $1 \le i \le k$. All corresponding values are identical. ♦ If all we know about *o* and *o'* are their values on variables $v_1,...v_k$, and both objects have these values pair wise identical, then we can say that "for all we know," *o* and *o'* are the same.
- *(Def.) o'* is **a substitute** for *o* if conf $(a_i', a_i) = 0$ for all $1 \le i \le k$. ♦ There is no confusion between a value of an attribute of *o'* and the corresponding value for *o*. We can use *o'* instead of the intended *o* with confusion 0.
- *(Def.) o*' is **very similar** to *o* if Σ conf $(a_i', a_i) = 1$. ♦ The confusions add to 1.
- *(Definition) o*' is **similar** to *o* if Σ conf $(a_i', a_i) = 2$. ♦
- *(Definition) o'* is **somewhat similar** to *o* if Σ conf $(a_i', a_i) = 3$. ♦
- *(Definition)* In general, *o'* is **similar$_n$** to *o* if Σ conf $(a_i', a_i) = n$. ♦

These relations are not symmetric.

Example 2 (We use hierarchies H_5, H_6 and H_7). Consider the objects

(*Ann*	(travels-by land-vehicle)	(owns animal)	(weighs weight))
(*Bob*	(travels-by boat)	(owns bird)	(weighs heavy))
(*Ed*	(travels-by water-vehicle)	(owns plant)	(weighs medium-weight))
(*John*	(travels-by car)	(owns cow)	(weighs light)).

Then Ann is similar$_4$ to Bob; Bob is very similar to Ann; Ann is somewhat similar to Ed; Ed is similar$_{3.5}$ to Bob;[11] Bob is similar$_6$ to John, etc. See Table 3.

Table 3. Relations between objects of Example 2. This table gives the relation obtained when using object *o'* (running down the table) instead of object *o* (running across the table)

	Ann	Bob	Ed	John
Ann	identical	similar$_4$	somewhat similar	similar$_5$
Bob	very similar	identical	very similar	similar$_6$
Ed	similar	similar$_{3.5}$	identical	similar$_6$
John	substitute	similar$_4$	similar$_{2.5}$	identical

3.4.2 Accumulated confusion

For compound predicates, a tighter control of the inaccuracy or confusion is possible if we require that the accumulated mismatch does not exceed a threshold ε. This is accomplished by the following definition.

(Def.) ***P* holds for object *o* with accumulated confusion ε**, written *P^ε holds for o*,

- When P^ε is formed by non-hierarchical variables, iff *P* is true for *o*.
- When *pr* is a hierarchical variable and P^ε is of the form (*pr*=c), iff for value *v* of property *pr* in object *o*, $v =_\varepsilon$ c. [That is, if the value *v* can be used instead of c with confusion ε].

[11] conf(water-vehicle, boat) = 1; conf(plant, bird) = 2; conf(medium-weight, heavy) = 0.5; they add to 3.5.

- When P^{ε} is of the form $P1 \quad P2$, iff $P1^{\varepsilon}$ holds for o or $P2^{\varepsilon}$ holds for o.
- When P^{ε} is of the form $P1 \wedge P2$, iff there exist confusions a and b such that $a+b = \varepsilon$ and $P1^{a}$ holds for o and $P2^{b}$ holds for o.
- When P^{ε} is of the form $\neg P1$, iff $P1^{\varepsilon}$ does not hold for o. ♦

Example 3: For Q = (travels-by helicopter)∧(owns cat), we see that Q^0 holds for nobody; Q^1 holds for nobody; Q^2 holds for nobody; Q^3 holds for John; Q^4 holds for {Ann, Bob, John}; Q^5 holds for {Ann, Bob, Ed, John}, as well as Q^6, Q^7...

Figure 6. Query *(address = california)*$_1$ returns customers in California with confusion 1

```
select customer.name, customer.address
from customer
where conf(customer.address,'california')<=1

NAME             ADDRESS
East coast meat  florida
Media Tools      new york
Tom's Hamburgers pasadena
Microsol         silicon valley
Tampa tobacco    tampa
Texas fruits     texas
```

4 Querying a Database with Predicates that are imperfectly fulfilled

(Definition) **Extended SQL.** To query with controlled precision a table T of a database, SQL is extended by these constructs:

- `conf(R,s)`≤ε, a SQL representation for $(R=s)_{\varepsilon}$, is a condition procedure[12] used in a `WHERE` or `HAVING` clause, which is true iff conf(r, s)≤ε. R is the name of a column of T that is a hierarchical variable (a variable or column having hierarchical values), *r* is each of these values, and *s* is the intended or expected qualitative value. ♦ *Example*: `conf(address, mexico)≤ 0` represents in extended SQL the predicate $(\text{address} = \text{mexico})_0$ and will select rows from Fig. 7 whose address is Mexico with confusion 0; that is, all rows where `(address = r)` and conf(r, mexico)≤ 0. It returns rows 2 and 7.
- `conf(R)` is a SQL expression [a shorthand for `conf(R, s)`], used in '`SELECT conf(R)`', or '`GROUP BY conf(R)`' or '`ORDER BY conf(R)`', which returns for each row of table T, conf(R, s). ♦ That is, `conf(R)` returns for table T a list of numbers corresponding to the confusion of the value of property R for each row of T. *Example*: see Fig. 8.

Writing queries in extended SQL. *(Definition)* The algorithm `EXPR` = replace(P) to replace a precision-controlled predicate P by its equivalent extended SQL expression `EXPR` is:

[12] (R=s) is a predicate, for instance (address=USA); adding confusion we have $(R=s)_{\varepsilon}$.

- $(R = s)_\varepsilon$ should be replaced by '`conf(`' R '`,`' s '`)`$\leq$' ε, when R is the name of a column of a table, and s a symbolic value.
- $(P1 \quad P2)_\varepsilon$ should be replaced by '`(`' replace($P1_\varepsilon$) '`OR`' replace ($P2_\varepsilon$) '`)`'.
- $(P1 \wedge P2)_\varepsilon$ should be replaced by '`(`'replace($P1_\varepsilon$) '`AND`' replace ($P2_\varepsilon$)'`)`'.
- $\neg$P should be replaced by '`NOT (`' replace (P) '`)`'.
- $(P1 \wedge P2)^\varepsilon$ should be replaced by '`(`'replace($P1$) '` AND `' replace($P2$) '` AND (conf(`' $P1$ '`) +conf(`' $P2$ '`))`$\leq$' ε '`)`'. ♦

Example: (industrial branch = food)$_0 \wedge$ [(address = pasadena) (address = mexico city)]$_1$ is replaced by `conf (industrial_branch, food)≤ 0 AND (conf(address, pasadena)≤ 1 OR conf (address, mexico city)≤ 1)`. Example: *(address = Mexico City ∧ industrial branch = computer)*^{1} is replaced by `(conf(address, mexico city)≤1 AND conf (industrial_branch, computer)≤1 AND conf(address) + conf (industrial_branch) ≤1)`.

Figure 7. Table of customers

name	industrial_branch	address	discount
Media Tools	computers	new york	0
Garcia Productores	tequila	mexico city	0
Tom's Hamburgers	food	pasadena	0
Microsol	software	silicon valley	0
East coast meat	meat	florida	0
Luigi's italian food	italian food	north america	0
Mole Doña Rosa	mexican food	mexico	0
Texas fruits	fruits	texas	0
Tampa tobacco	cigars	tampa	0
Canada seeds	food	canada	0

Figure 8. Querying, sorting and showing values for *(address = california)*$_1$

```
select customer.name, customer.address,
conf(customer.address)
   from customer
   where conf(customer.address,'california')<=1
   order by conf(customer.address)

   NAME             ADDRESS        CALIFORNIA
   Tom's Hamburgers pasadena       0
   Microsol         silicon valley 0
   Media Tools      new york       1
   Tampa tobacco    tampa          1
   Texas fruits     texas          1
   East coast meat  florida        1
```

Queries: retrieving objects that match P_ε

Example (refer to Fig. 9): (address = usa)$_1$ will return any object whose value of property `address` can be used instead of `usa` with confusion 1. *Example*: Fig. 6 shows customers (of Fig. 7) for which (address = california)$_1$. This returns every

record, except for Mole Doña Rosa customer because the customer's address is somewhere in Mexico and conf(Mexico, California) has a value of 2 (by Fig. 9); except for Garcia Productores because the address is in Mexico City and conf(Mexico City, California) is 2. Except for Luigi's Italian food because the address of the customer is somewhere in North America and conf(North America, California) is 2, and so for Canada seeds, because conf(Canada, California) is 2. For the customers of Fig. 7, we show in Fig. 9 the hierarchy for property `address`, and for `industrial branch` we show in Fig. 10 its percentage hierarchy.

Example: Fig. 8 shows how to sort the answers to (address = california)$_1$ by ascending confusion.

Figure 9. The addresses of customers form a simple hierarchy

```
Property: address;
hierarchy:
world{
   north_america{
      usa{
        california{
            silicon valley,
            pasadena, },
        new york{
            new york city },
        florida{
            miami,
            tampa },
        texas }
      canada,
      mexico{
        mexico city,
        jalisco{
            guadalajara } } } }
```

5 Conclusion

The notions of hierarchy and hierarchical variable are used to measure the *confusion* when a symbolic value is used instead of another (the intended, or correct, value). This creates a natural generalization for predicates and queries. *Confusions* were introduced and developed for hierarchies formed by sets, bags, and lists, and they were extended (§3.1.3) to mixed hierarchies too.

The concepts herein developed have practical applications, since they mimic the manner in which people process qualitative values. Predicates with controlled precision[13] $P_\varepsilon(o)$ (called "P holds for o with confusion ε") and $P^\varepsilon(o)$ (called "P holds for o with accumulated confusion ε") allow us to perform precision-controlled retrieval of hierarchical values. They permit "loose retrieval" (retrieval with defined confusion bounds) of objects that sit in a database. By §3.3, *relations* (columns in a db) could also be "sloppily" specified. Moreover, such db could be an existing "normal" db (where no precision-controlled retrieval was defined), to which one or more defini-

[13] The precision is controlled through the confusion ε.

tions of hierarchies are attached. This in fact provides a procedure (a "kit") to extend *any* (existing) db to another in which imprecise retrievals are possible. Furthermore, this extension can be done without recompiling application programs. Old programs (with no precision retrieval) still work as before, whereas new application programs can exploit the "normal" db as if it were precision-controlled. In fact, a "normal" db now becomes a "precision-controlled" db when the extension (the kit) is applied to it.

Although the paper does not show it, (`update` Texas_fruits**.**address USA) produces no update, since "it is already known (Fig. 7) that Texas_fruits is in USA," while (`update` Texas_fruits**.**address Austin) *will* change the record in the db.

Figure 10. Hierarchy of industrial branch for customers, a percentage hierarchy. It represents the products consumed in a business organization. Each set is followed by its size as a fraction of industrial branch=1. Example: `drinks and cigars` (0.14) split in two halves: `drinks` (0.056) and `cigars` (0.084).

```
Property: industrial branch;
hierarchy:
industrial branch(1){
    computer(.3){
        software(.12),
        hardware(.18)
    },
    human consumption(.7){
        food(.56){
            prepared food(.112){
                mexican food(.0448),
                italian food(.0672)
            },
            meat(.168),
            fruits(.28)
        }
        drinks and cigars(.14){
            drinks(.056){
                whiskey(.0112),
                beer(.028),
                tequila(.0168)
            },
            cigars(.084) } } }
```

References

1. Guzman, A., Dominguez, C., Olivares, J.: Reacting to Unexpected Events and Communicating in spite of Mixed Ontologies. *Lecture Notes in Artificial Intelligence*, Vol. **2313**. Springer-Verlag, Berlin Heidelberg New York (2002) 377-386
2. S. Levachkine, A. Guzman-Arenas, Hierarchies measuring qualitative variables. *Lecture Notes in Computer Science*, Vol. **2945**, Springer-Verlag (2004) 258-270
3. A. Guzman-Arenas, S. Levachkine, Graduated errors in approximate queries using hierarchies and ordered sets. *Lecture Notes in Artificial Intelligence*, Vol. **2972**, Springer-Verlag (2004) 119-128
4. S. Levachkine and A. Guzman-Arenas. Hierarchy as a new data type for qualitative variables. Submitted to *Data and Knowledge Engineering*.
5. J-C Simon. *Patterns and operators. The foundations of data representation.* (McGraw-Hill, 1984

Sememe Analysis and Structural Parsing*

Xiaodong Liu[1] and C. Hoede[2]

[1] School of Information, Xi'an University of Finance & Economics,
Xi'an, Shaanxi 710061, P.R.China
xdliu_npu@hotmail.com, xdliu@xaufe.edu.cn
[2] Department of Applied Mathematics, University of Twente,
P.O.Box 217, 7500AE Enschede, The Netherlands
c.hoede@math.utwente.nl

Abstract. Traditional language parsing of English is mainly based on generative grammar. In structural parsing sentences are mapped on a sentence graph. In this paper, first sememe analysis is introduced in structurally parsing Chinese. Second, we compare the passive sentence in English and Chinese with respect to sentence pattern, semantic relations and other aspects in view of knowledge graph theory. We find that after using sememe analysis in structurally parsing Chinese, we can easily deal with Chinese passive sentences.

1 Grammar and semantics

Traditional parsing of sentences in natural languages is processed by grammar. This method is available for languages like German, Russian and English, *etc.* In these languages, we encounter form change of words, *i.e.*, declension. But for Chinese, a language different from these languages, although grammar is still important in parsing, it is not sufficient to complete the parsing of a sentence.

In grammar sentence components and composition rules are studied. The focus is on determining which component is the subject of the sentence, and which component is the object of the sentence, *etc.* From the viewpoint of grammar, the subject of the sentence expresses the topic that is discussed in the sentence. The other parts are explanation of and comment on the topic.

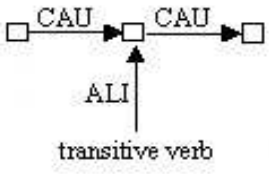
In semantics the meaning of the sentence is studied. In semantics, the question is which component is the agent or actor of the action, and which is the patient or receiver of the action, *etc.*

Liu [4] studied the problem of translation by means of knowledge graphs. Several papers on knowledge graphs have been published in the proceedings of ICCS conferences. For the ontology used see e.g. [2]. The basic idea of the translation is to build the *sentence graph* of a sentence, this is called *structural parsing*, *transform* this graph into one in the target language and then *utter* that sentence graph as a sentence in the target language. This paper focuses on a particular problem in the first step of this three-step procedure, in case we

* Research supported in part by SFAC through grant 01J53079.

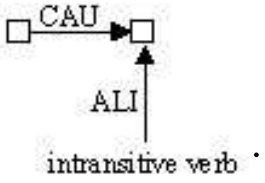
have a Chinese passive sentence. The problem is mainly due to the lack of form change in Chinese.

(1): For transitive verbs, we have the following representation as knowledge graphs,



The first token denotes the agent of the verb denoted by the middle token. The last token denotes the patient or receiver of the same verb.

(2): For intransitive verbs, we have the following knowledge graph,



The first token denotes the agent of the verb denoted by the second token.

(3): For a word like BE, we have the following knowledge graph,

We recall that the knowledge graph ontology consists of the token [ ], eight types of links: EQU, SUB, ALI, DIS, ORD, CAU, PAR and SKO (for equality, subset, alikeness, disparateness, ordering, causality, attribution and informational dependency) and four types of frames. One type is for focusing on a graph structure, similar to expressing "and" by an existential graph, see Sowa [5], and three for describing negation, possibility and necessity. The directed ALI-link between a word and a token is used for typing a token, whereas the directed EQU-link is used for instantiating a token.

Knowledge graphs are an alternative for conceptual graphs and have been introduced independently. The main reason to choose for knowledge graphs is that their ontology is considered to be more basic and that the representational ability seems greater. To illustrate this we consider the simple knowledge graph [ ] -CAU→[ ]-CAU→[ ], where the middle token is of type verb. The structure is

the *meaning* in knowledge graph theory. The left token functions as AGENT and the right token as PATIENT by the very structure of the graph. In conceptual graph theory the words AGENT and PATIENT are used as labels of arcs going out from the middle token, that have to be read as "has AGENT" and "has PATIENT"(Sowa uses "has OBJECT"). This then poses the question what these two labeling words *mean*. This holds even more so for other labels in conceptual graph theory. In knowledge graph theory the subgraphs [ ]-CAU→[ ]←ALI-verb and verb-ALI→[ ]-CAU→[ ] are word graphs and therewith express the meaning of AGENT and PATIENT respectively. Especially with respect to the second step of translation, transformation of the sentence graph, this focus on the structure is of essential importance and therewith justifies our preference for knowledge graphs as language model.

All three representations given before are not syntactic, but semantic knowledge graphs. That is to say, all three are the result of semantic analysis, not the result of syntactic analysis.

For an active sentence, not only for English sentences, but also for Chinese sentences, the mapping of the sentence on these three knowledge graph representations is not difficult, because the word order in the sentences is the same as in the representations. But for passive sentences, the map is very difficult, especially in Chinese.

We have discussed passive sentences in [1]. In that paper, we gave a Chinese passive sentence "Bei3jing1 ya1 kao3 shou2 le, Beijing duck bake well le". The numbers indicate the different intonations used in Chinese. Translated we would say "The Beijing duck was baked well". The structure of the sentence is the same as in "Bei3jing1 ya1 pao3 le, Beijing duck run le". Translated we would say "The Beijing duck ran". From the viewpoint of syntax, the subject is "Bei3jing1 ya1, Beijing duck" in both sentences. The others are predicates. But, as we know, the first sentence is a passive sentence, and the second one is an active sentence. Moreover, there are no marks to show which sentence is passive. Why do we know the differences between these two sentences? Why can we understand these two sentences? The answer is that we understand these two sentences when we address them at the semantic level. We want to show how the construction of the proper sentence graph can be made by means of sememe analysis.

2 Sememe analysis

In the 1940's, the Danish linguist L. Hjelmslev [6] introduced the idea of sememe analysis. In sememe analysis, the word is decomposed into some more basic concepts that themselves are words. In natural language processing, there are some advantages to use sememes to explain word meanings.

Before tying sememe analysis up with the concept of word graph in knowledge graph theory, we should give a definition of the concept of sememe.

***Definition 1*:** A morpheme is a meaningful linguistic unit consisting of a word, such as "man", or a word element, such as "-ed" in "walked", that cannot be divided into smaller meaningful linguistic parts.

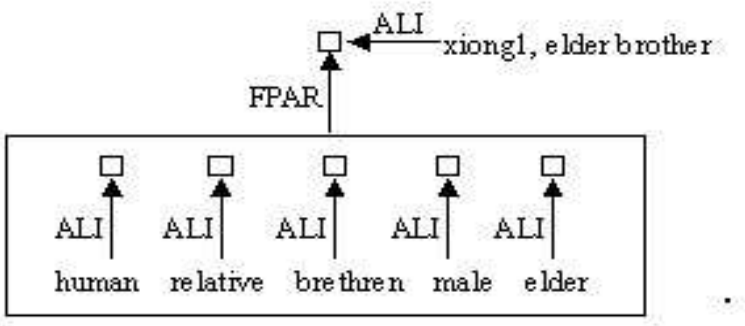
***Definition 2*:** A sememe is the meaning expressed by a morpheme.

In knowledge graph theory any subgraph of the mind graph can be framed and named, and the structure is considered to be the meaning.

Let us now consider the basic idea of description of concepts by sememes. The meaning of the concept is described by giving a set of morphemes. This is what is called *sememe analysis*. Knowledge graph theory can be considered to do just that, but also to impose an extra structure on these sememes.

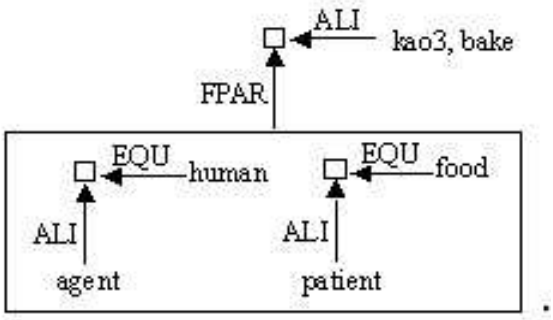
We want to make the usefulness of sememe analysis clear by the following examples.

(1) We know that a word in one language often does not have an exactly corresponding word in another language. Language has national features. If we do not use sememe analysis, or do not use the basic concepts to represent words, we cannot understand the exact meaning of the words. For example, words representing family relations of human beings show a lot of difference. The word "xiong1" in Chinese is an example. "xiong1" means "elder brother", there does not exist an exactly corresponding word in English. Likewise, if we meet the word "brother" in a text in English, for instance in "he is my brother", we cannot translate the sentence into Chinese exactly, because we do not know the exact meaning of the word "brother". The important point now is that if we use some more basic concepts to explain words, we can get the exact meaning of words. As for the word "xiong1, elder brother", we can represent it as the following graph:



Here the frame describes a set of sememes. This set is consider to "determine" the concept xiong1. The FPAR relation expresses that its contents are part of the knowledge graph for xiong1. The word graph for xiong1 should then be constructable from the word graphs of these collected concepts.

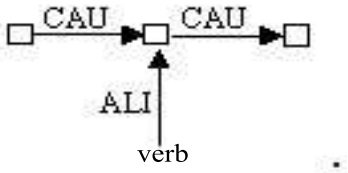
(2) We can use sememe analysis to explain passive sentences. Consider the sentence "Bei3jing1 ya1 kao3 shou2 le, The Beijing duck was baked well". For the word "kao3, bake", we can give the following word graph:



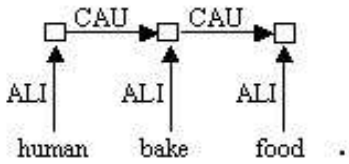
What are added here are not "pure" sememes. In fact, what is expressed is that for baking a human is the agent and food is the patient. This is enough to determine who is baking and what is baked.

Note that in a representation by a conceptual graph the same information would be needed to determine the position of the Beijing duck as patient in that graph.

In the knowledge graph description of a transitive verb we would give, as simplest word graph,



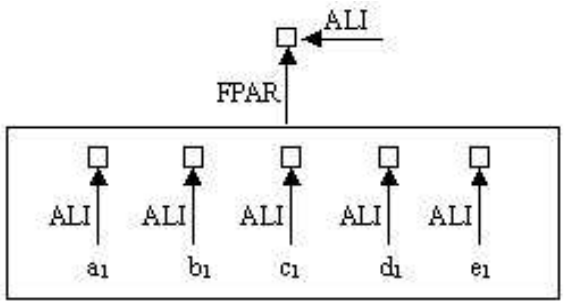
Instantiation of the verb by bake turns the AGENT word graph into the word graph for BAKER. Here the two unlabeled tokens play the role of agent respectively patient. The information introduced by the extending sememes can also be given by the graph



Now the power of the representation by knowledge graph becomes apparent. It covers the sememes as well as the structure between the sememes. Making such

a word graph from the "sememe word graphs" may not be easy. The problem is similar to that of structural parsing in which a sentence graph is made from word graphs.

Like the basic relationships that we have chosen in knowledge graph theory, if we define some symbols to represent basic concepts like sememes, the representations do not depend on the particular language. For instance, if we use a_1 for denoting a human, b_1 for denoting relative, c_1 for denoting brethren, d_1 for denoting male, and e_1 for denoting elder, the sememe word graph of xiong1 or elder brother becomes the following:



For obtaining an adequate sentence graph in the target language we only need a word graph dictionary to explain these symbols and put them together in a word graph structure.

3 Passive sentences and their representations in terms of knowledge graphs

In [1], we have discussed some particular problems in Chinese, and we have analyzed these problems by using knowledge graph methods. In that paper, we found that passive sentences are very difficult to deal with.

Before discussing this in the next section, we discuss the representation of tenses in knowledge graph theory.

In the theory of Reichenbach, tenses are expressed by ordering the time of the speech act with respect to the time of the verb act. Consider the sentences "boy loved dog" and "boy has loved dog". We have to make difference between "loved" and "has loved".

We will use t_s and t_v for indicating the time of speaking and the time of the act described by the verb respectively. $t_{v,b}$ and $t_{v,e}$ will indicate the begin-point and end-point of the time interval during which the act took place. The sentence graphs of "boy loved dog" and "boy has loved dog" are then given respectively by the following sentence graphs shown in Figure 1 on the next page.

The sentence "dog was loved by boy" has "dog" as grammatical subject, but the sentence graph is taken to be the same as that of "boy has loved dog" if we do not want to distinguish the subtle difference in meaning. But as we know, in the sentence "dog was loved by boy" the topic is "dog", and in the sentence "boy

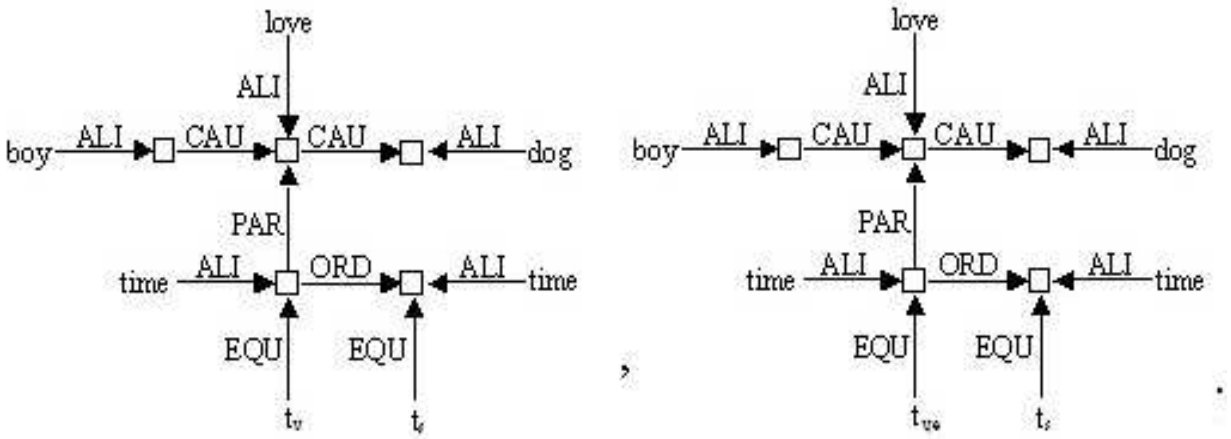
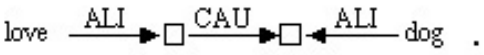


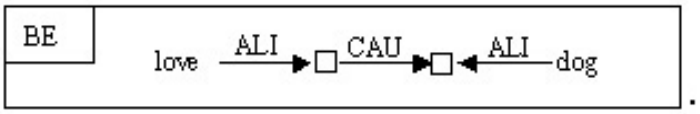
Figure 1

has loved dog", the topic is "boy". If we have to distinguish the subtle difference between these two sentences, we can do the following.

Now, firstly, we consider just the sentence "dog was loved". The basic structure is



It is NOT expressed who loves the dog! The token for the agent is missing. Now put a BE-frame around this structure to get:



We can now attach the t_v-t_s structure again, but not to "love" but to the BE-frame. This then expresses "was loved dog". The sentence graph is shown in Figure 2 on the next page.

If we want to say "by boy", this asks for "boy-ALI→[ ]-CAU→[ ]". The word "by" has word graph "[ ]-CAU→[ ]". Now the free token can be identified with the token for "love".

Concluding, we see that the sentence "dog was loved" and "dog was loved by boy" are expressed by the following sentence graphs shown in Figure 3 on the next page. Both sentence graphs stress the passive sentence structure by the BE-frame.

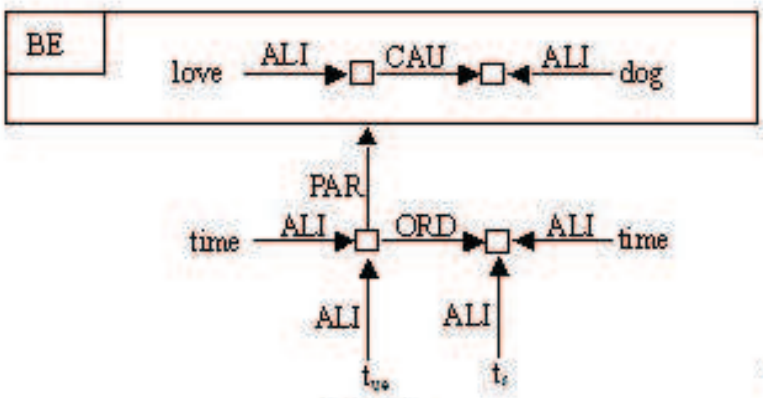


Figure 2

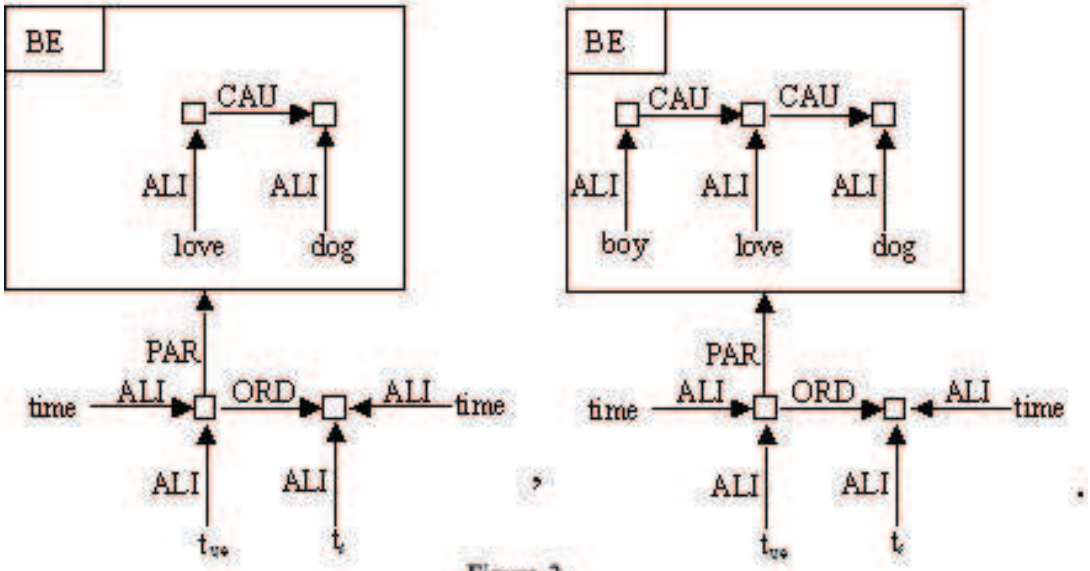


Figure 3

4 The passive sentence in English and in Chinese

The passive sentence both in English and in Chinese, can be a *"special" passive sentence* and a *"meaning" passive sentence*. There exist some special marks in the special passive sentences, but the meaning passive sentence does not have such a mark.

4.1 The special passive sentences in English

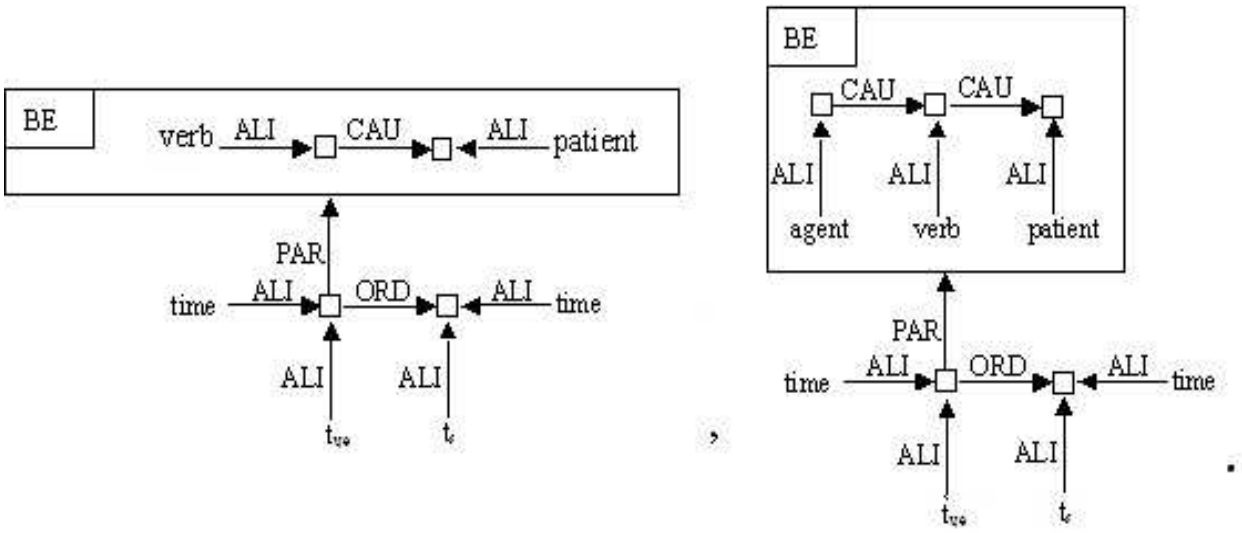
The special passive sentence in English is represented by the passive mode of the transitive verb. The basic form of the special passive sentence in English has the form

subject be past participle (by agent of the verb)

Here the auxiliary verb "be", the past participle, and "by" are called passive marks. The word "by" is used to indicate the agent of the verb (if one does not

clearly show the agent of the verb, the "by" phrase is omitted). The different sentence tense is shown through the change of the auxiliary verb "be". The special passive sentence in English stresses that the action has taken place.

This form of sentences is not difficult to express by means of a knowledge graph, because there are the obvious marks for passive use of the verb in these sentences. Although the same basic meaning can be expressed by an active sentence and a corresponding passive sentence, the emphasis is different. As we discussed above, in order to distinguish the subtle difference we use the following sentence graphs to express passive sentences respectively:



4.2 The special passive sentence in Chinese

There are some passive marks used in special passive sentences in Chinese, such as the words "bei4, you2, rang4, shou4, ai2, gei3, and zao1sho4", *etc.* We will discuss some of them in this paper. Unlike the special passive sentences in English, some of these words are not only passive marks, but also still predicate verbs in some sentences. So, the special passive sentences can be divided into two groups: Group A and Group B. In Group A, the marks used in them are only passive marks. In Group B, the marks used in them are still predicate verbs.

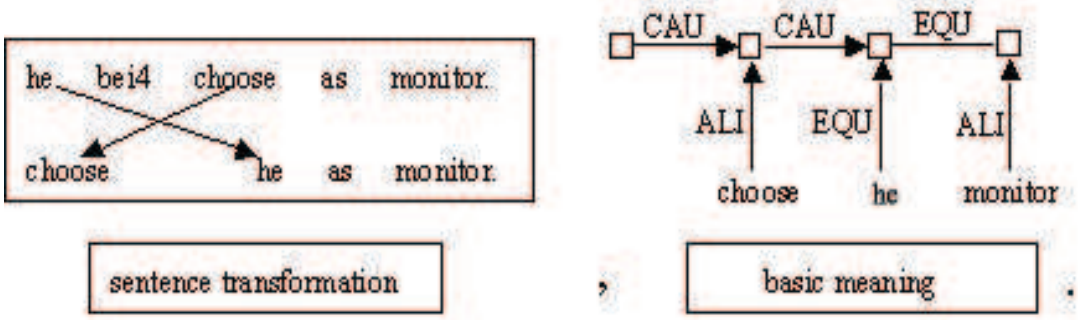
Like in the case of the special passive sentence in English, these sentences are not difficult to structurally parse by means of knowledge graphs, because there are obvious marks of the passive tense in these sentences.

"Bei4" is a very important passive mark in Chinese, belonging to Group A. It functions as a preposition or as an auxiliary word. The passive sentence with the mark "bei4" need not have an agent of the verb.

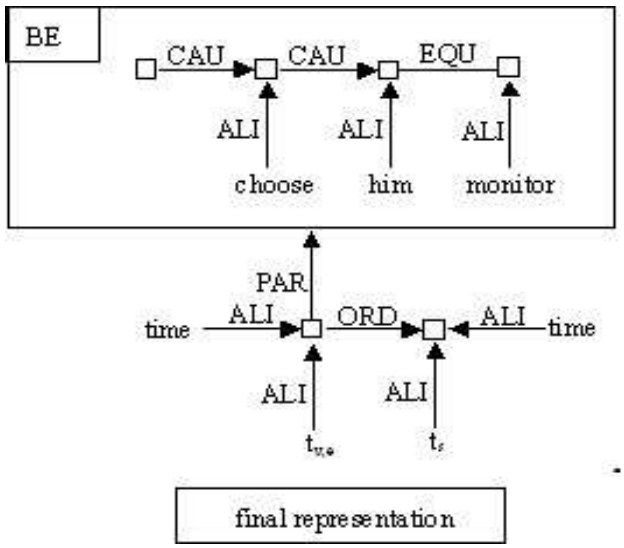
Example 1 "ta1 bei4 xuan3wei2 ban1zhang3, he bei4 choose as monitor" is an example of a passive sentence with the passive mark "bei4". The meaning

is "he had been chosen as monitor." "Bei4" in this sentence is only the passive mark. It is only used to form the passive sentence.

How to get the (semantic) sentence graph for this sentence? From the semantic point of view, "ta1, he" is the patient or receiver of the verb "xuan3, choose". So, we get the following representation by means of a knowledge graph after the transformation:



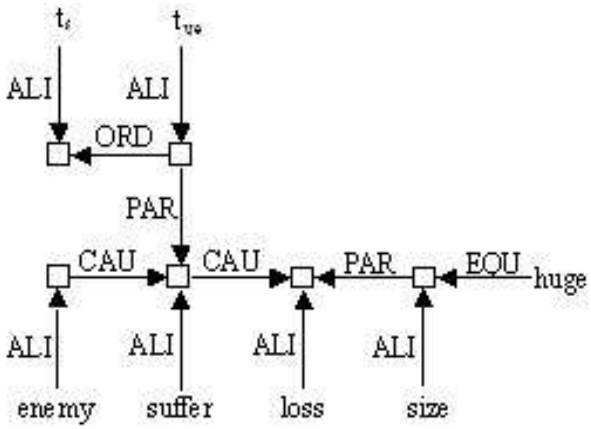
But, how to express the word "bei4"? How to express the passive tense? Because the passive sentence is expressed by a "BE"-frame, as we discussed above, we introduce the "BE"-frame and express "bei4" or the passive tense by the attribution (PAR) of the time relationship between $t_{v,e}$ and t_s. So, we get the following expression,



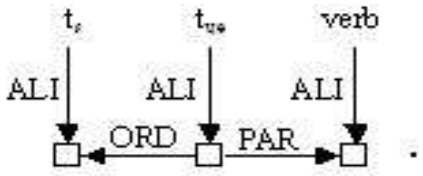
As a conclusion, in order to get the representation of the (semantic) sentence graph, we have to carry out two steps. In the first step, we use the transformation to get its basic meaning. In the second step, we express the word "bei4", expressing the passive nature of the sentence.

In Group B, the marks forming Chinese special passive sentences are still predicate verbs, the passive meaning of the sentences is shown by the verbs themselves.

Example 2 "di2ren zao1shou4 le ju4da4 de sun3shi1, enemy suffer le huge de loss" is a Chinese sentence that exhibits passive *tense* in a special way. Its meaning is "The enemy has suffered huge loss." Here, "zao1shou4, suffer" is not only marking the passive tense, but is the real verb in the sentence. Because there is this passive mark in the sentence, its expression by a sentence graph is very simple. Note that "le" also indicates the past tense. The sentence graph is:



Note that the word graph for "le" can be taken to be



Normally, when the verbs express actions like delivering or transmitting, there exist some special passive sentences. In these sentences, the patient of the verb can be the subject. But there are no such forms of passive sentences in Chinese.

4.3 The meaning passive sentence in English

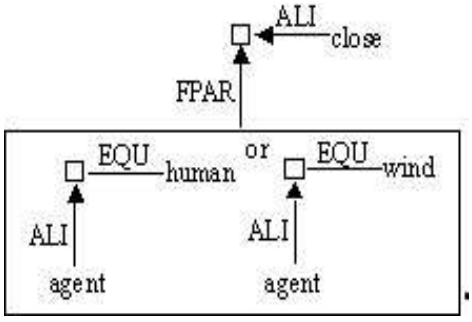
The meaning passive sentence in English is the sentence where the form is active, but the meaning is passive. Mostly, these sentences are used to express the relevant conditions that occur around or accompany the action that occurs. The

meaning passive sentences in English can be divided into the following two small groups approximately:

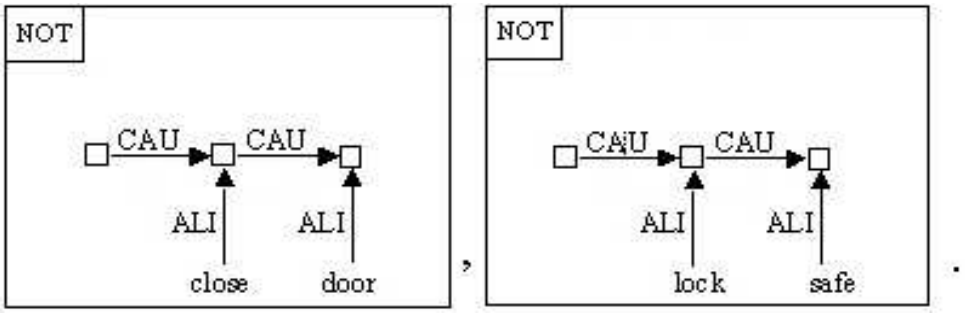
(1) The meaning passive sentence in which the verb is in active form

In English, some verbs can express passive meaning although the form is active, such as in the following examples.

Example 3 The sentences "The door won't close." and "The safe doesn't lock." are two English meaning passive sentences. The form is in active, but the meaning is passive. How to structurally parse these sentences? In these sentences, the form of the sentences is active, and there are no passive marks. Why do we say these sentences are passive sentences? The only way to check is according to the meaning of the sentences. We know "the door" could not close itself, and the safe could not lock itself. The door (safe) closed (locked) because something (someone) closes (locks) it. So, as we discussed in Section 2, sememe analysis, we should do some more work on word graphs. If we add extensions to the verbs "close" and "lock", the problems become easy. For example, if we add to the word graph some properties of the word "close", the parsing is easy.



After that, the expressions of the sentences by sentence graphs are simple:

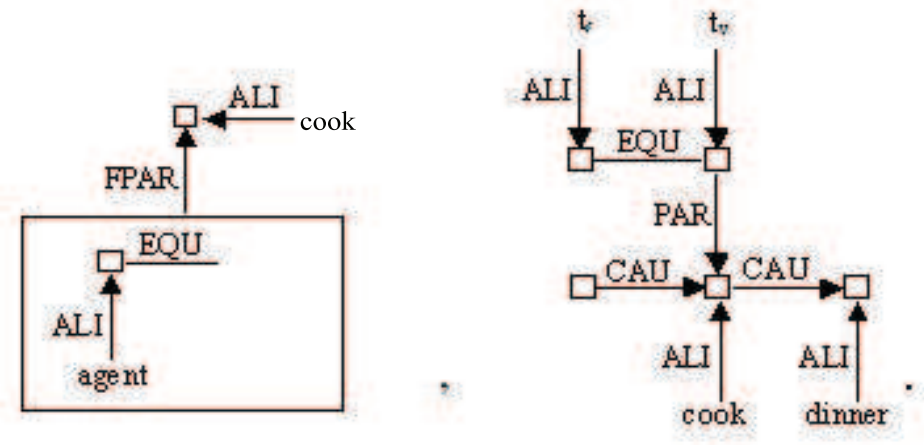


(2) The meaning passive sentence in which the progressive tense of the verb is used

There are a few verbs taking the active form to express a passive meaning.

Example 4 "The dinner is cooking" is a sentence with the progressive tense of the verb to express the passive tense in active form. This is a meaning passive

sentence because the dinner cannot cook itself. The dinner has to be cooked by someone. So, if we add this property to the verb "cook", the parsing is easy. The result is:



A sentence of this kind usually does not use some "person" to be the subject, instead the thing that is the object of the verb is taken as the subject.

4.4 The meaning passive sentence in Chinese

Although there are many marks to express passive meanings in Chinese sentences, there are still, sometimes, other approaches to express passive meaning in a more natural way without the passive marks. Such as in:

Example 5 "yi1fu3 xi3 jing4 le, clothing wash clean le" is a Chinese passive sentence in active form without marks for the passive sentence. Its meaning is "the clothing has been washed clean."

How to analyze these sentences? Again we have to use the meaning of the sentence as in English. The "clothing" will not "wash" itself. So, we have to extend word graphs. For instance, if we add a property to the verb "wash", the structural parsing is easy. The graph of the sentence is still simple, and shown in Figure 4 on the next page.

We want to conclude with a striking example of sentence formation in Chinese.

Any activity takes place in a certain space. Therefore pointing out the place where the act occurs can often give information about the activity.

Example 6 "zhe4 ji1qi1 shi4 wo3mei2 chang3 zhi4zao4 de, this machine be our factory make de" is a very special sentence in Chinese. Translated we would say, "this machine was made by (in) our factory". This is still a passive sentence. The topic is "ji1qi1, machine", but the stress is "wo3men1 chang3, our factory". The sentence is a very special sentence in Chinese, because this sentence has pointed out the place of production of the machine, and has also pointed out

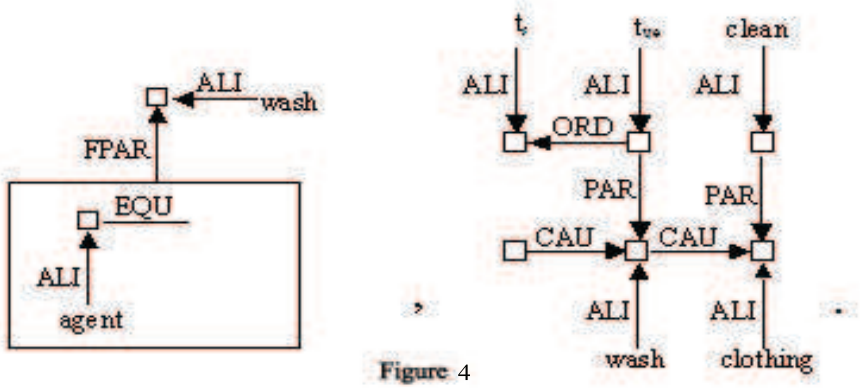


Figure 4

the machine producer. The implications "in our factory" and "by our factory" are inferred from the extended meaning of "zhi4zao4, make". This extension is not present in English, probably because in English the prepositions "by" and "in" are at hand.

5 Discussion

We have seen that, due to the fact that in Chinese semantic aspects play a dominant role in interpreting a sentence, and hence in mapping a sentence on a sentence graph, extensions of word graphs are needed. The addition of sememes is a useful way of achieving these extensions. Their corresponding knowledge graphs are easily incorporated in the sentence graph structure and may help in the interpretation of the sentence. For more discussion on the application of knowledge graph theory to Chinese language, we refer to Liu [4]. For an elaborate account of structural parsing we refer to Zhang [7].

References

1. Hoede,C., X. Li, X. Liu, L. Zhang: *Knowledge Graph Analysis of some Particular Problems in the Semantics of Chinese*, Memorandum No.1516, Department of Applied Mathematics, University of Twente, Enschede, The Netherlands, (2000).
2. Hoede,C., X. Liu: *Word Graphs: The Second Set*, in Conceptual Structures: Theory, Tools and Applications, Proceedings of the 6^{th} International Conference on Conceptual Structure, Montpellier, ICCS'98 (M.-L. Mugnier, M. Chein, eds.) Springer Lecture Notes in Artificial Intelligence 1453, 375–389, ISBN 3-540-64791-0 (1998).
3. Hoede, C., L. Zhang: *Structural Parsing*, in Conceptual Structures: Extracting and Representing Semantics, Aux. Proc. of the 9^{th} International Conference on Conceptual Structures (Ed. G. W. Mineau), CA, USA, 75–88, (2001).
4. Liu, X.: *The Chemistry of Chinese Language*, Ph.D. thesis, University of Twente, Enschede, The Netherlands, ISBN 90-3651834-2 (2002).
5. Sowa, J. F.: *Conceptual Structures: Information Processing in Mind and Machine*, Addison-Wesley, Reading Massachusetts, ISBN 0-201-14472-7 , (1984).
6. Jia YanDe: *Chinese semantics*, Peking University Press, 1999, (in Chinese).
7. Zhang, L.: *Knowledge Graph Theory and Structural Parsing*, Ph.D. thesis, University of Twente, Enschede, The Netherlands, ISBN 90-3651835-0 (2002).

An Axiomatic System for Peirce's Alpha Graphs

Xin-Wen Liu

Institute of Philosophy, Chinese Academy of Social Sciences
No. 5, Jianguomennei Avenue, 100732, Beijing, PRC
liuxw-zxs@cass.org.cn

Abstract. This paper presents a Hilbert-style system for Alpha graphs, the first part of Existential Graphs. A set of generalized Sheffer-strokes are the only connectives in the "symbol-based" formal system for Alpha graphs, and the most important advantage of the system is that both the decision procedure and the completeness's proof via countermodel are immediate.

1 Introduction

The present paper is an attempt to amalgamate two systems of logic that Peirce developed over his long career. In the paper titled "A Boolean Algebra with One Constant" of 1880, Peirce showed how all the elective functions of Boole could be expressed by use of a single primitive sign with the meaning of "neither ... nor ...". This is the first discovery of the truth-functional completeness of the 2-ary connective Sheffer's stroke function (henceforth, sheffer-stroke), "|" in notation. This fact is rediscovered by H.M. Sheffer in 1913 in his "A Set of Five Independent Postulates for Boolean Algebras, with application to logical constants". Then, the most important development is the presentation of the calculus in a strictly axiomatized form. Based on Peirce and Sheffer's taking $A|B$ as undefined, Nicod showed in his 1917 article "A Reduction in the Number of the Primitive Propositions of Logic" that the whole calculus could be based on the single axiom

$$[A|(B|C)]|\ ([D|(D|D)]|\ \{(E|B)|[(A|E)|(A|E)]\})$$

with

$$\alpha \qquad \alpha|(\beta|\gamma) \qquad / \therefore \gamma$$

as a rule of inference in place of the traditional *modus ponens*. But it can scarcely be said that the reduction achieved by Nicod is a simplification which makes the theory easier to grasp. Peirce himself also says: "Of course, it is not maintained that this notation is convenient." ([Pe33] 4.20)

The sheffer-stroke has two dual interpretations "neither ... nor ..." and "either not ... or not ...". It is worth noticing that introducing the sheffer-stroke into Existential Graphs should avoid this kind of troublesomeness resulted from

Peirce's "notation in which the number of signs should be reduced to a minimum" ([Pe33] 4.12) and simultaneously save the elegance because of the least primitive connective. Throughout his scientific life Peirce explored, with seemingly endless creativity and stamina, one notational device after another. Peirce's chapters on Existential Graphs in volume 4 of his *Collected Papers* contain a wealth of ideas. We know that sheffer-stroke "either not ... or not ..." could be defined by two classical connectives negation and conjunction as "it is not the case that A and B". Interpreted as negation and conjunction, the only two primitive operations "Cut" and "Juxtaposition" underlie Existential Graphs. From this point of view, then, introducing the sheffer-stroke "either not ... or not ..." into Existential Graphs is very much apropos although the most outstanding characteristic of Existential Graphs is the "Iconicity". Of course according to Peirce a graph is in the main an "Icon" of the forms of relations in the constitution of its "Object," but nevertheless, it will ordinarily have symbolic features:

> Now since a diagram, though it will ordinarily have Symbolic Features, as well as features approaching the nature of Indices, is nevertheless in the main an Icon of the forms of relations in the constitution of its Object, the appropriateness of it for the representation of necessary inference is easily seen. ([Pe33] 4.531).

In accordance with this idea, certainly Existential Graphs can be regarded as an iconic system as well as a symbolic system. Based on the sheffer-stroke interpreted as "either not ... or not ..." and two rules of inference in place of the original ones, this paper develops an axiomatic system for the first part of the Existential Graphs, Alpha. The most outstanding advantage of this system is that the proof of every provable graph is effective. In the original occasion it is allowable to draw n graphs on the same area simultaneously, so in the system to be developed we adopt a generalized version instead of 2-ary sheffer-stroke, which occurs implicitly like many situations of Peirce himself.

We proceed as follows: Section 2 is a short introduction to the Peirce's Alpha Graphs, the readers who are familiar with Existential Graphs should jump over this section. An axiomatic system for the Alpha graphs defined by this paper and an informal definition of a deduction-tree of this axiomatic system are given in section 3, moreover, this section describes a backward chaining procedure along with the inference rules for every graph. Based on the semantics presented in section 4, section 5 establishes the soundness and completeness theorems for our formal system. In the concluding section we present some expectations for further research and defaults of this approach.

2 Alpha Graphs

Peirce sets up Existential Graphs with the intention of providing a logical analysis of mathematical reasoning. We start by considering the grammar of the Alpha system. In fact, the language of the Alpha system can be described as a

propositional language augmented with a propositional constant and based on a small complete set of connectives, namely negation and conjunction.

In the Alpha part there are just three primitive types of symbol:

1. A piece of paper or a blackboard upon which it is practicable to scribe the graphs, termed the *Sheet of Assertion*. Every part of the surface is called the *blank*.
2. Propositional signs (e.g., A, B, ...), probably any symbols, words or natural language sentences.
3. A self-returning finely drawn line known as a *Cut* or *Sep*.

Peirce occasionally employs a linear (or, bracket) notation for his graphs, a notation that is convenient for typesetting and space considerations though not as visually perspicuous as his two-dimensional notation. [1] In the situation without confusion, it is convenient to adopt this linear notation (more precisely, square bracket notation). Semantically, the sheet of assertion represents the universe of discourse. Writing the propositional signs on the sheet of assertion amounts to asserting their truth. For example, as Peirce states ([Pe33] 4.433), by writing

The pulp of some orange is red.

we assert that in our domain of discourse it is true that the pulp of some orange is red. Negation and conjunction are the principal logical connectives of the Alpha system. The sign for negation is the cut. By encircling the above assertion we get

[*The pulp of some orange is red*].

which asserts that it is false that the pulp of some orange is red. The conjunction of two or more assertions is obtained by juxtaposing the assertions together on the sheet. For example,

The pulp of some orange is red.
To express oneself naturally is the last perfection of a writer's art.

asserts that the pulp of some orange is red and to express oneself naturally is the last perfection of a writer's art. The other propositional connectives can now be defined in terms of cut and conjunction. Truth is represented by the empty sheet of assertion and falsity by an empty cut. In the Alpha system Implication is presented as $[A[B]]$.

One of the advantages of graphic notation is the ease of reading the graphs in many different ways, for example, the disjunction "A or B" may be read in at least five more ways. [2] In the same time, on the other hand, the syntactic history of Peirce's graphs is fundamentally different from the way a formula is

[1] See, e.g., [Pe33], 4.378–389.
[2] According to a manuscript of Shin's.

composed out of the basic vocabulary of its system, that is to say, Peirce's graphs are pressed for the property of unique readability.

The rules are the following:

1. *Deletion* and *Insertion*: "Within an even finite number (including none) of seps, any graph may be erased; within an odd number [of seps] any graph may be inserted." ([Pe33] 4.492)
2. *Copying Rule*: "Any graph may be iterated within the same or additional seps, or if iterated, a replica may be erased, if the erasure leaves another outside the same or additional seps." ([Pe33] 4.492)
3. *Double Negation Rule*: "Anything can have double enclosures added or taken away, provided there be nothing within one enclosure but outside the other." ([Pe33] 4.379)

3 Axiomatics

In the system to be developed for Alpha the primitive symbols include sentence letters $p_0, p_1, \ldots,$ (p, q, r, s for metavariables) and the cut [ ] (or ( ), { } if necessary) in brackets-notation. And the number of the primitive connectives is infinite and each is a *generalized n-ary sheffer-stroke* (henceforth *nand*) which occurs implicitly, for $n \geq 0$.

Now define a set, *Ag*, of our *Alpha graphs* in the following way:

Definition 1. *1. Every sentence letter is a graph.*
2. For $n \geq 0$, if $\alpha_0, \ldots, \alpha_{n-1}$ are all graphs, then the single cut of the juxtaposition $[\alpha_0 \ldots \alpha_{n-1}]$ of the n graphs $\alpha_0, \ldots, \alpha_{n-1}$ is a graph. [3]
3. Nothing else is a graph.

In the definition of "graph" there is no restriction on the order of the n graphs $\alpha_0, \ldots, \alpha_{n-1}$. This applies also to linear transcriptions of graphs. In other words, the order of the n graphs $\alpha_0, \ldots, \alpha_{n-1}$ in graph $[\alpha_0 \ldots \alpha_{n-1}]$ has no logical significance.

By the above definition, if $n = 0$, then the empty cut (called *Enclosure*), [ ] in bracket notation, is a graph; and if $n = 1$ and α_i is a sentence letter say p, then a single cut of α_i is a graph, i.e., $[p]$; similarly, $[[\alpha_i]]$ is a graph. Additionally, p, the scrolls $[p[q]]$, $[[p][q]]$, the *double cut* (namely the empty scroll) [[ ]], etc., are all graphs. Evidently, the empty space and the juxtaposition of the n graphs $\alpha_0, \ldots, \alpha_{n-1}$ such as pqs, $p[q]$, are all "well-formed" graphs by Peirce's definition but no more by ours. So, it is not the case that all of the original Alpha graphs are captured by our definition. But the graphs at present do have the property

[3] Shin develops a definition for Peircean Alpha graphs. In her definition the implicit sheffer-stroke seems ready to come out at one's call. Our definition could be regarded as a refined version of hers. See [Sh02], p.65.

of unique readability because the two operations Cut and Juxtaposition in fact have been handled unitarily as one connective though.

Definition 2. *The set of subgraphs of a graph α is the set Sub(α) such that:*

1. *Sub(*p*)=*{p}*, for every sentence letter* p *;*
2. *Sub($[\alpha_0 \dots \alpha_{n-1}]$)=$\{[\alpha_0 \dots \alpha_{n-1}]\} \cup \bigcup_{i=0}^{n-1} Sub(\alpha_i)$*

Sentence letters have no *immediate subgraphs*, and the collection of *immediate subgraphs* of $[\alpha_0 \dots \alpha_{n-1}]$, *I-Sub*($[\alpha_0 \dots \alpha_{n-1}]$) in notation, is the collection of graphs $\alpha_0, \dots, \alpha_{n-1}$; similarly, for a fixed set Γ of graphs and $\alpha_i \in \Gamma$, *I-Sub*(Γ)=$\bigcup_{i=0}^{n-1}$ *I-Sub*(α_i).

By the definition of graphs every well-formed graph is a sentence letter or a single cut of a graph α_j, then every immediate subgraph of the graph $[\alpha_0 \dots \alpha_{n-1}]$, i.e., $\alpha_0, \dots, \alpha_{n-1}$, is a sentence letter or a single cut of a graph. For example, let α be

$$[p\ q\ (r[\ ]\ ][s])\ [\ ]\ ]$$

Then *I-Sub*(α) is exactly the set

$$\{p,\ q,\ (r[\ ]\ ][s]),\ [\ ]\}$$

which includes four members.

Let α, β be graphs and $\beta \in$ *I-Sub*(α). Then we have the following notions:

Definition 3. *1. α is a simple graph if and only if α=[ ] or α=p or α=[p] for a propositional variable p;* [4]
2. β is an atomic nand if each immediate subgraph α in I-Sub(α) is a simple graph.

That is to say, a graph is an atomic *nand* if and only if its immediate subgraphs are all simple graphs.

Now we state the axioms and diagrammatic transformation rules for our graphs. There are two axiom schemas and two rules of inference. The two axiomatic schemas are:

Definition 4 (Axiom).

1. *Every atomic nand whose set of immediate subgraphs contains at least one empty cut is an axiom.*

2. *Every atomic nand whose set of immediate subgraphs contains some sentence letter and its negation is an axiom.*

[4] This definition can be viewed as a contraction of the definition 4.3 of Shin's. See [Sh02], p.65.

Rules are schemas too. Let $\alpha_0, \ldots, \alpha_{m-1}, \beta_0, \ldots, \beta_{n-1}$ be graphs for m, $n \geq 0$, α and β denote these two sequences of graphs respectively. The two diagrammatic transformation rules, called *rule of double cut insertion* (rule 1) and *rule of unification of n graphs* (rule 2) respectively, are as follows:

Definition 5 (Rules).

1. *From [αβ] infer [α[[β]]];*

2. *From n graphs [α [β_0]], ..., [α[β_{n-1}]] infer [α[$\beta_0, \ldots, \beta_{n-1}$]].*

Note that the axioms are exactly the tautologous atomic nands, and that the rules are equivalence-transformations. The definition of a graph being provable from a set of graphs is then defined as follows: Let $\Gamma \cup \{\alpha\}$ be a set of graphs, then α is a *provable* from Γ (written $\Gamma \vdash \alpha$) if and only if there is a finite nonempty sequence of graphs $\langle \alpha_0, \ldots, \alpha_{n-1}, \alpha \rangle$ such that each member of this sequence is either a member of Γ, an axiom, or follows from previous graphs by one of the two diagrammatic transformation rules. The sequence $\langle \alpha_0, \ldots, \alpha_{n-1}, \alpha \rangle$ is called a *deduction* of graph α and the number of $\langle \alpha_0, \ldots, \alpha_{n-1}, \alpha \rangle$ is called the length of a deduction. If Γ is empty (i.e., $\vdash \alpha$), then α is a *theorem*.

Generally speaking, Hilbert-style systems (i.e. axiomatic systems) can be useful as formal representations of what is provable, but the actual finding of proofs in Hilbert-style systems is next to impossible. This is not the case in the system just stated. In this system, deductions will be presented as *trees*, called *deduction-trees*; the *nodes* will be labeled with graphs; the labels at the immediate successors of a node v are the premises of a rule application, the label at v the conclusion. At the *root* of the tree we find the conclusion of the whole deduction.

In accordance with the from-bottom-to-top direction in the two diagrammatic transformation rules, every deduction-tree grows upwards from its root, i.e., the graph needed to be proved. The procedure is as follows:

1. If a label at the node v is a graph which set of the immediate subgraphs has a graph with one double cut of a graph (or many graphs) as its member, then, in accordance with the rule of double cut insertion, we erase the double cut and take the result as the immediate predecessor of v.
2. If a label at the node v is a graph that the set of its immediate subgraphs has a graph with one cut of many graphs as its member, then, in accordance with the rule of unification of n graphs, we take the cut of many graphs apart and take the result as the immediate successors of v.

In general, the step (1) is considered prior to the step (2). Repeatedly use these two steps the deduction-tree grows from bottom to top gradually. Whenever all the labels at the tops of every branch are atomic *nand*s, the growth of the tree ends. Finally, write down on the paper all of the graphs from top to bottom line by line (and delete the repetitions, if any) which complete the proof. By constructing a deduction-tree with these two steps, the provability of a graph is

reduced to the provability of a finite collection of atomic *nand*s, and when all of these atomic *nand*s are axioms then the original graph is provable.

To illustrate the use of the two diagrammatic transformation rules,let's prove a complex theorem. Consider Leibniz's *PraeclarumTheorema*:

$$((p \to r) \wedge (q \to s)) \to ((p \wedge q) \to (r \wedge s))$$

This theorem is

[[p[r]][q[s]][[pq[rs]]]]

in bracket-notation and

I-Sub([[p[r]][q[s]][[pq[rs]]]])={[p[r]], [q[s]], [[pq[rs]]]}.

From the two graphs
(1) [rspq[r]]
(2) [rspq[s]]
one can infer
(3) [rspq[rs]]
according to the second rule, from which in turn can infer
(4) [r[[s]]pq[rs]].
according to the first rule. Similarly, from the two graphs
(5) [r[q]pq[r]]
(6) [r[q]pq[s]]
one can infer
(7) [r[q]pq[rs]]
and from
(8) [[p]spq[r]]
(9) [[p]spq[s]]
infer
(10) [[p]spq[rs]],
from which in turn infer
(11) [[p][[s]]pq[rs]],
and from
(12) [[p][q]pq[r]]
(13) [[p][q]pq[s]]
infer
(14) [[p][q]pq[rs]].
Now, from (4) and (7) one can infer
(15) [r[q[s]]pq[rs]],
from which in turn infer
(16) [[[r]][q[s]]pq[rs]],
and from (11) and (14) one can infer
(17) [[p][q[s]]pq[rs]].
Now, from (16) and (17) one can infer
(18) [[p[r]][q[s]]pq[rs]],

from which in turn infer the root, i.e., Leibniz's *PraeclarumTheorema*
(19) [[p[r]][q[s]][[pq[rs]]]]
So, the sequence $\langle(19),(18),\ldots,(2),(1)\rangle$, which reverse is the deduction of (19), is the deduction-tree of (19). Because all the atomic *nand*s (1), (2), (5), (6), (8), (9), (12) and (13) are axioms, (19) is a theorem.

4 Semantics

Having stated the axioms and rules, we can now turn to the task of showing that they are complete. To formulate the completeness issue precisely it is necessary to provide a semantics for the graphs. The semantics for the system is similar to that for propositional logic. Let Ag be a set of graphs, then we have

Definition 6 (truth assignment). *A truth assignment is a function* * *from* Ag *onto* $\{1, 0\}$ *such that* $([\alpha_0 \ldots \alpha_{n-1}])^* = 1$ *if and only if* $(\alpha_i)^* = 0$ *for some* $i<n$.

By this definition, the graph $[\alpha]$ is called the *negation* of the graph α, [5] and now the 2-ary sheffer stroke | is expressed by $[pq]$ in bracket-notation. Moreover, the definitions of all the seven signs in a Boolean Algebra are restated in bracket-notation as follows: [6]

1. $\neg p =_{df} [p]$
2. $p \wedge q =_{df} [[pq]]$
3. $p \vee q =_{df} [[p][q]]$
4. $p \rightarrow q =_{df} [p[q]]$
5. $1 =_{df} [[\ ]]$
6. $0 =_{df} [\ ]$
7. $p \leftrightarrow q =_{df} [[pq]\ [[p][q]]]$

Let * be a truth assignment function, Ag a set of graphs and α a graph. We say that α is *satisfiable* provided that $(\alpha)^*=1$; in this case * *satisfies* α, otherwise $(\alpha)^*=0$. Similarly, we say that Γ is *satisfiable* provided that for each $\alpha \in \Gamma$, $(\alpha)^*=1$; in this case * *satisfies* Γ (written $(\Gamma)^*=1$; otherwise $(\Gamma)^*=0$). We say that α is a *tautology* (written $\models \alpha$) provided that for each truth assignment function *, $(\alpha)^*=1$. Finally, if $\Gamma \cup \{\alpha\}$ is a set of graphs we say that α is a logical consequence of Γ (written $\Gamma \models \alpha$) if every function * that satisfies Γ satisfies α.[7]

[5] This definition of negation differs from the others. Generally, negation is defined as $\neg A =_{df} A|A$ including Peirce's rubbing out the |. See the note of [Pe33] 4.20.

[6] The symbol "$=_{df}$" means a definition.

[7] This semantics is similar to which is provided in [Ha95] in the case of Peirce diagrams.

5 Soundness and Completeness

The soundness and completeness theorems together assert the equivalence of provability ($\vdash$) with entailment ($\models$).

We state three propositions here before our proof of the soundness and completeness theorems:

Lemma 1. *An atomic nand is a tautology if and only if it is an axiom.*

Proof. Without loss of generality, let α be a sequence of simple graphs, then according to our semantics defined above it is easy to prove that $[[\ ]\alpha]$ is a tautology. Similarity for the second axiom. □

Lemma 2. *The two rules preserve tautologicality.*

Proof. Leave it to the readers. □

Lemma 3. *Let * be a truth assignment. If * doesn't satisfy the premise(s) of one rule then * doesn't satisfy the conclusion of it.*

Proof. Let * be a truth assignment, α and β denote the sequence of graphs $\alpha_0 \dots \alpha_{m-1}$ and $\beta_0 \dots \beta_{n-1}$ respectively, and Γ is a set of graphs:

$$\{[\alpha[\beta_0]], \dots, [\alpha[\beta_{n-1}]]\}$$

By hypothesis we have $([\alpha\beta])^*=0$ (the first rule) and $(\Gamma)^*=0$ (the second rule).
First rule:

$$\begin{aligned}([\alpha\beta])^* = 0 &\Longleftrightarrow \text{for all } j < m, i < n,\ (\alpha_j)^* = (\beta_i)^* = 1 \\ &\Longleftrightarrow ([\alpha[[\beta]]])^* = 0\end{aligned}$$

that is to say, * doesn't satisfy $[\alpha[[\beta]]]$.
Second rule:

$$\begin{aligned}(\Gamma)^* = 0 &\Longleftrightarrow \text{for some } i < n,\ ([\alpha[\beta_i]])^* = 0 \\ &\Longleftrightarrow \text{for some } i < n,\ (\alpha)^* = 1,\ (\beta_i)^* = 0 \\ &\Longleftrightarrow (\alpha)^* = 1,\ ([\beta_1 \dots \beta_n])^* = 1 \\ &\Longleftrightarrow ([\alpha[\beta_1 \dots \beta_n]])^* = 0\end{aligned}$$

that is to say, * doesn't satisfy $[\alpha[\beta]]$. □

Let α be a graph, now from the first two propositions by induction on the length of deductions we have:

Theorem 1 (Soundness). *If $\vdash \alpha$ then $\models \alpha$.*

Now we turn to the task of showing that the rules are complete.

Theorem 2 (Completeness). *If $\models \alpha$ then $\vdash \alpha$.*

Proof. We show the contrapositive. Assume that $\vdash \alpha$ does not hold. We construct a tree for α, thus each branch must be ended with a set of atomic *nands*, in which there must be at least one member that is not an axiom. Without loss of generality, let $\langle \alpha_0, \ldots, \alpha_{n-1}, \alpha \rangle$ be a branch of α, Γ a set of graphs at the top node α_0, $\{p_0, \ldots, p_{n-1}\}$ a complete set of letters occurring in Γ. Then Γ is not satisfiable. Now we have a function * such that for $i < n$,

1. If $p_i \in I\text{-}Sub(\Gamma)$, then $(p)^*=1$;
2. If $[p_i] \in I\text{-}Sub(\Gamma)$, then $(p)^*=0$.

By the third proposition proved above, α is not a tautology. □

6 Conclusion

This paper provides a new way to reason with Alpha graphs in a way to check easily whether an Alpha graph is tautologous or not, and the procedure presented in this paper can also be extended to the other two parts of Peirce's excellent logic system, Existential Graphs. On the other hand, Peirce always emphasized the experimental character of his rules and this character is missing in our approach. So, it should be understood as an additional approach for this area.

7 Acknowledgement

We would like to thank Frithjof Dau, Qingyu Zhang, and two anonymous referees for helpful comments on this paper.

References

[Ha95] E. Hammer: Logic and Visual Information, CSLI Publications, 1995.

[Kn62] W. Kneale and M. Kneale: The Development of Logic, Clarendon Press, 1962.

[Pe33] C. S. Peirce: Collected Papers of C. S. Peirce, volume 4, Charles Hartshorne and Paul Weiss, editors, Harvard University Press, 1933.

[Ro64] D. D. Roberts: "The Existential Graphs and Natural Deduction," In Edward Moore and Richard Robin, editors, Studies in the Philosophy of Charles Sanders Peirce. University of Massachusetts Press, 1964.

[Ro73] D. D. Roberts: The Existential Graphs of Charles S. Peirce, Mouton and Co., 1973.

[Sh13] H. Sheffer: "A Set of Five Independent Postulates for Boolean Algebras, with application to logical constants," Transactions of the American Mathematical Society, 1913.

[Sh02] S. J. Shin: The Iconic Logic of Peirce's Graphs, The MIT Press, 2002.

[Wh84] R. White: "Peirce's Alpha Graphs: The Completeness of Propositional Logic and the Fast Simplification of Truth Functions," Transactions of the Charles S. Peirce Society 20, 1984.

[Ze67] J. Zeman: The Graphical Logic of C. S. Peirce, Doctoral Dissertation, University of Chicago, 1964.

Establishing connections between Formal Concept Analysis and Relational Databases

Uta Priss

School of Computing, Napier University, Edinburgh, UK
www.upriss.org.uk
u.priss@napier.ac.uk

Abstract. The relationship between relational databases and formal concept analysis (FCA) has been the topic of several papers in the past. This paper intends to extend some of the ideas presented in the previous papers by analysing the relationship between FCA and two central notions of relational databases: database schemata and normalforms. An object-relational algebra is suggested in this paper as a possible future replacement of relational algebra.

1 Introduction

The relationship between relational databases and formal concept analysis (FCA, cf. Ganter & Wille (1999)) has been the topic of several papers in the past. Hereth (2002) discusses relational scaling and databases. He translates relational algebra, which provides a theoretical foundation for relational databases, into an FCA representation. This paper intends to extend the previous research on FCA and databases, by analysing areas of database design in which FCA methods could be useful for modelling and visualisation: the areas of database schemata and normalforms.

This paper attempts to develop the foundations for an "object-relational algebra", which one day may serve as a substitute for the traditional relational algebra used as the foundation of the database query language SQL (cf. Negri et al. (1991)). Object-relational algebra is compatible with FCA because it uses binary relations, which can be interpreted as formal contexts. Some of the context concatenations proposed in object-relational algebra have already been used in other FCA applications (such as described by Priss(1998)).

With respect to database normalforms, Hereth (2002) has already described the relationship between FCA and functional dependencies, which are essentially implications between different columns of the database tables. Related to functional dependencies are several normalforms, which are defined in traditional database theory as a means of reducing redundancy and avoiding update anomalies in relational databases. In this paper, it is argued that FCA can also be used to visualise normalforms, which could be beneficial primarily for educational purposes. To our knowledge, so far this relationship between normalforms and FCA has never been stated in any research papers.

2 An Object-Relational Algebra

In traditional database theory, a relational algebra is used as the underlying formalisation for query optimisation and formal semantics (cf. Negri et al. (1991)). This relational algebra is closely tied to SQL because most of its operations correspond directly to SQL keywords. Because of this close connection, it can be difficult to extend relational algebra to more general languages, such as object-relational formalisms and other relational operations. It is therefore of interest to study possible alternatives to relational algebra.

There are at least three types of algebras which are similar or related to relational algebra: first, there are relation algebras (i.e. "relation" instead of "relational"). They are studied in the field of algebraic logic and can be traced back to Peirce and de Morgan via Tarski (1941) and Schröder (cf. Pratt (1992) for an overview). Second, there are Krasner algebras, which have been explored with respect to their relationship to Peirce's algebra by Hereth Correia & Pöschel (2004). Third, different variations have been suggested to Codd's (1970) original relational algebra in the field of relational databases.

The approach suggested in this paper is to use relation algebra (i.e. a Tarski-Peircean algebra) instead of relational algebra as the foundation for relational databases. This relation algebra (RA) is interpreted, first, with respect to Boolean matrices (cf. Kim (1982)) and, second, with respect to formal contexts in FCA. The goal is to develop an algebra which is suitable both for relational databases but also for more general object-relational structures. We call this algebra "object-relational algebra".

In object-relational algebra, binary relations, which correspond to Boolean matrices and to (binary) formal contexts in FCA, are used to represent most aspects of a database schema, such as table JOINs and subtype relations. Because of this, the terms "relation" and "matrix" may be used interchangeably in this paper. In addition to binary relations, many-valued matrices, which correspond to many-valued contexts in FCA, are used to represent the values of relational database tables.

2.1 An interpretation of relation algebra with respect to FCA contexts

A Tarski-Peircean relation algebra (RA) consists of a set with operations for complementation ($^-$), dual (^{d}), logical OR ($\cup$), composition ($\circ$), null element (nul) and compositional one-element (dia). Maddux (1996) provides a detailed overview of these operations and of the axioms and basic theorems of this algebra. Further operations can be derived from the initial ones, such as logical AND ($\cap$) and the de Morgan complement of composition ($\bullet$). The complements of nul and dia are $\overline{nul}$ and $\overline{dia}$, respectively. It appears that RA has a large field of applications. Pratt (1993) explains that Chu spaces can be seen as an interpretation of RA. Maddux derives programming language semantics as an interpretation of RA.

In this paper the RA operations are interpreted with respect to Boolean matrices (in the sense of Kim (1982)). The RA operations cannot be applied to all Boolean matrices without further restrictions. For example, $\cup$ and $\cap$ require the matrices to have equal numbers of columns. In the case of composition (see the example below), the number of columns of the left matrix must equal the number of rows of the right matrix. Thus the operations are only partially defined on the set of all Boolean matrices. In matrices, logical AND and OR are calculated position-wise combining each value of the left

matrix with the one at the same position in the right matrix. Different one-elements are required for matrices of different sizes. But all one-elements are of the form, $dia_{\mathtt{m},\mathtt{n}}$, with m rows and n columns, 1s on the diagonal and 0s otherwise. The null elements, $nul_{\mathtt{m},\mathtt{n}}$, are matrices with just 0's. Two new types of operations are of relevance for Boolean matrices: first, for matrices I and J, apposition and subposition consist of adding rows or columns (written as $I|J$ and $\frac{I}{J}$). Second, the reduction operation red deletes certain rows and columns from a matrix.

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \circ \begin{pmatrix} e & f \\ g & h \end{pmatrix} = \begin{pmatrix} (ae+bg) & (af+bh) \\ (ce+dg) & (cf+dh) \end{pmatrix}$$

$$\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \circ \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

Example 1. Matrix composition.

The interpretation of RA with respect to Boolean matrices can be extended by considering the matrices to be the cross-tables of formal contexts in the sense of formal concept analysis (Ganter & Wille, 1999). RA becomes thus an algebra of operations on FCA contexts. This interpretation requires some further adjustments of the operations. In the rest of this paper, it is assumed that in each formal context, the set $\mathtt{G}$ of formal objects and the set $\mathtt{M}$ of formal attributes are linearly ordered and that if different formal contexts share a set of objects or attributes then these are in the same linear order. The RA operations may only be meaningful for formal contexts, if they are applied to formal contexts which share sets of objects and/or attributes. For example, composition of formal contexts usually assumes that each attribute of the left context equals an object of the right context.

To distinguish the traditional set-based FCA notations from matrices, type-writer font is used in this paper for sets and elements while italics are used for matrices and formal contexts. Unless stated otherwise, for the rest of this paper, $K = (\mathtt{G}, \mathtt{M}, I)$ denotes a formal context with $\mathtt{G}$ as set of objects and $\mathtt{M}$ as set of attributes. $\mathtt{H} \subseteq \mathtt{G}$; $\mathtt{N} \subseteq \mathtt{M}$ are subsets and $\mathtt{g} \in \mathtt{G}$; $\mathtt{m} \in \mathtt{M}$ are elements.

The interpretation of RA on FCA contexts necessitates the introduction of further operations so that matrices can be converted into sets (of objects or attributes) and vice versa. For a linearly ordered set $\mathtt{T}$ and a subset $\mathtt{S}$ of $\mathtt{T}$, the operation $row_{\mathtt{T}}(\mathtt{S})$ denotes a row matrix of length $|\mathtt{T}|$ which has a 1 in the position of each element of $\mathtt{S}$ that is also in $\mathtt{T}$ and 0's otherwise. The operations $col_{\mathtt{T}}(\mathtt{S})$ and $dia_{\mathtt{T}}(\mathtt{S})$ are defined accordingly: $col_{\mathtt{T}}(\mathtt{S})$ producing a column matrix and $dia_{\mathtt{T}}(\mathtt{S})$ producing a matrix with 1's on the diagonal according to the positions of the elements in $\mathtt{S}$ and 0's otherwise. (Both $col_{\mathtt{T}}(\mathtt{S})$ and $row_{\mathtt{T}}(\mathtt{S})$ can be derived from $dia_{\mathtt{T}}(\mathtt{S})$ through composition with $\overline{nul}$).

With the notations from the previous two paragraphs, sets in a formal context can be converted into matrices as follows: $G := row_{\mathtt{G}}(\mathtt{G})$, $H := row_{\mathtt{G}}(\mathtt{H})$, $\underline{g} := row_{\mathtt{G}}(\{\mathtt{g}\})$, $M := col_{\mathtt{M}}(\mathtt{M})$, $N := col_{\mathtt{M}}(\mathtt{N})$, $\underline{m} := col_{\mathtt{M}}(\{\mathtt{m}\})$. Underlining is used to indicate that the matrices correspond to single-element sets and thus contain exactly one 1. For the remainder of this paper, row or column matrices with a single 1 shall be called "single-element matrices". The matrices can be converted back into sets using the following operations: $\mathtt{H} = \mathtt{set}_{\mathtt{G}}(H)$ and $\mathtt{N} = \mathtt{set}_{\mathtt{M}}(N)$.

The main operations of FCA can now be represented as summarised in table 1. The plus ($+$) operator, which is somewhat dual to the prime ($'$) operator originates from use in lexical databases (cf. Priss(1998) and Priss & Old (2004)). The operations on attributes $\underline{m}'$, $\underline{m}^+$, N' and N^+ are analogous to the ones in the table. It should be noted that for single-element matrices, the plus and prime operators yield the same result (i.e., $\underline{g} \circ I = \overline{\underline{g} \circ \overline{I}}$) because composing with $\underline{g}$ corresponds to selecting one row from I, or, in standard FCA terminology, the existence- and all-quantifiers are equivalent for single-element sets.

standard FCA	relation algebra
$\mathtt{g}I\mathtt{m}$	$\underline{g} \circ I \circ \underline{m} = (1)$
$\mathtt{g}' := \{\mathtt{m} \in \mathtt{M} \mid \mathtt{g}I\mathtt{m}\}$	$\underline{g}' = \underline{g}^+ := \underline{g} \circ I$
$\mathtt{H}' := \{\mathtt{m} \in \mathtt{M} \mid \forall_{\mathtt{g} \in \mathtt{G}} : \mathtt{g} \in \mathtt{H} \Longrightarrow \mathtt{g}I\mathtt{m}\}$	$H' := \overline{H \circ \overline{I}}$
$\mathtt{H}^+ := \{\mathtt{m} \in \mathtt{M} \mid \exists_{\mathtt{g} \in \mathtt{G}} : \mathtt{g} \in \mathtt{H} \text{ and } \mathtt{g}I\mathtt{m}\}$	$H^+ := H \circ I$

Table 1. FCA terminology translated into relation algebra

$\mathtt{N}^+$	$I \circ N$	$\exists_{\mathtt{m} \in \mathtt{M}} : \mathtt{m} \in \mathtt{N} \text{ and } \mathtt{g}I\mathtt{m}$	at least one, some
$\mathtt{G} \setminus \mathtt{N}^+$	$\overline{I \circ N}$	$\neg\exists_{\mathtt{m} \in \mathtt{M}} : \mathtt{m} \in \mathtt{N} \text{ and } \mathtt{g}I\mathtt{m}$	none
$\mathtt{N}'$	$\overline{\overline{I} \circ N}$	$\neg\exists_{\mathtt{m} \in \mathtt{M}} : \mathtt{m} \in \mathtt{N} \text{ and } \neg(\mathtt{g}I\mathtt{m})$	relates to all
$\mathtt{G} \setminus \mathtt{N}'$	$\overline{I} \circ N$	$\exists_{\mathtt{m} \in \mathtt{M}} : \mathtt{m} \in \mathtt{N} \text{ and } \neg(\mathtt{g}I\mathtt{m})$	does not relate to all
$(\mathtt{M} \setminus \mathtt{N})^+$	$I \circ \overline{N}$	$\exists_{\mathtt{m} \in \mathtt{M}} : \mathtt{m} \notin \mathtt{N} \text{ and } \mathtt{g}I\mathtt{m}$	relates to those that are not only
$\mathtt{G} \setminus ((\mathtt{M} \setminus \mathtt{N})^+)$	$\overline{I \circ \overline{N}}$	$\neg\exists_{\mathtt{m} \in \mathtt{M}} : \mathtt{m} \notin \mathtt{N} \text{ and } \mathtt{g}I\mathtt{m}$	relates to those that are only
$(\mathtt{M} \setminus \mathtt{N})'$	$\overline{\overline{I} \circ \overline{N}}$	$\neg\exists_{\mathtt{m} \in \mathtt{M}} : \mathtt{m} \notin \mathtt{N} \text{ and } \neg(\mathtt{g}I\mathtt{m})$	relates to all outside
$\mathtt{G} \setminus ((\mathtt{M} \setminus \mathtt{N})')$	$\overline{I} \circ \overline{N}$	$\exists_{\mathtt{m} \in \mathtt{M}} : \mathtt{m} \notin \mathtt{N} \text{ and } \neg(\mathtt{g}I\mathtt{m})$	does not relate to all outside

Table 2. Eight quantifiers

Operations on FCA contexts have been known for a while (cf. Ganter & Wille (1999)). But to our knowledge there has not been a significant amount of interest among FCA researchers in the RA that underlies such operations. The reason for this may be that in many FCA applications, context operations are not of relevance because they may not lead to interesting properties of the lattice visualisations. We argue, however, that in the field of databases, such context operations are of major importance because they can express quantified relationships between database tables. A many-valued database table can be interpreted as a binary matrix by replacing every non-NULL value with 1 and every NULL value with 0. Using such binary matrices, quantified relationships among database tables can be described using RA as summarised in table 2. The plus and prime operators and their complements roughly correspond to eight basic natural language quantifiers: “some”, “none”, “all”, “not all”, “only”, “not only”, “all outwith”, “not all outwith”. This should be of interest to relational databases because the currently most frequently used database language SQL supports “exists”

as a basic quantifier, but not any of the other seven. Hansen & Hansen (1996, p. 202) observe that the lack of an "all" quantifier in SQL creates problems for database users. Another advantage of this approach is that in contrast to SQL which always requires a three-valued logic because of NULL (cf. Negri et al. (1991)), in this approach many operations can be performed in the two-valued binary logic.

2.2 Compositional Schemata

One goal for this paper is to construct a complex formal context which summarises all tables of a given relational database. In order to reach this goal, the notion of a "compositional schema" is introduced. A compositional schema (as shown in figure 1) consists of several formal contexts, some of which are composed using any of the quantifiers in table 2. Such compositional schemata are called "relational context schemata" by Priss (1998). Similar concatenations using formal contexts are discussed by Ganter & Wille (1999) and, for example, by Faid et al. (1997).

	A	B	C
B	1	2	3
A	4	5	6
D	7	8	9

	A	B	C
B			L
A		J	$J \circ L$
D	I	$I \circ J$	$I \circ J \circ L$

Fig. 1. A compositional schema

The numbering of the nine cells as presented in the left hand side of figure 1 shall be used in the remainder of this paper. The compositional schema in figure 1 is built from the formal contexts $K_J := (\mathtt{A}, \mathtt{B}, J)$; $K_I := (\mathtt{D}, \mathtt{A}, I)$ and $K_L := (\mathtt{B}, \mathtt{C}, L)$. Because K_I and K_J share the set $\mathtt{A}$, a context $K_{I \circ J} := (\mathtt{D}, \mathtt{B}, I \circ J)$ can be formed. Similarly, a context $K_{J \circ L}$ can be formed. Instead of the existence quantifier used in the construction of the matrices in cells 6, 8 and 9, any of the other seven quantifiers from table 2 can be used. A further context $K_{I \circ J \circ L}$ can be formed in cell 9 to complete the schema. It should be noted that while $K_{J \circ L}$ and $K_{I \circ J}$ are formed by composing the context to the left with the one above, $K_{I \circ J \circ L}$ is formed by composing the context to the left with the context two steps above (or the context two steps to the left with the one above). An exception is if J is a reflexive, transitive relation, in which case $J \circ J = J$ and $I \circ J \circ L = I \circ J \circ J \circ L$. The cells 1, 2, 4 can be left empty or cells 2 and 4 can be filled with an identity relation dia.

Figure 2 shows a compositional schema which can be constructed for any formal context $K := (\mathtt{G}, \mathtt{M}, I)$ where $\mathtt{P}(\mathtt{G})$ and $\mathtt{P}(\mathtt{M})$ denote the power set of $\mathtt{G}$ and $\mathtt{M}$, respectively, and $\in$ and $\ni$ denote the "element of" relation: $\mathtt{H} \ni \mathtt{g}$ and $\mathtt{m} \in \mathtt{N}$. Again, any of the 8 quantifiers from table 2 can be used in cells 6 and 8. This compositional schema summarises information relevant to K. for example, the rows of cell 8 show the chosen quantifier for every subset of $\mathtt{G}$. It can be shown that if the set $\{\mathtt{N}, \mathtt{M} \backslash \mathtt{N}\}$ is chosen instead of $\mathtt{P}(\mathtt{M})$, if $I \circ \in$ is chosen for cell 6, and a lattice diagram is constructed for cells 2, 3, 5, 6, all 8 quantifiers according to table 2 can be read from the lattice diagram.

	G	M	P(M)
M		dia	$\in$
G	dia	I	$I \circ \in$
P(G)	$\ni$	$\ni \circ I$	

Fig. 2. A compositional schema for $(\mathtt{G}, \mathtt{M}, I)$

Compositional schemata can also be utilised for summarising information about object-oriented class hierarchies. The relational composition, $\circ$, seems to be a natural means for expressing inheritance with respect to object-oriented modelling. Figure 3 shows a compositional schema for a class in the sense of object-oriented modelling. The matrix I_{inst} describes which instances belong to which classes; I_{sub} contains the class hierarchy; I_{attr} describes which classes have which attributes. It should be noted that I_{attr} only refers to the existence of attributes not their values. For example, I_{attr} could specify that a class "employee" contains the attributes "employeeID", "name" and "address". This compositional schema assumes that attributes are named uniquely across the whole class hierarchy. But this condition can always be fulfilled by initially prefixing all attributes with the name of the class in which they are defined, and then, optionally, dropping those prefixes which are not required. The relation I_{sub} is assumed to be reflexive and transitive, thus $I_{inst} \circ I_{sub} \circ I_{attr} = I_{inst} \circ I_{sub} \circ I_{sub} \circ I_{attr}$.

	Classes	Classes	Attributes
Classes			I_{attr}
Classes		I_{sub}	$I_{sub} \circ I_{attr}$
Instances	I_{inst}	$I_{inst} \circ I_{sub}$	$I_{inst} \circ I_{sub} \circ I_{attr}$

Fig. 3. A compositional schema for class hierarchies

According to Atkinsion et al. (1989), there are at least four types of inheritance: substitution, inclusion, constraint and specialisation inheritance. Because various degrees of these types can be combined in actual systems, the notion of inheritance can be quite confusing. Object-relational algebra helps to clarify two of the four types: inclusion inheritance is the ordering of classes that emerges from I_{inst} because it refers to subset relations among the sets of instances of classes. Specialisation inheritance is the ordering of classes that emerges from I_{attr} because it refers to subset relations among the sets of attributes of classes. It should be noted that the concept lattice of the formal context based on cells 5, 6, 8 and 9 in figure 3 contains the information about both inclusion and specialisation inheritance and also about I_{sub} by itself in a single diagram - although in practical applications this diagram can become too large to be graphically displayed. The other two types of inheritance are based on behaviour (substitution inheritance) and values (constraint inheritance) and cannot be explained using just binary matrices. Constraint inheritance requires a many-valued relation instead of $I_{inst} \circ I_{sub} \circ I_{attr}$ which contains the actual values of the attributes for each instance. Such many-valued matrices are further discussed in the next section.

2.3 Relational database tables in a compositional schema

	Tables	Tables	Columns
Tables			I_{attr}
Tables		dia	I_{attr}
Keys	I_{inst}	I_{inst}	$I_{inst} \circ I_{attr}$

Fig. 4. A compositional schema for relational databases

The basic compositional schema for relational databases in figure 4 is a simplified version of the one for classes in figure 3. In relational databases, tables (or entity types) cannot be subtypes of other tables. For example, it cannot be expressed that a manager table shall be a subtype of a staff table. Therefore instead of an I_{sub} matrix, cell 5 is filled by an identity matrix dia. It follows that cell 9 contains $I_{inst} \circ I_{attr}$ because $I_{inst} \circ I_{attr} = I_{inst} \circ dia \circ I_{attr}$.

		Columns	
	Tables	Key Columns	Other Columns
Tables	dia	I_{attr}	
Keys	I_{inst}	$I_{inst} \circ I_{attr}$	

Fig. 5. A relational database schema

The first row and column in figure 4 do not contain any useful information and can be omitted. The only reason for presenting the schema in that manner is to demonstrate that relational databases are a special case of object-relational databases. After omitting the first row and column, the attributes can be grouped so that all key attributes appear on the left side of I_{attr}. Figure 5 shows the emerging schema and figure 6 shows an example. Figure 5 (or 6) contains all the information that is contained in a single relational database. Using the example in figure 6, the employee table (Emp) can be retrieved as follows: $\underline{Emp} = col_{\texttt{Tables}}(\{\texttt{Emp}\})$ retrieves the positions of the key attributes among the formal objects. $\texttt{G}_{Emp} = \texttt{set}(red_{\texttt{Keys}}(\underline{Emp'}))$ is the set of formal objects of Emp; $\texttt{M}_{Emp} = \texttt{set}(red_{\texttt{Columns}}(\underline{tEmp'}))$ is the set of formal attributes; $I_{Emp} = red_{\texttt{G}_{Emp},\texttt{M}_{Emp}}(I_{inst} \circ dia(\underline{Emp}) \circ I_{attr})$ is a binary matrix for the relation of Emp and, finally, $C_{Emp} = (\texttt{G}_{Emp}, \texttt{M}_{Emp}, I_{Emp})$ represents a formal context for Emp. The matrix I_{Emp} is binary and contains a 1 for every non-NULL value and 0 for every NULL value. The actual values of the table can be stored using a mapping $mv()$ from binary matrices to many-valued matrices. The full database table Emp thus corresponds to the many-valued formal context $(\texttt{G}_{Emp}, \texttt{M}_{Emp}, mv(I_{Emp}))$.

All tables of a relational database can be combined in this manner in one many-valued matrix (as in figure 6) if three assumptions are made:

– First, if the same name of an attribute is used in different tables it should have a similar meaning - otherwise the name should be prefixed by the table name.

	Emp	Work	Proj	eID	pID	ename	eaddr	cost	pname	ptime
tEmp	1			1		1	1			
tWork		1		(1)	(1)			1		
tProj			1		1				1	1
e1	1			1		paul	UK			
e2	1			2		mary	UK			
e3	1			3		carl	USA			
e4	1			4		sue	USA			
p1			1		1				A	2 wk
p2			1		2				B	2 wk
p3			1		3				C	30 wk
e1/p1		1		1	1			100		
e1/p2		1		1	2			100		
e1/p3		1		1	3			200		
e2/p1		1		2	1			400		
e3/p2		1		3	2			100		
e3/p3		1		3	3			200		

Fig. 6. An example of a relational database schema

– Second, entities are identified by keys.
– Third, the multiple inheritance anomaly (see below) is avoided.

With respect to the second assumption, in many relational databases, tables use foreign keys as part of composite keys, but the exact same set of key attributes is usually unique to each table. For such databases, each entity is unique to a single table and the cells 7, 8, 9 in figure 4 will simply show concatenations of the individual relations. But in some cases, relational databases contain hidden inheritance structures, which contain overlapping entities. For example, a database could contain separate tables for staff and managers but some employees could be both staff and manager. In this case, the database needs to be carefully designed to avoid a multiple inheritance anomaly. The schema in figure 5 helps identifying such possible problems. A possible problem occurs if the same keys are used in different tables and those different tables use the same attributes. In figure 6, all tables have different keys. Even though `eID` and `pID` are used in more than one table, they are part of a composite key in `Work` which is different from their use in `Emp` and `Proj`. A multiple inheritance anomaly would occur if a key is used in different tables relating to the same attribute, but the attribute has different values in each table. In that case the mapping $mv()$ cannot be defined.

The formal context in figure 5 contains all of the information that is contained in a power context family (Wille, 2002). This representation is different from Hereth's (2002) representation of relational database tables in FCA. Because power context families can represent conceptual graphs, a connection from relational databases to conceptual graphs can thus be established. The general idea of object-relational algebra is to define an algebra that uses compositional schemata as basic elements, that uses operations from RA and that contains a binary RA as a subalgebra. Due to space limitations, further details cannot be discussed in this paper.

2.4 Object-relational algebra and SQL

Using the definitions so far, it can now be attempted to translate SQL expressions using the RA operations described so far. Obviously, this is a complex task, only the beginnings of which can be explored in this paper. Two of the basic SQL operations are projection and selection. Projection corresponds to composition with a dia matrix from the right and selection corresponds to composition with a dia matrix from the left. For example, `select ename,eaddr from Emp` corresponds to $I_{Emp} \circ dia(\underline{ename} \cup \underline{eaddr})$ Selection is usually based on constraints. In object relational algebra, such constraints often require the use of many-valued matrices. For each many-valued attribute a so-called value context V_{attr} is constructed depending on the type of the attribute. Due to space limitations the details cannot be discussed in this paper. But the construction is similar to conceptual scaling (Ganter & Wille, 1999) or relational scaling (Hereth, 2002) in FCA. To provide an example for a select statement:
`select ename,eaddr from Emp where ename = 'mary' and eaddr = 'UK'` corresponds to
$dia((V_{ename} \circ \underline{mary}) \cap (V_{eaddr} \circ \underline{UK})) \circ I_{Emp} \circ dia(\underline{ename} \cup \underline{eaddr})$.

The natural JOIN in relational databases is fairly closely related to relational composition. This is because a natural JOIN combines two tables based on a shared column. A difference between an SQL JOIN and $\circ$ is, however, that depending on how the natural JOIN is formed (e.g. whether subqueries are involved), the query can return any number of duplicates. This is why such queries are often formulated using "select distinct" which eliminates the duplicates.

Consider the following example based on figure 6. Three database tables are Emp (an employee table), $Proj$ (a table for projects) and $Work$ (a table which shows the relationship of which employee works on which project). In SQL, it is fairly easy to retrieve all employees which work on a project:

```
select * from Emp, Work where Emp.eID=Work.eID
```

It is somewhat more difficult to retrieve all employees which do not work on any project:

```
select * from Emp where not exists
(select * from Work where Emp.eID=Work.eID)
```

And it is even more difficult to retrieve all employees who work on all projects:

```
select * from Emp where not exists
(select * from Proj where not exists
(select * from Work where Emp.eID=Work.eID
and Work.pID = Proj.pID))
```

A comparison with table 2 shows that the number of negations (relational complements) in SQL corresponds to the ones in table 2. But in contrast to the relational composition in the second column from table 2, the SQL statements vary in complexity. Of course, all of the statements above could be written using 2 subselect statements so that they are of equal complexity. But in our experience, many average database users would have extreme difficulty formulating SQL statements of such complexity. Most users would probably create a view or a temporary table, if they were asked to retrieve employees on all projects, to avoid using two subselect statements. In object-relational algebra, all of the statements above would be formed using $dia(I_{Work} \circ \underline{pid}) \circ I_{Emp}$ and using any of the quantifiers in the second column of table 2.

3 Normalforms and FCA

3.1 A brief introduction to database normalforms

This section explores a second means of using FCA for modelling an aspect of relational databases. In relational database design, it is of importance to avoid data redundancy and anomalies that can occur when data is inserted, deleted or updated (cf. any textbook on relational databases). The standard solution to such problems is database normalisation. Several of so called "normalforms" are in use, some of which rely on the notion of "functional dependency". Hereth (2002) has shown that a database can be translated into a power context family and that the functional dependencies of such a database correspond to FCA implications in a certain formal context. In this paper, FCA is used as a means for visualising the structures of the different normalforms by representing functional dependencies as implications in a lattice as Hereth (2002) proposed. It is not suggested that these visualisations solve any computational problem or create new means for practical implementations. Instead, the visualisations are meant to serve as explanatory aids. In our experience, many students tend to experience a great deal of difficulty in grasping the idea of normalisation. Thus a visual aid should be beneficial.

This section mainly focuses on the Second and Third Normalform (2NF and 3NF) and on the Boyce Codd Normalform (BCNF), which are all briefly described below. It should be noted that beyond the textbook versions of the standard normalforms, there have also been some other normalforms proposed in the literature. The sequence of the normalforms has emerged due to historical reasons, which explains why BCNF was inserted between 3NF and 4NF. The normalforms, 2NF, 3NF and BCNF, rely on functional dependencies within a single table. Functional dependencies can either describe the current state (an attribute is dependent on another one) or a prescriptive state (an attribute must always depend on another one). Thus it is not always unambiguous as to what constitute the functional dependencies of a database table. Other types of dependencies utilise JOINs (such as 5NF) and inter-table dependencies. In general, although current database researchers consider the study of normalforms as completed, it might still be interesting to revisit this area from an FCA viewpoint, to shed some light on the differences and relations between the normalforms and to investigate whether there are any new insights for FCA implications to be drawn from database research.

According to Hansen & Hansen (1996) "a relation is in first normalform (1NF) if the values in the relation are atomic". In other words, the values are elements, not sets, tuples or subtables. Relational databases usually fulfil this normalform automatically (although modern object-relational databases may not - but that is a different story). A database table is in 2NF if it is in 1NF and "no non-key attribute is functionally dependent on just a part of the key" (Hansen & Hansen, 1996). A database table is in 3NF if it is in 2NF and there are no transitive functional dependencies, where "transitivity" refers to the existence of functional dependencies among non-key attributes. A database table is in BCNF if it is in 3NF and every non-trivial, left-irreducible functional dependency has a candidate key as its determinant (where a "trivial" implication is one where the left-hand side and the right-hand side are identical or where the left-hand side is either the empty set or the full set of attributes; "determinants" are left-hand sides of functional dependencies and "candidate key" means that these attributes could be used as

an alternative key). Other normalforms, such as 4NF and 5NF, shall be ignored in this paper.

3.2 FCA visualisations of database normalforms

To use FCA for visualising normalforms, the functional dependencies (including the dependencies on the key attributes) are considered as an implicational basis of a lattice. This lattice is called the "FD lattice" of a database table for the remainder of this paper. It can be used to visually inspect the normalforms. Because by definition all attributes of a database table must be dependent on the key, the meet of the key attributes must equal the bottom node of the FD lattice. The easiest normalform to be visualised is BCNF as described in the following proposition.

> **Proposition**. A database table is in BCNF if
> 1) all non-trivial implications of its FD lattice have a left-hand side which meets in the bottom node of the lattice;
> 2) if the lattice contains any non-key attributes, then no true subset of the key attributes can meet in the bottom node; and
> 3) if a set consisting of only non-key attributes meets in the bottom node, then these must be the only non-key attributes in the lattice.

This proposition is simply a translation of the BCNF definition into FCA terminology: for a determinant to be a candidate key means that the meet of its attributes equals the bottom node because all keys meet in the bottom node. Item 2) and 3) of the proposition ensure that the database table is also in 2NF and 3NF, respectively. Because of item 1), the conditions for 2NF and 3NF are much simpler than in the general case because only implications based on the bottom node need to be considered. It should be noted that 2NF and 3NF only restrict implications from key to non-key attributes and among non-key attributes. Implications from non-key to key attributes are not restricted in 2NF and 3NF, which is why items 2) and 3) make careful consideration of key and non-key attributes.

The examples of FD lattices in figure 7 shall help to illustrate some of these points. In this figure, key attributes are distinguished from other attributes by enclosing their names in a box. If the lattice contains only two nodes, then either the lattice is isomorphic to the one depicted in the far left example in figure 7, or it has only key attributes, in which case, more than one attribute can be attached to the bottom node. In general for BCNF lattices of any size, attributes must either be attached to the bottom node or they must be attached to co-atoms. If any of the co-atomic attributes meet above the bottom node, then the filter above this node must be a complete Boolean lattice (so that no subset of the attributes implies the other attributes).

2NF and 3NF are more difficult to visualise. One problem is that there is no FCA notion of key versus non-key attributes. The lattice on the left side of figure 8 is identical to the lattice in the middle from an FCA viewpoint. But the left lattice is not in 2NF because non-key attribute b depends on the partial key c. The lattice in the middle is in 2NF and 3NF but not in BCNF because c implies b but c is not a candidate key. The decomposition of this lattice into two lattices that are in BCNF is shown on the right.

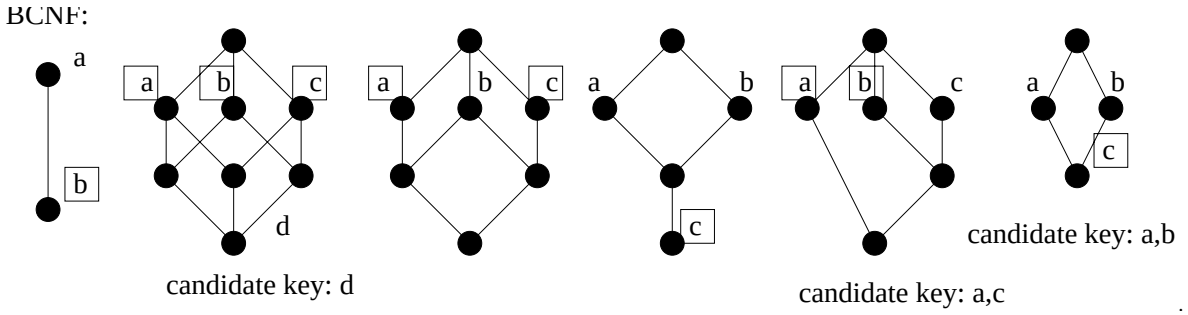


Fig. 7. FD lattices for relations in BCNF

This decomposition does not preserve the dependencies, because now a implies c. It is a known fact in the literature that BCNF decomposition can sometimes not be achieved with dependency preservation. It would be an interesting FCA question, whether from a lattice structural approach it can be determined in which cases BCNF is predictably dependency preserving and in which cases it is not.

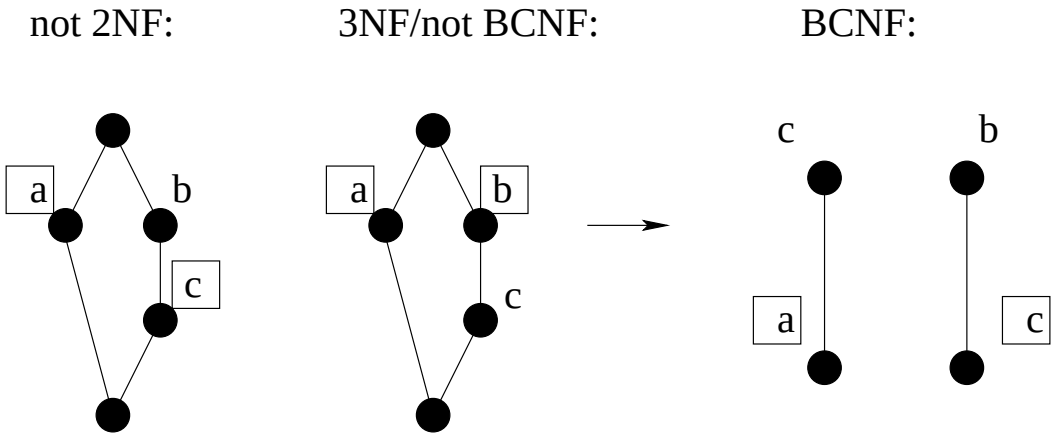


Fig. 8. The role of keys and BCNF decomposition

Figure 9 shows some more examples of lattices which are not in BCNF. It should be pointed out that the sequence of the normalforms (first, second, third, ...) is somewhat random because of its historical origin. In our experience with tutorials, students tend to detect and remove transitive dependencies (the main 3NF problem) before they tackle partial key dependencies (i.e., 2NF problems). Thus 3NF may in some way be more basic than 2NF.

For determining whether an FD lattice is 2NF or 3NF, all non-trivial implications other than the ones whose left-hand side meets in the bottom node need to be checked. Obviously, this can mean that many nodes need to be checked and - from a practical viewpoint - it may be easier to check the implication list directly without considering the lattice. But as explained before, the lattice visualisations can still help to convey an understanding of what the different normalforms mean. For 2NF, the meet of any subset of key attributes needs to be checked. In the examples in figure 9, a and b imply d in

the middle lattice and c implies a in the right lattice. These are both violations of 2NF. For 3NF, it first needs to be determined whether the lattice is in 2NF. In the next step, only non-key attributes need to be considered. If any implications can be found among non-key attributes at all, then 3NF is violated.

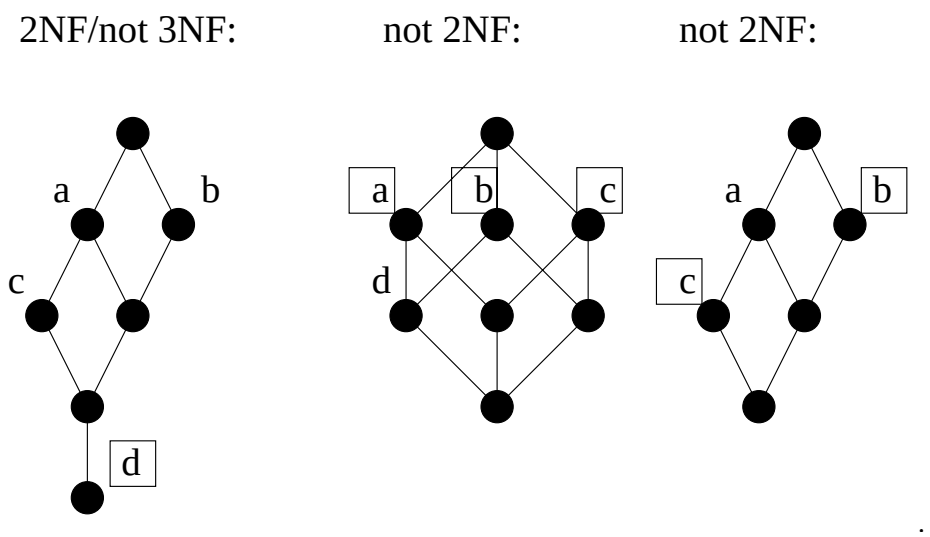


Fig. 9. FD lattices for relations in/not in 2NF and 3NF

4 Conclusion

This paper presents two applications of FCA with respect to relational databases. Object-relational algebra is suggested as a means for managing database schemata. FCA visualisations of database normalforms can serve as a means for exploring internals of such normalforms. Both topics still leave many open research questions.

A long-term goal for the research undertaken for this paper is to, ultimately, build FCA database exploration software, which can be used as a front-end to relational databases of any content. As far as we know, the only tool which currently exists that can be used in that manner is Cernato. But Cernato is a commercial tool and not freely available. Also it focuses on scaling of attributes and does not facilitate joining of database tables. As Cernato demonstrates, such FCA tools require user interaction and probably some degree of user training. It is not possible for users to simply “press a button” and view results. But other database exploration tools as used in Business Intelligence also require user interaction and training. Thus, in our opinion, FCA stands a chance to compete with such tools. To minimise the workload of users, an FCA database front-end would have to be carefully designed. We hope that research as presented in this paper might highlight some of the structures that could lead the way.

References

1. Atkinson, M.; Bancilhon, F.; DeWitt, D.; Dittrich, K.; Maier, D.; Zdonik, S. (1989). *The Object-Oriented Database System Manifesto.* In: Proceedings of the First International Conference on Deductive and Object-Oriented Databases, Kyoto, Japan. p, 223-240.
2. Codd, E. (1970). *A relational model for large shared data banks.* Communications of the ACM, 13:6.
3. Faid, M.; Missaoui, R.; Godin, R. (1997). Mining Complex Structures Using Context Concatenation in Formal Concept Analysis. In: Mineau, Guy; Fall, Andrew (eds.), Proceedings of the Second International KRUSE Symposium. p. 45-59.
4. Ganter, B.; Wille, R. (1999). *Formal Concept Analysis.* Mathematical Foundations. Berlin-Heidelberg-New York: Springer, Berlin-Heidelberg.
5. Hansen, Gary; Hansen, James (1996). *Database Management and Design.* Prentice Hall.
6. Hereth, J. (2002). *Relational Scaling and Databases.* In: Priss; Corbett; Angelova (Eds.) Conceptual Structures: Integration and Interfaces. LNCS 2393, Springer Verlag. p. 62-76.
7. Hereth Correia, J.; Pöschel, R. (2004). *The Power of Peircean Algebraic Logic (PAL).* In: Eklund (Ed.) Concept Lattices. LNCS 2961, Springer Verlag. p. 337-351.
8. Kim, K. H. (1982). *Boolean Matrix Theory and Applications.* Marcel Dekker Inc.
9. Maddux R. (1996). *Relation-algebraic semantics.* Theoretical Computer Science, 160, p. 1-85.
10. Negri, M.; Pelagatti, G.; Sbattella, L. (1991). *Formal Semantics of SQL Queries.* ACM Transactions on Database Systems, 17, 3. p. 513-534.
11. Pratt, V.R. (1992). *Origins of the Calculus of Binary Relations.* Proc. IEEE Symp. on Logic in Computer Science, p. 248-254.
12. Pratt, V.R. (1993). *The Second Calculus of Binary Relations.* Proc. 18th International Symposium on Mathematical Foundations of Computer Science, Gdansk, Poland, Springer-Verlag, p. 142-155.
13. Priss, U. (1998). *Relational Concept Analysis: Semantic Structures in Dictionaries and Lexical Databases.* (PhD Thesis) Verlag Shaker, Aachen 1998.
14. Priss, U.; Old, L. J. (2004). *Modelling Lexical Databases with Formal Concept Analysis.* Journal of Universal Computer Science, Vol 10, 8, 2004, p. 967-984.
15. Tarski, A. (1941). *On the calculus of relations.* Journal of Symbolic Logic, 6, p. 73-89.
16. Wille, Rudolf (2002). *Existential Concept Graphs of Power Context Families.* In: Priss; Corbett; Angelova (Eds.) Conceptual Structures: Integration and Interfaces. LNCS 2393, Springer Verlag, p. 382-395.

Conceptual Graph based Criminal Intelligence Analysis

Rhys N. Reed and Pavel Kocura

Department of Computer Science
Loughborough University
Loughborough, Leicestershire. LE11 3TU
England
R.N.Reed@lboro.ac.uk and P.Kocura@lboro.ac.uk

Abstract. This paper addresses the use of Conceptual Graphs as an underlying representation for Criminal Intelligence Analysis. Mapping between CGs and existing CIA representations, which incorporate the attributes 'Strength' and 'Validity' are demonstrated. The concept of Repeating Criminal Occurrence Patterns, a special form of Canonical Model, is introduced. Given criminological data in CG form the ability to reason and generate hypotheses is examined and a proposed architecture submitted.

1 Introduction

Criminal Intelligence Analysis is the process used by both government and private agencies throughout the world. The purpose of which is to identify and understand the relationships in the domain of Crime.

Criminal Intelligence Analysis, in particular the methodologies developed by ANACAPA Sciences Inc, were developed in the late 1970's in response to organized crime, and remain the core techniques. Criminal Intelligence Analysis can be subdivided into Operational and Tactical. The representations used in Criminal Intelligence Analysis are varied. By using a formal underlying representation it would be possible to assist the Intelligence process. This paper concentrates on the representation and automated reasoning of Criminological Data using Conceptual Graphs [1] for Operational Techniques. Operational Techniques concentrate on the identification, arrest or disruption of existing criminal activity.

2 Representation

Many of the representations used in Criminal Intelligence Analysis are restricted orthogonal views of intelligence data. The representations are restricted in that only a subset of the total information is shown, the defining terms for the generation of the subset being what is deemed important. The purpose being to allow the investigator to comprehend the information. Various representations exist and new ones are slowly being added. The most frequent used shall be examined.

2.1 Link Analysis

Link Analysis chart represent graphically the relationship between concepts. The first example of a Criminal Intelligence Analysis representation is that of a Link Chart as shown in Figure 1. A link chart consists of concepts, which usually utilize iconic representation to allow rapid identification and arcs which are labelled with the relationship. Link Charts are one of the more frequent representations used.[2] This representation can easily be converted to CG format as follows:

[Man:Pavel] <-(owner) <-[M 96 Falcoln:#12345]

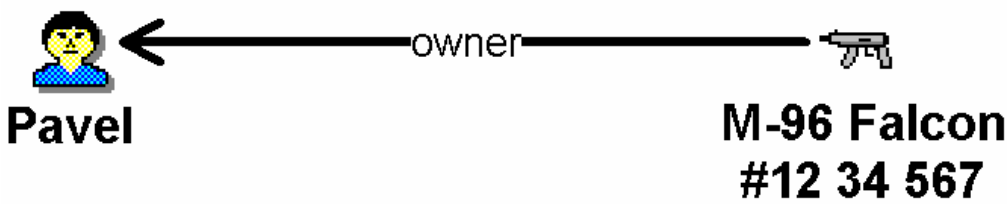


Fig. 1. Link Chart stating Pavel is the owner of an M-96 Falcon number 1234567

However, Link Analysis charts lack a formal definition. Directions on arcs are optional and may also be shown as bi-directional. So whilst it is relatively straight forward to map a CG to a Link Analysis Chart, the converse is not true. There are various ways that this could be overcome. Two possible options are requiring users to enter additional information and using Canonical Models [6]. Requiring users to enter directions on the arcs, although an apparent simple solution suffers from the problem that the user may incorrectly enter the information. Given that our goal is to reason about the information, syntactic correctness is essential. Canonical Models provide a solution to both conversion of information and ensuring syntactic correctness. Given a graph such as that shown in Figure 2 and the Positive Canonical Graph

[Human] <-(owner) <-[Object]

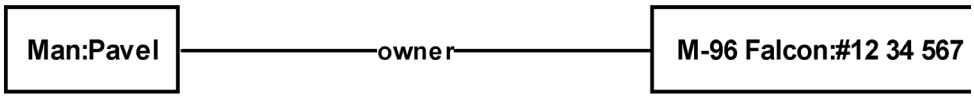


Fig. 2. Link Chart, non iconic, representing Pavel is the owner of an M-96 Falcoln reference 1234567.

where Human is a supertype of Man and Object is a supertype of M 96 Falcoln. It is possible to identify from the set of all possible arc directions generated from the source Link Chart the correct version. This is achieved by projecting

the Canonical Graph onto the generated set. It is possible for certain relations that multiple generated graphs will be matched because of their bidirectional nature which is acceptable. Certain relationships that can exist in either direction, but are not necessarily bidirectional may produce incorrect results. For example a link chart stating:

? [Person:*x] – (fan) – [Person:*y]

and the Positive Canonical Graph

[Person] -> (fan) -> [Person]

could, using the above methodology produce the graph

[Person:*x] –

-> (fan) -> [Person:*y]

< −(fan) <-[Person:*y],.

Although being syntactically correct, it is semantically incorrect. The problem has arisen due to the lack of information in the original Link Chart. By distinguishing relationships that are truly bidirectional from those that are not, it would be possible to identify the missing information and prompt the person supplying the information for clarification. Such identification could be carried out in list form or using the relation hierarchy. By defining a top level relation 'bidirectional relation' we can identify using projection those relationships that are truly bidirectional. To formalize the procedure:

G is the set of graphs of differing link direction generated from the original Link Chart.

P is the set of Positive Canonical Models

N is the set of Negative Canonical Models

Projection(X,Y) is the set of graphs by projecting graph X onto the set of graphs Y.

Let S be the set of graphs from G that are syntactically correct

$$\forall s(s \in S)$$
$$(\exists p, p \in P, s \in \text{Projection}(p, G))$$
$$\neg(\exists n, n \in N, s \in \text{Projection}(n, G))$$

For all s in the set of graphs S it is the case that, there exists a positive canonical graph p from the set of Positive Canonical Graphs P such that s is a graph that matches a positive canonical graph in set G, and it is not the case that there exists a graph n which is a Negative Canonical Graph, and s matches the projection of n on G.

If only one graph exists in this set then it can be accepted (no semantics missing or possible bidirectionality of the relation)

$$acceptedgraph = s$$
$$(s \in S)(1 = \#(S))$$

The accepted graph is an element of S and S contains only one graph.

Let A be the set of graphs from S that are automatically accepted due to true bidirectional relations.

$$\forall a$$
$$a \in S$$
$$\neg \Pr ojection('\text{bidirectional\_relation}', relation(a)) = \{\}$$

For all a in A, a is a syntactically correct graph and the relation in graph a is a bidirectional graph. (It is not the case that the projection of the bidirectional relation on the relation in the graph a results in the empty set.

Finally, our graphs that require further information:

$$S - A$$

We can conclude therefore that it is possible to convert a conceptual graph to a Link Chart easily, and by using Canonical Graphs it is possible to take a Link Chart and convert it directly to a conceptual graph or identify semantics which are missing and are necessary for the conversion.

2.2 Activity and Event Charts

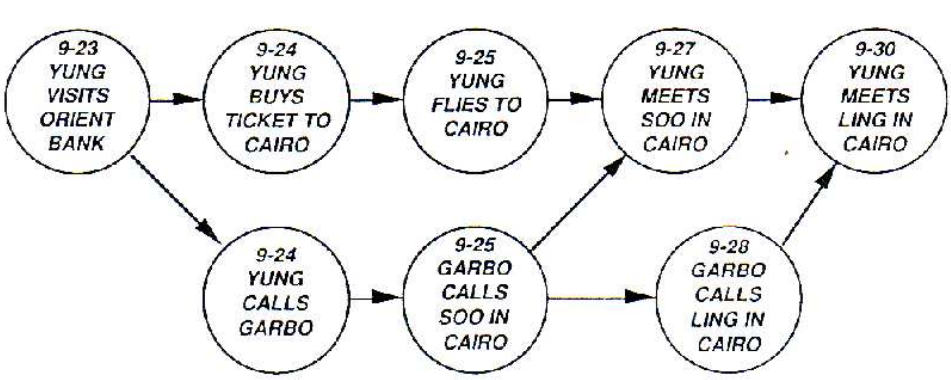


Fig. 3. Example Activity Chart (source: ANACAPA Sciences Training Manual).

These represent a chronologically ordered sequence of occurrences. In the example in Figure 3 each node represents a time, one or more people, some activity and some location. The arcs link these nodes on the basis of a chronological sequence and where a person in the source node exists also in the target node. The activity cannot in the Activity Chart can not be taken as a relationship. Whilst many activities can be represented as relationship e.g.

[Person:Yung] -> (visits) -> [Place:Orient Bank],

others, such as flies indicates that the Activity is in itself a concept and there is some relationship between the person and the concept.

Taking the 25 September Activity of Yung Flying to Cairo we can produce the graph shown in Figure 4.

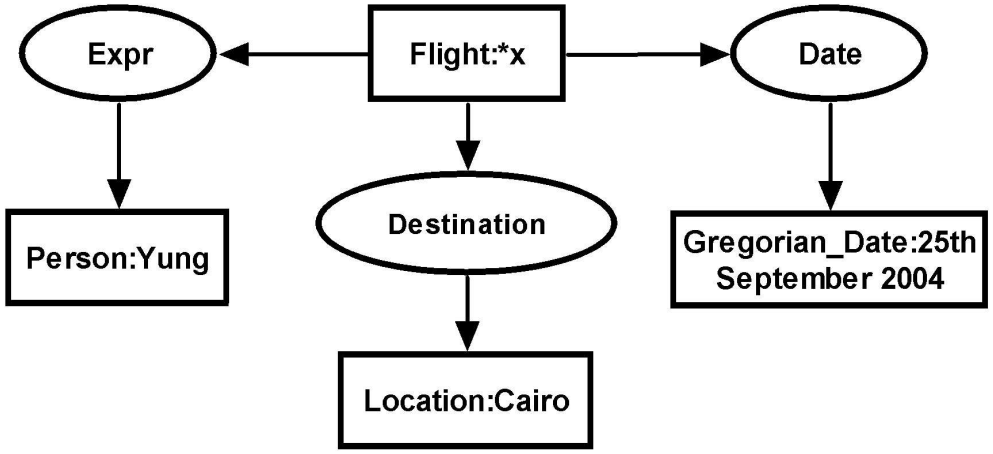


Fig. 4. Person Yung is the experiencer of a flight to Cairo on the 25th September 2004.

It is possible to identify common relations that are implied in an Activity Chart e.g. agent, patient, experiencer, recipient. The next property of the Activity Concept is that of location. A concept may have a single location (as in a meeting), multiple locations (as in a telephone call) or a path (as in a journey). The common characteristic observed in Activity Charts is that, if the location is shown, then it is the Location at the end of the Activity that is shown, the path followed being ignored.

Given that an Activity Chart can be represented in CG format we now examine the mapping of a Conceptual Graph to an Activity Chart. The first stage is to restrict the CG knowledge base to just the people under investigation. This can be achieved with projection, terms of the projection query including the person and the relationships referred to above. However this may produce information that is not relevant e.g.'Yung ordered ham and eggs'. The purpose of CIA is to aid comprehension, consequently it is undesirable to introduce irrelevant information. Unfortunately it is not possible to distinguish which activities are relevant and those that are not. Nor is it possible to order these Activities ('X' is more relevant than 'Y'..) as each is dependant on the situation e.g. 'Chain Smoking' has differing relevancies depending on if the agent is a heavy smoker or infrequent smoker. Consequently the process of deciding which Activities should be shown would require further information from the investigator, as is currently the process when an Activity Chart is drawn directly.

2.3 A Conceptual Graph Criminal Intelligence Knowledge Base

It has been shown that Criminal Intelligence Data can be represented as an underlying representation which can then be accessed to produce CIA representations. This approach offers several advantages.

Rapid change of views. Typically software is used (e.g. I2, Watson) where the user enters information into Link Analysis Software to create a Link Analysis Chart. Using the CG approach information is entered to create a graph that can then be used to create many differing representations.

Time savings. The amount of time to create a CG from Criminal Intelligence Data maybe greater than the amount of time to create a single CIA representation. However, given the overlap of a lot of the information displayed in CIA representations, a CG would save time if multiple CIA representations were subsequently required.

Reasoning. As the Criminal Intelligence Data is now stored logically, it is possible to carry out logical operations such as inference and pattern identification. The ability to introduce reasoning allows a degree of automation to a key stage of the Criminal Intelligence Analysis Process, hypothesis generation.

Validity Checking. Given a set of canonical models it would be possible to check the information that has been entered for syntactic and semantic correctness. The identification of semantic inconsistencies also allows for the early identification of contradictory information indicating that further exploration by the investigators is required.

3 Strength and Validity

Criminal Intelligence data has associated with it two attributes, those of validity and strength. These are applied to each relation in every n-tuple. The validity can be viewed as the likelihood of the relation being true. The allocation of a validity factor is achieved through an approximation of the probability derived from the reliability of the source and what the source believes to be true. Value ranges for this validity factor vary, but typically the values 1 to 99 are used where 0 indicates false and 100 indicates true. It is important to note that normally nothing is considered either true or false in Criminal Intelligence and that the value attributed is an approximation of the underlying unknown probability.

There are several options that are available to the representation of this validity.

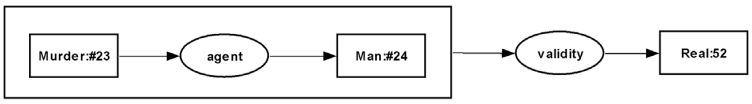


Fig. 5. The validity of the Concept Man No 24 is the agent of Murder No 23, has the value 52.

The purpose of allocating a validity is to allow the investigator to restrict his view of the information by changing the threshold validity. Consequently the need for an approach such as 'fuzzy logic' is removed. The mechanism for achieving this in conceptual graph format is a variation of projection where the values for the Validity concept are above a defined level.

The strength of a relationship is important in criminal investigations. It is often more relevant that the strongest relationship between two entities rather than the shortest path. As with the Validity attribute, the strength of a relationship is given (an approximate) value. Given these it is possible to discover

the Strongest path between two concepts in a graph. The example in Figure 7 illustrates that the strongest relationship between Sam and Claire is through Tom rather than a direct relationship.

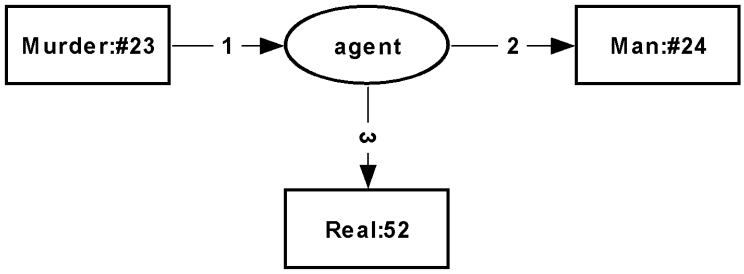


Fig. 6. Man No 24 is the agent of Murder No 23 and the relationship has a validity of 52.

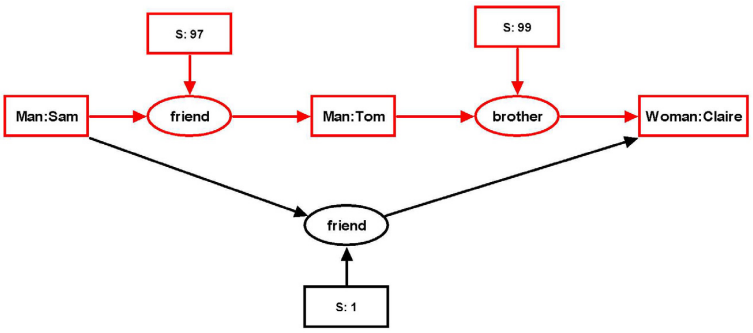


Fig. 7. Example graph showing the relationships of varying strengths.

It may appear on first instance that relationships may have fixed strengths associated with them, e.g. brother. However, brother could be subdivided down into various subtypes, 'brother separated at birth', 'adoptive brother', etc. Each of these would be associated with a different strength. It is possible to create an inheritance hierarchy of these types. However this will generate at least as many types as there are possible values for the strength, and possibly more. Given the purpose is to restrict the view of information being held to a certain strength level and discover the numerically strongest path through the graph, the pragmatic approach of attributing a numerical strength to each relation is the best approach.

The resulting arity of any relation has now increased by two, one for strength and one for validity as shown in Figure 8. Whilst it may appear that this may obscure some understanding of the graph due to its increased number of nodes and arcs, it should be remembered that the CG is anticipated as being an underlying representational schema that would be restricted and viewed in differing formats by the user.

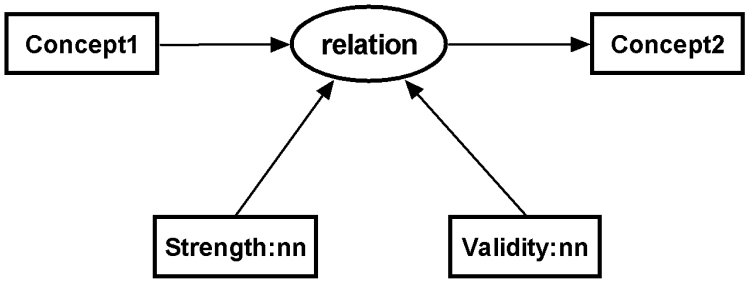


Fig. 8. Representation of Strength and Validity on each relation for Criminal Intelligence Analysis.

The process of projection to retrieve information needs to be expanded to take account of strength and validity. Investigators typically select information above certain strength and validity levels (starting with the strongest / most probable and then expanding the selection if there is insufficient information). For the projection of a graph consisting of a single relation this can be achieved by projecting the whole graphs with varying validity and strengths. For more complex graphs however it is necessary to project the source graph onto the knowledge base regardless of the strength and validity values, then select from the results the ones that satisfy the requirements. This selection would require an evaluation of the overall strength and validity. This evaluation could be achieved through probability theory.

4 Hypothesis Generation

Hypothesis generation is part of the Criminal Intelligence Process, represented in Fig. 9. It is at this stage that the expert skill of the investigator is used to produce a hypothesis that is then used as a basis for further data collection. Most work towards automation of CIA has revolved around data selection and pre-processing.[3]

It has been identified that this hypothesis generation can be divided into two broad types. These categories are conflicting information and the application of models.

4.1 Conflicting Information Identification

The identification of information that is contradictory often provides a clue as to who is responsible for the crime. A characteristic of major investigations is that information is gathered by different people, and given the volume of the information the identification of contradictions are unlikely. Canonical Models can be used to automate this. Only Negative Canonical Models can be used. As an example a NCM stating that something cannot exist in the same place at the same time could be created.

Projection of this NCM onto the graph would identify all things that have two or more locations.

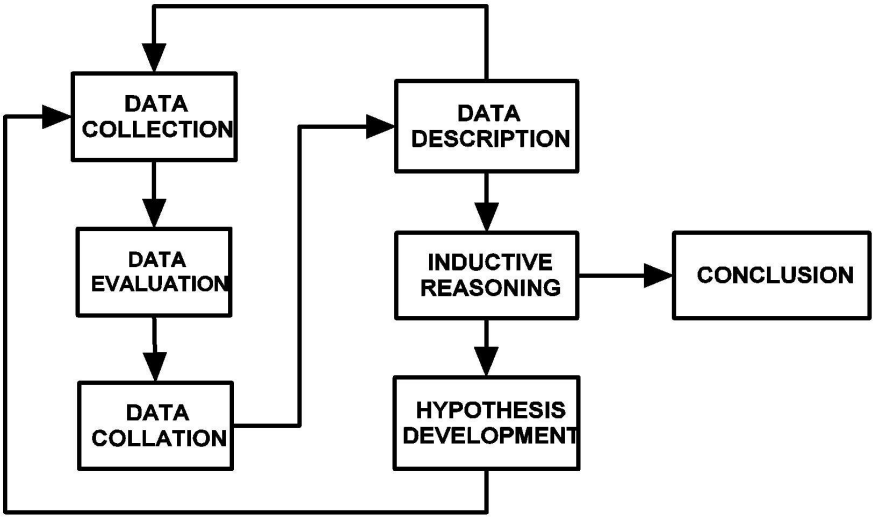


Fig. 9. Stages of the Criminal Intelligence Process (ANACAPA Sciences).

4.2 Pattern Matching

Investigators use their expertise to create hypothesis that fit the facts that they have. These are then tested by further investigation and the collection of further data. Adhami and Browne[4] found that sources of 'investigative expertise' included:

– 'Know-how' developed from personal involvement with previous cases.
– 'Good practice' obtained from operating procedures.
– Knowledge from other cases,
– Insight from academic studies.

All these items share the same thing in common, a model of Criminal Behaviour is being used. The investigator will have numerous models available to him, either is the form of studies or internal models. These models being derived from knowledge gained from past crimes. The choice of which model to apply will depend on two factors. These factors are how well the current case information fits the model and how frequently the model occurs.

Fig. 10. Person #12 is the victim of Murder #9

For example, the information about a crime, Figure 10, fits the models shown in Figure 11 and Figure 12. Assuming that the model suggests a relationship between the victim and agent is more common than the situation where there is no known relationship between the victim and the agent, the model in Figure 11 is used. The result is that the investigators best efforts would be to identify and question people known in some way to the victim. Which models fits the crime

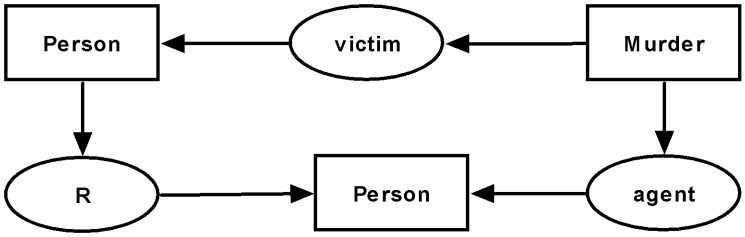


Fig. 11. The person who is the victim of a murder has a relationship with the agent of the murder.

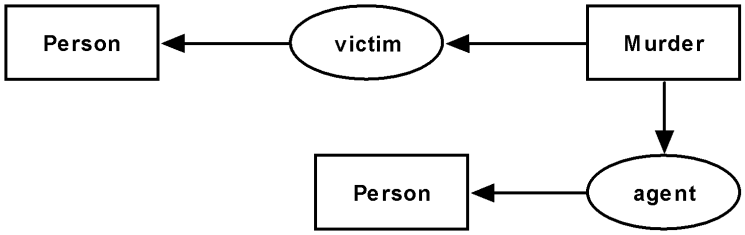


Fig. 12. The victim of a Murder is a person. The agent of the Murder is a Person.

can be identified through the generalization of the known facts, then projecting this new graph onto each model. A successful projection indicates the crime fits the model at least partially.

Having identified that models can be used to assist in directing further enquires the nature of the models being used need to be examined. In order to be of use these models need to be as specific as possible (The model in Figure 11 suffers from the fact that it is too general in that there could be an unmanageable number of possible people that have been in any sort of relationship with the victim).

Three possible options for use as criminological models have been identified, the use of Canonical Graphs, the use of Canonical Models and a new type of graph, which is closely related to the previous two types, Repeating Criminal Occurrence Patterns (RCOPS).

Canonical Graphs[5] could provide a solution in that every instance in the graph could be generalised. However the problem arises when one considers at which generalization is the best. Each generalization of the graph will produce a Canonical Graph, and may increase the number of results of a projection query of the Canonical Graph on the knowledge base. At which point generalizations should cease is unclear.

Likewise a Canonical Model[6] could be used. However it is possible to generalize a Positive Canonical Model and introduce Negative Canonical Models whilst maintaining the same semantics.

Consequently, for the automatic creation of Criminological Models, the concept of Repeating Criminal Occurrence Patterns has been developed.

4.3 Repeating Criminal Occurrence Patterns

These are essentially a special type of Positive Canonical Model such that the necessity for Negative Canonical Models is removed. This is achieved by defining an RCOP as a model where all specializations of the model have instantiations in the criminal knowledge base graph. Using models that are more general than an RCOP may encompass more instantiations of specializations of the graph, however it also introduces specializations that are not capable of instantiations. It is difficult in this scenario to state which is the best generalization. RCOPs do not have this problem. It is possible to subdivide these RCOPs further. RCOPs where all subtypes have instantiations, and those where instantiations only occur on the leaf nodes of the type hierarchy. In practice non leaf nodes tend to be abstract and will therefore not be capable of instantiation, consequently any generalization of a leaf node will immediately be rejected due to the introduction of a combination that does not have an instantiation. It would also be possible to design the type and relation hierarchies to ensure instantiations only occurred at leaf nodes. Effectively the only models that would be created would be derestrictions of graphs rather than generalizations. By only considering the leaf nodes as capable of instantiations we allow generalization of models to the point at which a non instantiated leaf node is introduced.

For example if our Criminal CG had the following subgraphs:

[Murderer:Hodgins] <-(tall).

[Football Hooligan:Jeremy] <-(tall).

[Extortionist:Helen] <-(short).

By using the type hierarchy shown in Figure 13 we could derive the following RCOPs

[Violent Criminal] <-(tall).

[Extortionist] <-(short).

In neither case are the graphs generalized to the type 'Criminal' as this would introduce specializations that are not instantiated.

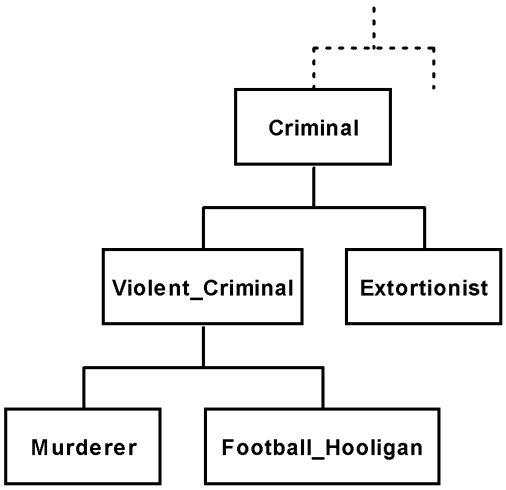


Fig. 13. Partial Inheritance Hierarchy

4.4 Automated Discovery of RCOPs

The automated discovery of RCOPs can be subdivided into those models that start and consequently finish as a fixed shape, and those where the shape changes. The former shall be examined here.

The first stage in the identification of an RCOP is to create a pattern that will be projected against the CG that contains information on past crimes. All the types and relation types should be root types / relations. As we wish to consider crime a RCOP must have at least one concept type that is a subtype of 'Crime'.

This model can then be projected against the main graph and its instantiations retrieved. This graph is then generalized until the RCOP has been found -the point where any generalization would introduce a specialization that has no instantiations in the knowledge base. This process can be repeated until there are no further instances in the graph -all RCOPs have been discovered.

Given a single inheritance hierarchy the process of each generalizing the model is straight forward, the possible number of generalizations at each stage will be the number of nodes in the graph (assuming both types and relations are being generalized). However a combination of multiple inheritance and large hierarchies will significantly increase the number of possible combinations that could occur. Multiple inheritance introduces more than one possible generalization for each node in the graph and therefore increases the number of combinations possible at each stage. The number of types in the hierarchy that would be necessary for the domain of criminal intelligence is unknown, but is probably $> 10^6$ types. Given the size of the hierarchy and the possible combinations of model that can be created a heuristic approach rather than brute force is appropriate. Fortunately, by using heuristics that allow the rapid conversion on the answer, Genetic Algorithms for instance, this can be overcome.[7]

If the past crime knowledge base is partitioned into individual crimes, it is possible, for each RCOP, to identify the number of crimes where the RCOP applies. Consequently the most frequently occurring model can be identified and take precedence over the less likely models. This is to allow the investigator to utilize his limited resources optimally.

5 Architecture Of A Criminal Intelligence Analysis System

A proposed Architecture for an automated Criminal Intelligence Analysis utilizing Conceptual Graphs is shown in Figure 14. The user inputs information and to ensure integrity of the semantics the information entered is checked against Canonical Models for the domain. Assuming the information is correct this is added to the graph containing the details of the current crime. On the completion of each investigation the information gathered on the current crime is added to the Past Crime CG. This CG is a single large CG of all crime rather than multiple smaller graphs. Whilst smaller graphs may prove to be computationally easier to process a single graph is required due to the nature of crime.

Many crimes are related and any partitioning would be arbitrary. This Past Crime graph is used to generate Repeating Criminal Occurrence Patterns. Entered Criminological data is displayed back to the user in a variety of methods by use of the Orthogonal View generator which restricts and displays the information according to the users request. In order to assist the direction of the investigation the 'Hypothesis Generator' identifies possible avenues of further investigation by identifying inconsistencies in the information gathered (through the use of canonical models) and also likely patterns (through RCOPs). These are presented back to the user in order of most probable (based on the number of times these have occurred in past crimes).

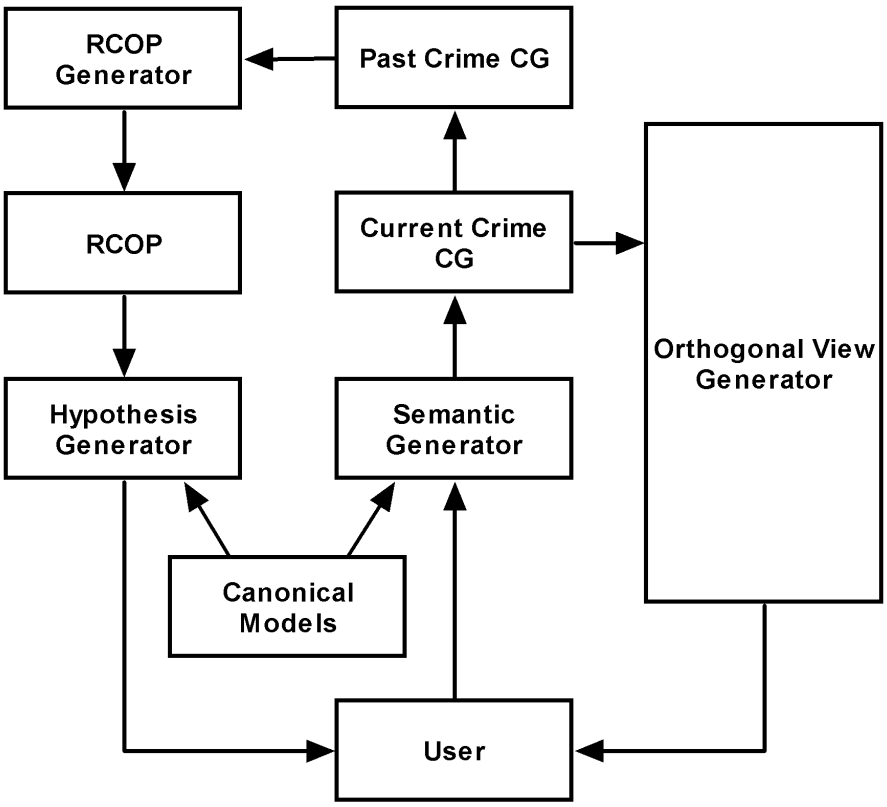


Fig. 14. Architecture Of A Conceptual Graph based Criminal Intelligence Analysis System

Given such an architecture it would be possible to expand the system for the automated processing of information. Discussed so far is an operational approach whereby a crime has been committed and information is gathered relating to that crime. However large amounts of data are collected each day e.g. telephone calls, bank transactions, shop purchases. Entering this information it would be possible to identify possible criminal activities that are occurring. For example the identification of a large group of people all making calls to a certain number and also withdrawing a large amount of cash on the same day could indicate a drug dealer and their customers. However such a system would be hampered by the large amounts of information involved (one small mobile phone operator

in the UK handles 16 billion calls per year). Also issues regarding civil liberties would need to be resolved.

6 Conclusion

It has been shown that many forms of representation used in Criminal Intelligence Analysis can be, with the use of Canonical Models, be translated into Conceptual Graph format. Once in this format it is possible to convert to another representation. Importantly, once the information is held in Conceptual Graph format, it is possible to reason with the information and identify inconsistencies and patterns. This reasoning allows for the assistance in Hypothesis Development which hitherto has been the domain of the Criminal Intelligence Analysis. This approach allows the investigator to allocate resources based on the knowledge of all crime and should, as the system would not be biased by personal experience or bias, enable faster resolution of investigations. Finally the potential ability to constantly scan for and identify possible criminal activity using such a system is identified.

References

1. Sowa, John F.: Knowledge Representation: Logical, Philosophical and Computational Foundations, Brooks/Cole 2000
2. Coffman, T., Greenblatt, S., Marcus, S.:Graph Based Technologies For Intelligence Analysis. Communications of the ACM. March 2004, Vol. 47. No. 3.
3. Dahbur, H., Muscarello, T.: Data Selection and Preprocessing for pattern classifications in criminal databases. Proceedings of the IASTED International Conference Artificial Intelligence and Applications. September 2001.
4. Adhami, E. & Browne, D. (1996). Major crime enquiries: improving expert support for detectives (Paper 9). London: Police Research Group Special Interest Series, UK Home Office.
5. Sowa, John F.: Conceptual Structures: Information Processing in Mind and Machine, Addison-Wesley, 1984.
6. Kocura, P.: Conceptual Graphs and Semantic Constraints. Contributions to ICCS96, Sydney, Australia, pp 133-145.
7. Reed, Rhys N.: Automated Discovery Of Recurring Criminal Occurrence Patterns, In: Contribuions to the 13th International Conference on Conceptual Structures, ICCS'05, Kasel, Germany.

Automated Discovery Of Recurring Criminal Occurence Patterns

Rhys N. Reed and Pavel Kocura

Department of Computer Science
Loughborough University
Loughborough, Leicestershire. LE11 3TU
England
R.N.Reed@lboro.ac.uk P.Kocura@lboro.ac.uk

Abstract. This purpose of this paper is to show how Genetic Algorithms can be used to improve the automated discovery of Repeating Criminal Occurrence Patterns, a specialised form of Canonical Graph specific to Criminal Intelligence Analysis. Approaches to mutation and cross over functions using conceptual graphs and type hierarchies are described and how a rapid approach to pattern identification can be achieved.

1 Introduction

Modern Criminal Investigation involves the identification of patterns of criminal behaviour. Traditionally these patterns are arrived at from personal experience of the investigator and statistical studies [7]. However this approach suffers from the problems of personal bias, lack of experience of the investigator and a limited number of studies. Although work has been done on the automation of representation [9], little work has been done on the automated identification of criminal behaviour patters.

Given Criminological Information in Conceptual Graph[5] format it would be possible to automate the identification of repeating patterns of a knowledge base of a crime. These could then be used to give assistance to the investigator which does not suffer from personal bias. Recurring Criminal Occurence Patterns (RCOPs) have been identified as a model that will provide such assistance [10]. An RCOP is a specialized type of Canonical Graph [6] such that it is of predefined shape and the most generalized form of a Canonical Graph where all possible specializations of the graph have instantiations in the knowledge base (removing the need for negative canonical models). One further characteristic of an RCOP is that it relates to crime - at least one concept type is of the type crime. The automated discovery of RCOPs is possible through various techniques although there are time and complexity issues. Genetic Algorithms (GA) provide a possible solution. An approach using GAs with CGs is shown together with details of an implementation that demostrates the use of this methodology.

2 Problem

2.1 Brute Force

It is possible, in order to find RCOPs, to create every possible combination of a graph of a specific pattern using the type and relation hierarchies. These can then tested to see if each possible specialization has an instantiation in the knowledge base. However this rapidly proves to be computational expensive because of a combinatorial increase. For example the pattern shown in Figure 1 and type / relation hierarchies of 2^{14} nodes, the number of possible graphs that can be created is 2^{84} (for a graph of n vertices and a hierarchy of m types the number of combinations is n^m). This is not practicable for anything other than trivial hierarchies and another approach is required.

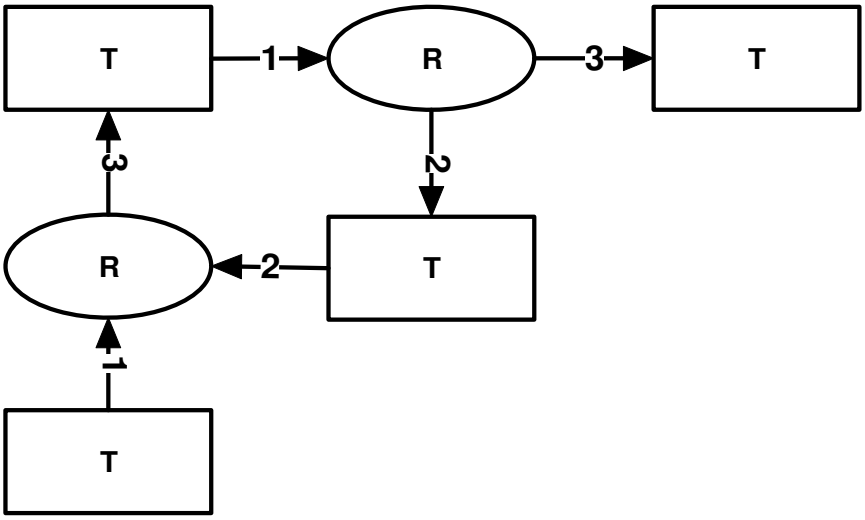


Fig. 1. Example graph used for the generation of RCOPs

2.2 Hill Climbing

Given a small graph with limited type and relation hierarchies which have single inheritance it is possible to find a RCOP. This is achieved by generalizing the concept and relation types of a derestriction of a subgraph from the knowledge base of the required shape. Generalization of each type and relation is carried out until a possible specializations of the test graph lack instantiations is introduced. At any stage the number of possible generalizations will be equivalent to the number of nodes in the graph. Given a small inheritance hierarchies the RCOP will be found in a relatively few steps.

The problem arises however with multiple inheritance. This introduces multiple possible generalizations per node. The worst case situation is one where the inheritance hierarchy tree is fully inverted. The problem also increases with the size of the inheritance hierarchies as many intermediate nodes need to be

traversed before the desired node is reached. Hierarchies for Criminal Intelligence Analysis, which can encompass many domains (investigators often use specialists) and may be estimated at over 2^{20} nodes. An approach that avoids a brute force approach and the computational difficulties introduced by multiple inheritance would therefore be beneficial to the automated discovery of RCOPs.

3 Genetic Algorithms

Genetic Algorithms offer a potential solution that is essentially a hill climbing algorithm that enables rapid convergence on a solution. They have been used for a variety of purposes including pattern matching [2]. However this approach does not garantee convergence on the correct solution. The probability of this occuring was tested and is discussed later. This approach requires the data to be represented as chromosomes, have cross-over, mutation and fitness functions. An overview of the process is shown in Figure 2.

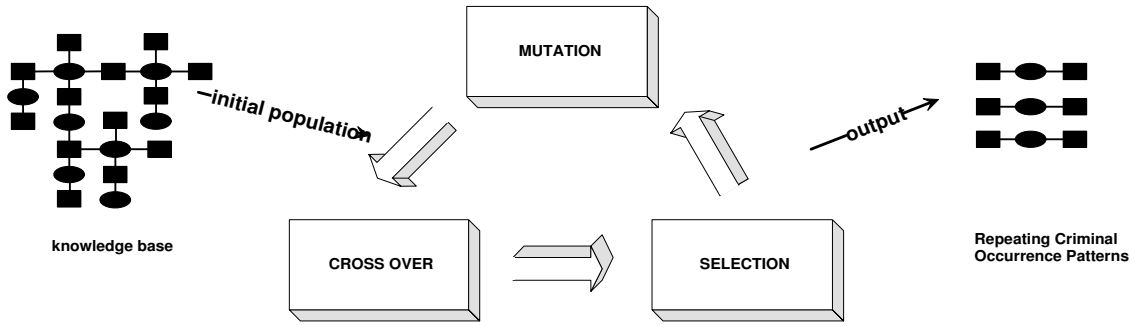


Fig. 2. Genetic Algorithm approach to RCOP discovery

3.1 Representation

The first stage in using a genetic algorithm approach is to represent the information as 'chromosomes' [1] . This is typically achieved using character strings or numbers representing different types e.g. values and operators in the case of mathematical problems.Conceptual Graphs lend themselves to creating two categories of chromosome – concept type chromosomes and relation type chromosomes. The later category requires subdivision into relation types of differing arity. This is to maintain the same structure of the pattern being sort – it is not possible to exchange relations of differing arity.

3.2 Initial Population

For a GA approach an initial population is required. This intital population is used later in the cross over and selection process. The initial population of graphs can be obtained in two ways. The graphs can be generated at random or

can be derived from the knowledge base. Generating the source population from the knowledge base has the advantage that the graphs can be selected such that they are restrictions of the (unknown) RCOP. With a large hierarchy it is likely that the randomly created graphs would be so far distant from the solution that a large number of mutations, and consequently generations would be required to produce graphs that are as close as graphs drawn from the knowledge base. If the size of the graph knowledge base is small it is relatively straight forward to carry out a projection query using the root relation and types and selecting graphs at random from this set. These graphs can be derestricted and then added to the initial pool. Selections are made in this manner until the initial pool is at the desired size. The selection at random from a complete set of results produced from a large knowledge base could require significant computation. In this eventuality the complete random selection of graphs could be utilized. The trade off however is that the number of generations would have to be increased and / or the fitness function changed.

3.3 Cross-over

Cross-over is the process whereby two parents are taken and their chromosomes are used to create off-spring. An example of the potential offspring of a pair of parents is shown in Figure 2.

The selection of parents for breeding can be done in many ways e.g. roulette wheel selection, tournament selection. (reference here) However a simple method, which proved effective in trials was to have the probability of selection Pselection = k.position^n. Using negative values for n effectively biases the selection of parents to those with a higher fitness function. Fig. 3 shows the results indicating that the lower the value of n the faster the population converges on the correct result. This graph show that with no bias the proportion of the population that matches the underlying rule slowly increases. However with a bias on the selection of parents in favour of those with a higher fitness function the population matching the underlying rule increases significantly faster. In this particular instance about 4 times more rapidly than the unbiased selection.

3.4 Mutation

The chromosomes representing the relation and concepts need to be mutated. There are two methods of doing this. The first is to randomly change the chromosome in question. The second is to change the chromosome in accordance with some rules. The first option was rejected as a random change to the chromosome would in all probability produce a chromosome that did not exist in the type/relation hierarchy and would therefore produce no results and be rejected immediately. The later option is therefore preferential as the mutation can be restricted to a chromosome that can potentially exist. Furthermore if we consider that the best mutation is likely to be in the vicinity (branch of the tree) of the current node rather than at a completely random location it is sensible to restrict how mutations occur.

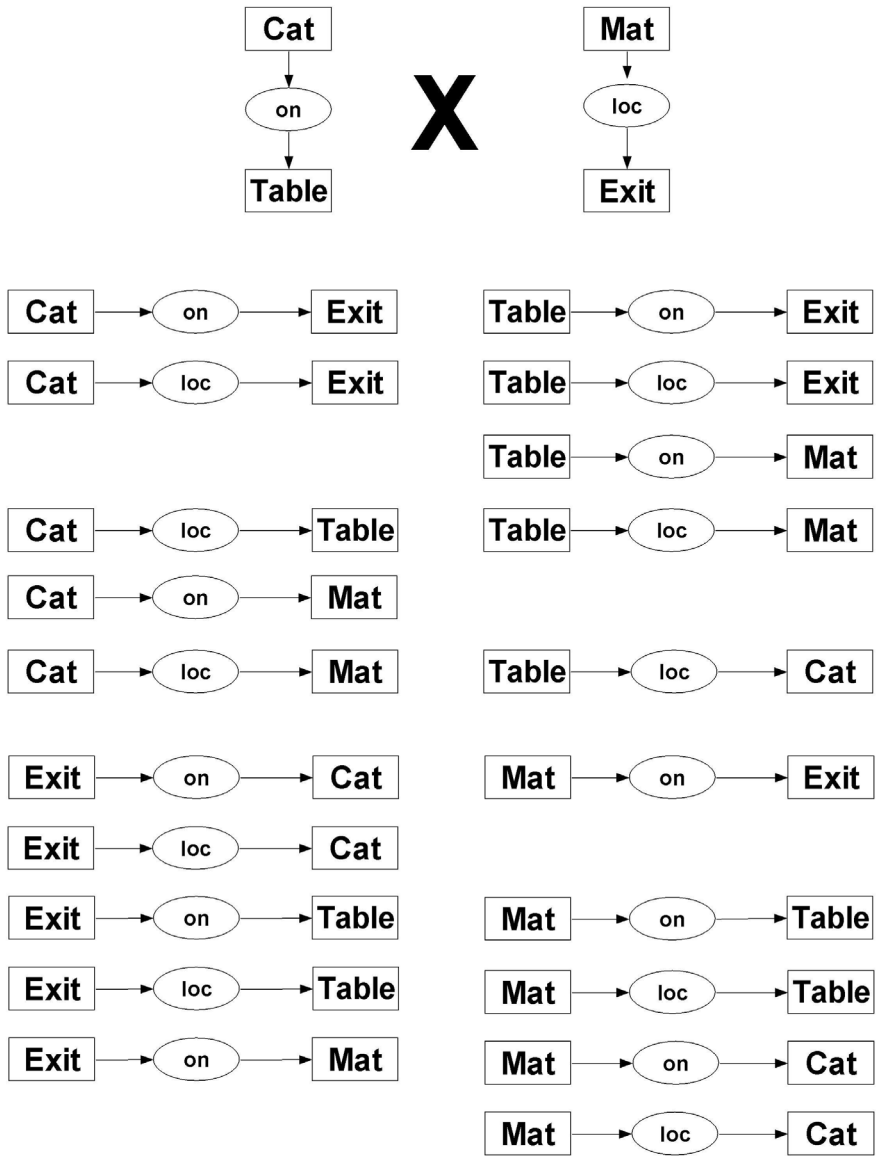


Fig. 3. Example of possible child graphs from two parents where the intial two graphs produce an offspring population of 20 different results

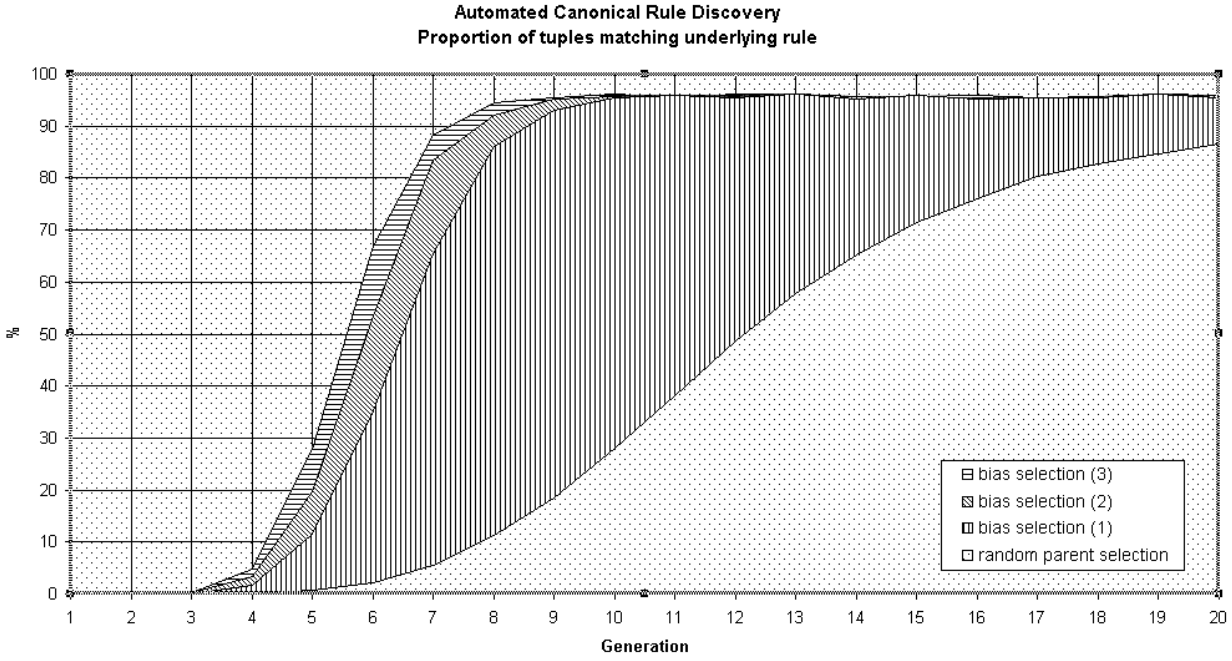


Fig. 4. Effects of increasing bias on selection of parents for breeding on convergence on the correct canonical rule. Graph demonstrates that increasing the bias towards selection of 'fitter' parents produces faster convergence on the overall result.

This is achievable by introducing the concept of Maximum Hierarchy Mutation Distance. This restricts the degree of mutation by stating how far on the inheritance hierarchy can be travelled. As each step in the traversal could be in either direction (towards the root or away from the root) this approach produces a distribution of mutations with P(max) being the nodes closest the mutated node. A hierarchy illustrating the mutation distance from a single node is shown in Fig. 5.

In deciding the maximum mutation distance the governing factor is the anticipated distance from the leaf nodes to the underlying rule. If the mutation distance is set to 1 then each intermediate node has to be traversed. If however the mutation distance greatly exceeds the distance in the hierarchy then the mutation may over shoot. In practice it was found that a maximum distance equivalent to the overall hierarchy height provided good results in a test environment. However this is an area not fully explored and would be worthy of further research. Not tested, but would aid rapid convergence on the answer, is the decreasing of the mutation distance as the number of generations increases. This would allow for rapid convergence as initial generations would allow for a large mutation, but later generations would be prevented from similar large mutations away from the goal.

3.5 Fitness Function

In order to determine which graphs to breed from it is necessary to quantify how good a particular graph is in relation to another. For the following it will

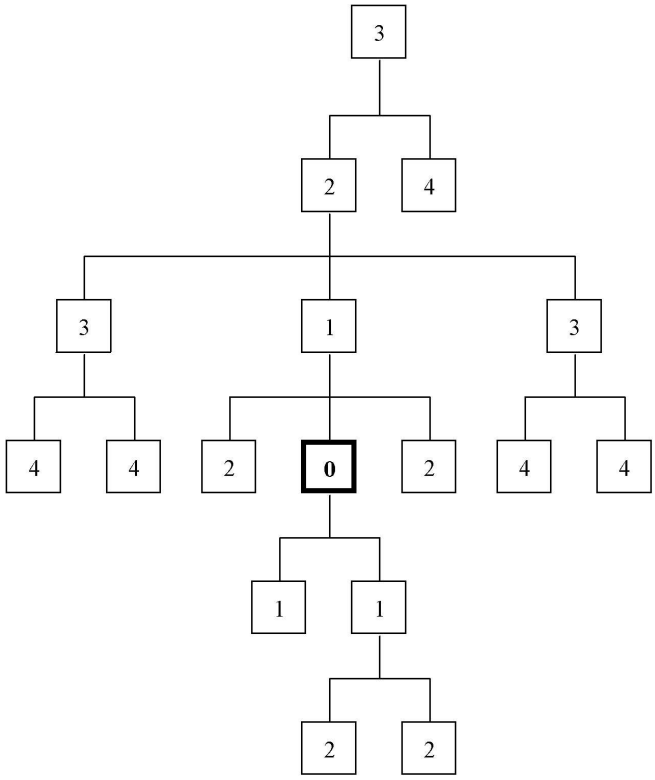


Fig. 5. Example Mutation Distance for Inheritance Hierachy.

be assumed for simplicity that only leaf types and relations are capable of instantiation. There are two characteristics of potential graphs that can be used in deciding which graphs are closer to being an RCOP which are demonstrated in Figure 6. The first is the number of possible specializations capable of instantiation. The second is the actual number of specializations that have instantiations. If the graph in question is more general than the RCOP then its number of possible specializations will be greater than the actual specializations. If however the graph is more specialized than the RCOP, then both the possible and actual number of specializations will be the same. It is therefore possible to distinquish between RCOP and non-RCOP graphs. The next stage is to order the graphs in accordance with how general they are. For all RCOP graphs this can be achieved by ordering them in accordance with the number of specializations. The greater the number of specializations the greater the degree of generalization. By combining the above two attributes it is possible to compare any given graphs.

To demonstrate the above concepts an example shown in Figure 6 andTable 1. The graphs in question have not used an inheritance hierarchy on the relation for clarity in the diagram.

Shown in the diagram are four potential graphs that are being tested. It is shown in the table that by using the actual number of specializations and the possible number of specializations it is possible to determine not only the graphs that are types of RCOP (graphs 2 and 3) but also which is the best RCOP graph (graph 2 as it has a greater number of specializations).

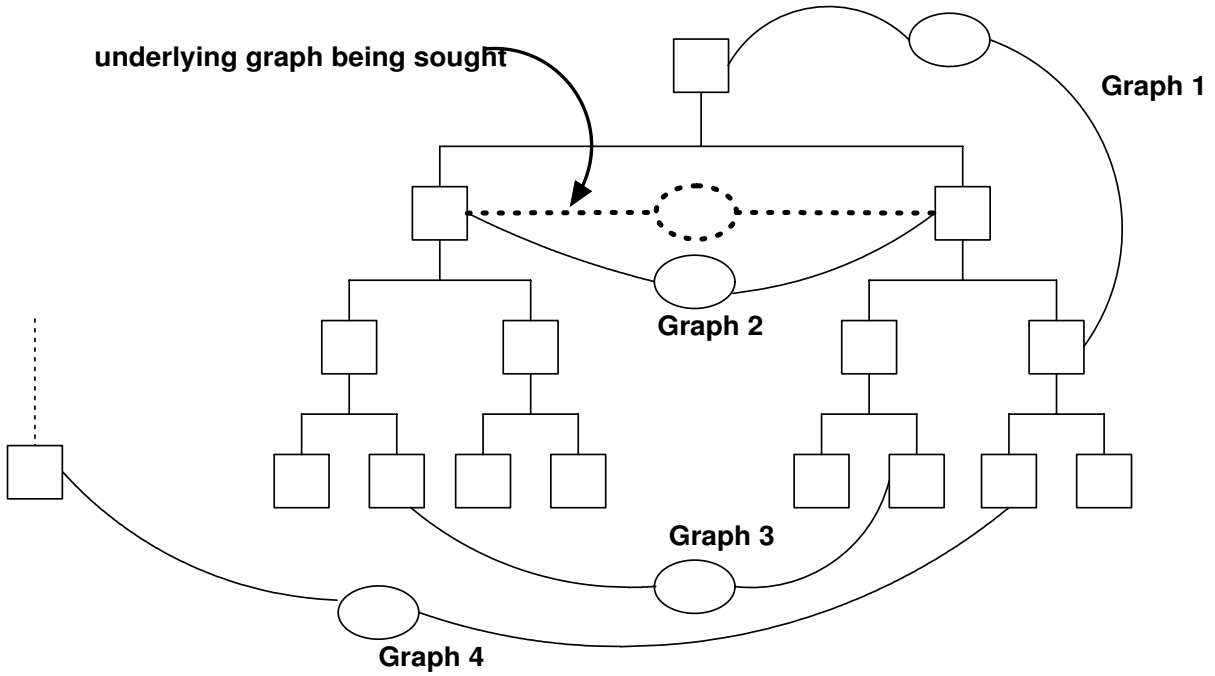


Fig. 6. Detecting Restricted Canonical Graphs on a type hierachy.

Table 1. Properties for test graphs and their fitness functions

	Possible Specializations	**Actual Specializations**	**RCOP ?**	**Example Fitness Function value**	**Comments**
Graph 1	16	8	no	0.08	Parent(s) too high.
Graph 2	16	16	yes	1.16	RCOP (and max specializations)
Graph 3	1	1	yes	1.01	RCOP
Graph 4			no	0	Mutated outside of possible RCOPs

Finally, as patterns relating to criminal behaviour are being sort, those graphs that contain a type or subtype of crime need to be given preference. In order to achieve this the set of graphs can be partitioned into those relating to crime and those that don't. Those that relate to crime are given higher preference to those that do not.

3.6 Convergence

There are two frequently used methods of determining when an answer has been arrived at. These are checking the population for the dominance of one particular answer or using the best answer after a certain number of generations. It was decided for ease of coding that the later method would be used and that the optimum number of generations would be derived from test data.

4 Implementation

A system was constructed according to the architecture shown in Figure 7. The projection engine used an adapted multi-level hierarchical retrieval mechanism [8] that enabled the return of a pointer to a list of subgraphs in logarithmic time (with respect to the type and relation hierarchies). As the GA approach is only concerned as to the existence of an instance satisfying the projection query the performance becomes independent of the number of instances and consequently the knowledge base size. Performance is also increased with the introduction of a projection cache. As with each generation the number of identical items in the breeding population increases, the number of identical projection queries also increase. By caching previous queries the need to rerun these against the knowledge base is removed.

Into the system a type hierarchy of 2^{16} and a relation hierarchy of 2^{14} were loaded. An arbitrary canonical rule was then chosen and from this a non-disjoint graph consisting of instantiations of all possible specializations was created.

The first operation was to confirm that a statistically reliable guarantee of converging on a correct answer was possible. This was achieved by running the algorithm multiple times and recording at which generation the highest rated graph became the target answer. It was found that whilst over 50% of the tests achieved the correct answer in under 5 generations, it was 19 generations to assume a 99.997% success rate. By increasing the number of generations to 25 the projected robability of an incorrect answer drops to $1{:}10^9$. Running on a 2.4G PC each generation took on average 29ms for the test data detailed, increasing the certainty of the results by increasing the number of generations is therefore a feasible option . The overall time to select the initial data set and run 25 generations was 775ms.

Given a fixed number of generations of breeding and mutating, θ for the discovery of a RCOP remains logarithmic with respect to the number of types in the type hierarchy and independent of the size of the knowledge base. In order to confirm this performance the time for finding a RCOP was recorded

for differing numbers of instances in the conceptual graph knowledge base. The results shown in Figure 9 however demonstrate that there is in fact a linear increase albeit small. It is suggested that this maybe due to the underlying operating system rather than the performance of the algorithm.

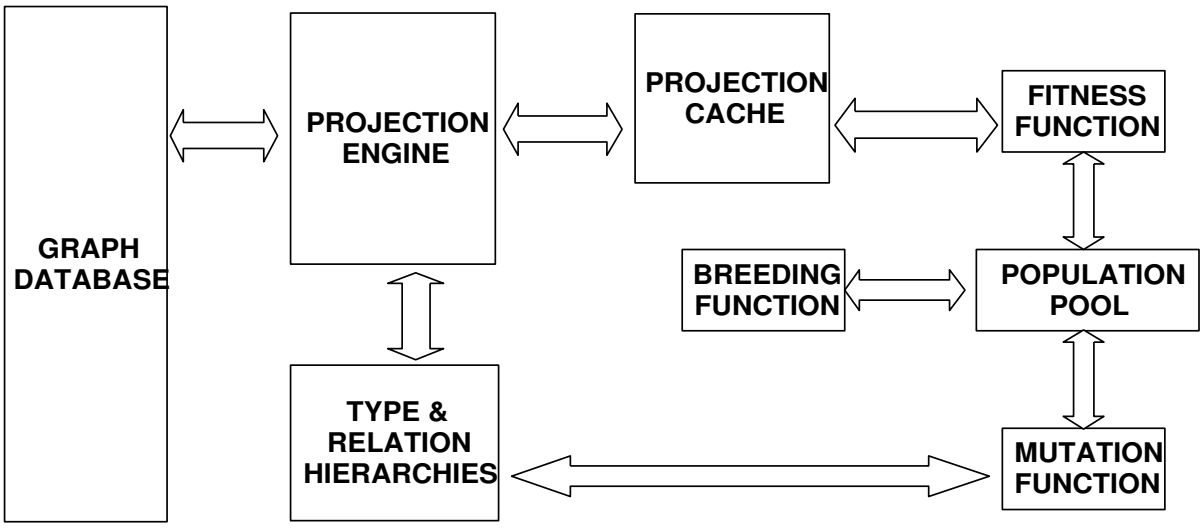


Fig. 7. Architecture for genetic algorithm approach for Canonical Graph Discovery

4.1 Multiple rule Recovery

Multiple Rule Recovery from a Conceptual Graph Knowledge Base is, once a single rule can be recovered, a relatively straight forward matter. Once a RCOP has been discovered a projection query of this graph against the knowledge base will produce a set of all matching graphs. These can then be retracted from the knowledge base. The process of finding the next RCOP can then be repeated. This process has the characteristic that the knowledge base graph becomes disjoint, however this does not effect the performance of the projection algorithm. It was found, for efficiency, that the best approach was to delete the pointers to the list of results rather than removing the items in the list. This enabled the deletion operation to perform logarithmically.

In order to test multiple RCOP discovery the above methodology was adapted to include retraction and a non-disjoint graph consisting of approximately 26M nodes was created from 4 models in a similar fashion to the single RCOP test. The output of the system is shown in Figure 8.

Given a single run of the GA is required to identify a RCOP, the performance for multiple rule recovery is linear with respect to the number of underlying rules.

$$\mathbf{n}_{rcops}\mathbf{log(h)}$$

Where n_{rcops} is the number of RCOPs in the knowledge base and h is the number of types in the type hierarchy.

```
>count project (R) [T] [T]
Returned a total of 26,071,040 results

>allrules
Rule Found (RA) [TA] [T]  (12,451,840 matches in database)
Rule Found (RC) [TC] [T]  (6,225,920 matches in database)
Rule Found (RD) [TD] [T]  (5,836,800 matches in database)
Rule Found (RB) [TB] [T]  (1,556,480 matches in database)
Run time: 6.078000 seconds
>
```

Fig. 8. Output of multiple restricted canonical graph discovery program.

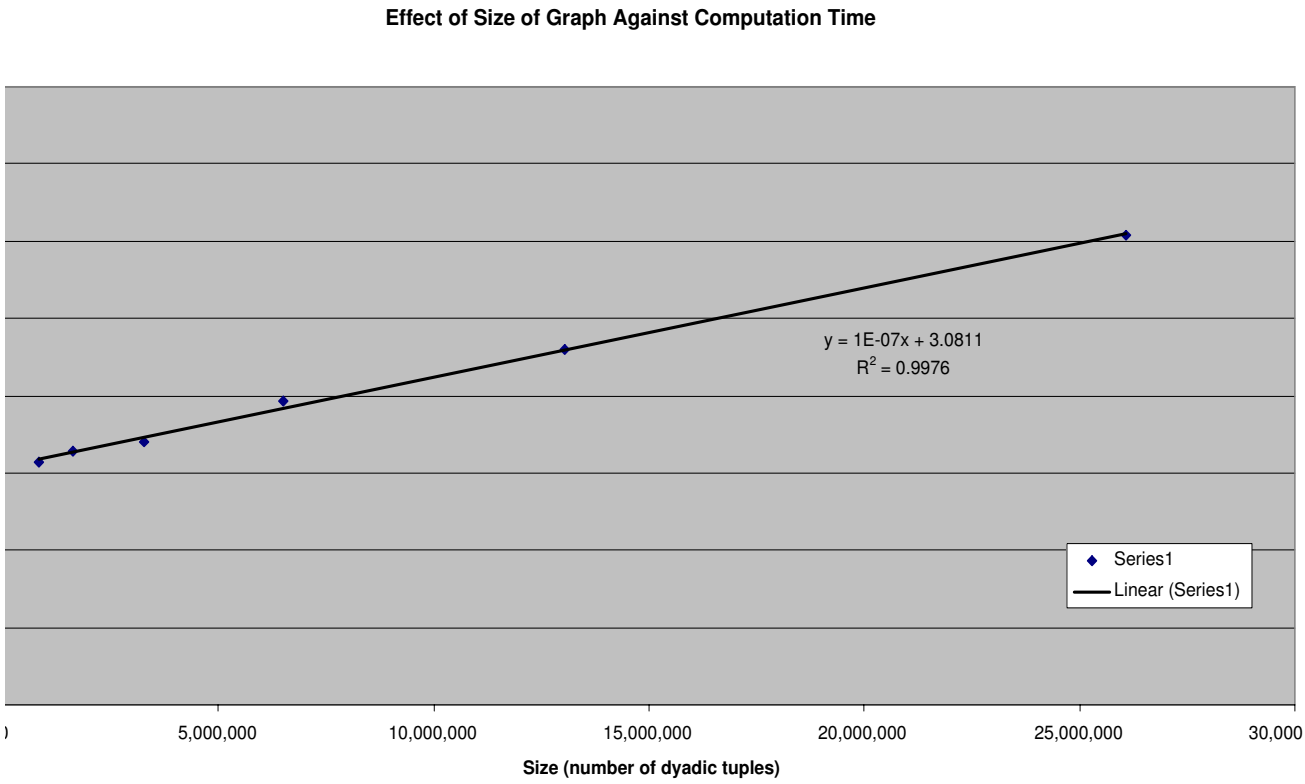


Fig. 9. Effect of knowledge base size on time to recover Restricted Canonical Graphs.

5 Conclusion

It has been attempted to show that Genetic Algorithms provide a solution to the automated discovery of Recurring Criminal Occurrence Patterns. In particular, we have analysed the representation of conceptual graphs as chromosomes and how the concept of maximum mutation distance can assist in the rapid convergence on a solution. The issue of GA's not guaranteeing to converge on the correct answer has been addressed and the likelihood of an incorrect answer proved to be negligible by increasing the number of iterations of the algorithm. Of significance is that the process of automated RCOP discovery is independent of the graph size and is governed by the number of RCOPs and the size of the inheritance hierarchies. This project is by no means finished: in particular, the ability to identify RCOPs of varying shape will require further intensive study. It appears that an Evolutionary Algorithm would be best suited to this problem. However the ability to automatically discover patterns in criminal behaviour from past crimes is a significant advance on existing methodologies.

References

1. Rothlauf, Franz.: Representations for genetic and evolutionary algorithms. New York : Physica-Verlag Heidelberg, 2002.
2. Pal,Sankar,. Wang, Paul.: Genetic algorithms for pattern recognition Boca Raton, Fla. ; London : CRC, 1996
3. Goldberg, David E.: Genetic algorithms in search, optimization, and machine learning. Reading, Mass. ; Wokingham : Addison-Wesley, 1989.
4. Sowa, John F.: Knowledge Representation: Logical, Philosophical and Computational Foundations, Brooks/Cole 2000
5. Sowa, John F.: Conceptual Structures: Information Processing in Mind and Machine, Addison-Wesley, 1984.
6. Kocura, P.: Conceptual Graphs and Semantic Constraints. Contributions to ICCS96, Sydney, Australia, pp 133-145.
7. Adhami, E., Browne, D.: Major crime enquiries: improving expert support for detectives (Paper 9). London: Police Research Group Special Interest Series, UK Home Office. 1996.
8. Levinson, R. A., Ellis, G.: Multi-level hierarchical retrieval. Knowledge Based Systems, pages 233-244, 1992.
9. Harris, D. H.: Development of a computer-based program for criminal intelligence analysis. Human Factors, 20(1):47-56, 1978.
10. Reed, R. N., Kocura, P.: Conceptual Graph based Criminal Intelligence Analysis. Contributions to ICCS 2005, Kassel, Germany.

A Multilingual Information Retrieval System based on Conceptual Graph Structure

Catherine Roussey*, Sylvie Calabretto**

*LIRIS UMR 5205 , Claude Bernard University, Bâtiment Nautibus
43, Boulevard du 11 novembre 1918
F-69622 Villeurbanne Cedex
** LIRIS UMR 5205 , INSA of Lyon, Bâtiment Blaise Pascal
7, avenue Jean Capelle
F-69621 Villeurbanne Cedex
{firstname.lastname}@liris.cnrs.fr

Abstract. In this paper, a graph formalism is proposed to describe the semantics of documents in a multilingual context. This formalism called semantic graph is an extension of the conceptual graphs of Sowa. Semantic graphs are based on a semantic thesaurus which stores two kind of knowledge: domain knowledge and lexical knowledge. The goal of semantic thesaurus is to share a common semantic for different languages. Thus, semantic graphs enable to develop several useful functionalities for a multilingual system. For example, users can compose their queries using a thesaurus written in their language. Resulting documents can be adapted to the user language etc…. All these functionalities are implemented in our multilingual documentary system, called SyDoM.

Keywords. information retrieval, conceptual structure, conceptual graph, multilingual context, multilingual information retrieval system.

1 Introduction

Our work deals with conceptual structures that describe document contents in a multilingual documents system dedicated to digital libraries. In digital libraries, retrieval systems based on automatic indexing or those based on manual indexing using thesauri never satisfied completely users. The former is considered inadequate because it deals with out-of-context terms. The latter is hard to manage by the indexing team since it needs periodic updates of both thesauri and indices in order to reflect scientific progress. Our aim is to propose document system based on manual indexing that allows taking into account the vocabulary evolution. Manual indexing is often criticized because of its cost and the indices variability. Our motivation is that manual indexing method is more exact than the automatic one. One can argue that automatic indexing is more regular because it supplies always the same index for the same document. This constitutes a quality of the system but this property is different

to exactness. Indeed, automatic indexing can't interpret a text and can't adapt itself to new vocabulary. For example, if the system doesn't have knowledge for removing term ambiguities, it will generate errors in interpretation of meaning and this will carry incoherencies in the index base.
Traditionally, manual indexing tries to find keywords to represent the document content. To take the multilingual aspect of collections into account, it is necessary to improve the representation of documents. In multilingual context, terms are no more sufficient to express the document contents. It is thus necessary to work on elements more significant than terms, namely "concepts". Moreover, according to [2,3], semantics of indices have to be enhanced. A solution is to transform the usual list of keywords into a more complex indexing structure, in which relations link concepts. That is the reason why the Sowa formalism of **Conceptual Graph** (CG) [7] is chosen to express the document contents. Nevertheless, one of the drawbacks of IR systems based on CG is the time consuming effort to carry on a retrieval process. Moreover, the CG matching function produce lot of silence[1] that decrease the recall rate. In this article, Matching functions and algorithms are proposed in order to improve the retrieval effectiveness of our system. We also proposed an adaptation of the CG formalism able to deal with multilingual representation. This model is called semantic graph.

First of all, we describe the semantic graph model and its corresponding matching function. Then, the SyDoM prototype based on semantic graphs is presented. Finally, we will present the evaluation of our proposition.

2 Semantic Graph Model

We have simplified the CG formalism to get it closer to a documentary language. So, concepts are limited to generic concepts because descriptors represent main notions and not individual object. Moreover, the comparison between graphs should be based only on semantic similarity, the graph structure is less important.

2.1 Semantic thesaurus

We proposed an extension of the Sowa formalism in which two kinds of knowledge are identified in a semantic thesaurus:

1. Domain knowledge organizes domain entity in two hierarchies of types. Types defined a pivot language used to represent document and query graphs
2. Lexical knowledge associates terms, belonging to a vocabulary, to types. Terms are used to present semantic graphs in the user's native language.

2.1.1 Domain conceptualization or Support

A support S is a 2-tuple $S = (T_C, T_R)$ such as:

[1] Relevant documents not retrieved (forgotten) by the IR system..

- T_C is a set of concept types partially ordered by the specialization relation, noted $\leq$, and it has a greatest element, noted *T*.
- T_R *is a* set of binary relation types[2] partially ordered by $\leq$ and it has a greatest element, noted T_2.

2.1.2 Semantic thesaurus

A semantic thesaurus, noted *M*, (composed of *P* languages) is a 3-tuple $M = (S, V, \lambda)$ such as :

- *S* is a support (cf. § 2.1.1).
- *V* is a set of vocabularies, split into set of terms belonging to the same language (a vocabulary). $V = V_{L1} \cup V_{L2} \cup ... \cup V_{Lj} \cup ... \cup V_{LP}$ such as V_{Lj} is a set of terms belonging to the language *Lj*.
- $\lambda = \{ \lambda^{VL1}, ..., \lambda^{VLj}, ..., \lambda^{VLP} \}$ is a set of *P* mapping such as $\lambda^{VLj} : T_C \cup T_R \rightarrow V_{Lj}$ is a mapping $\lambda^{VLj}(t)$ which associates a term of the language $Lj \in V_{Lj}$ with a type $t \in T_C \cup T_R$.

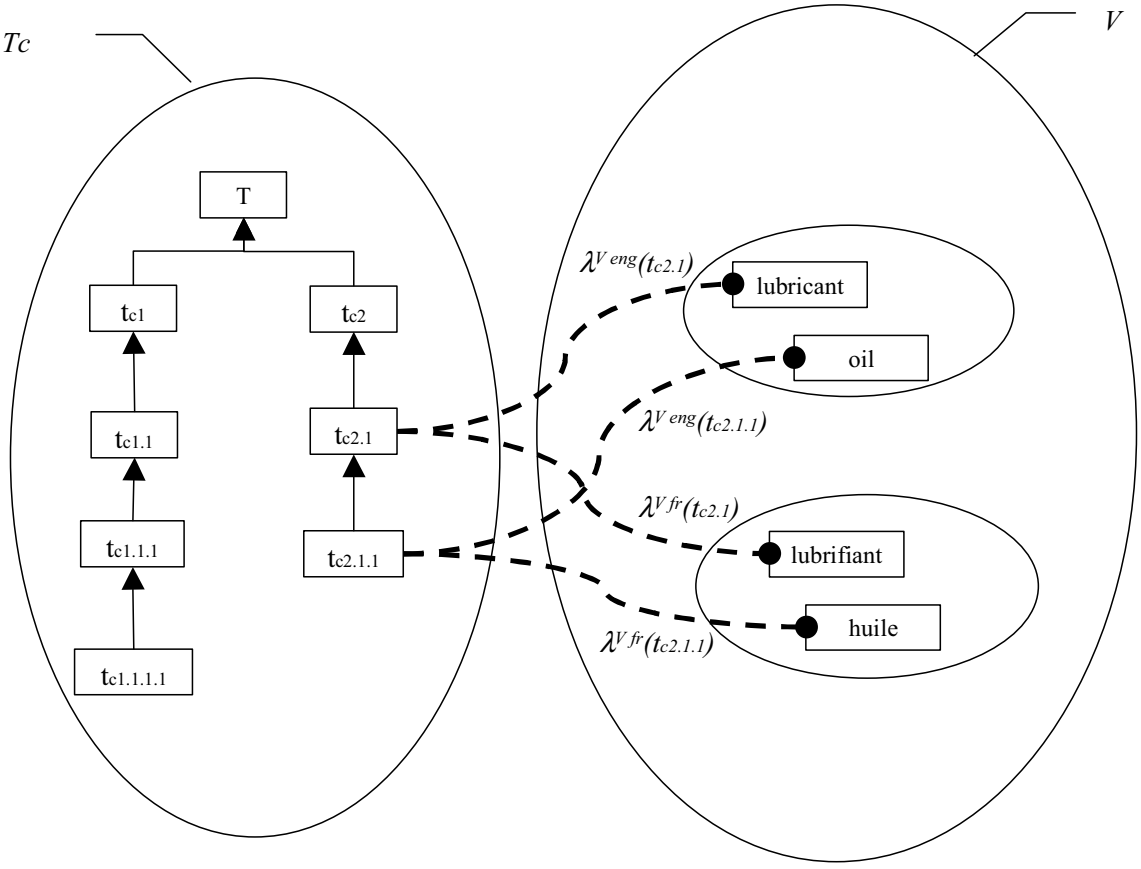


Fig. 1. An example of semantic thesaurus.

Figure 2 presents an example of mapping λ, in which *V* is composed of two vocabularies: an English vocabulary, noted V_{eng}, and a French vocabulary, noted V_{fr}. Each concept type is linked to a term of each vocabulary. For example, the concept type $t_{c2.1}$ is linked to the English term $\lambda^{Veng}(t_{c2.1})$*="lubricant"* and it is also linked to a French term $\lambda^{Vfr}(t_{c2.1})$*="lubrifiant"*.

[2] In general, a type of relation can have any arity, but in this paper, relations are considered to be only binary relations like case relations or thematic roles associated with verbs [8].

From this semantic thesaurus defining domain knowledge and lexical knowledge, our formalism, called **semantic graph**, is defined. A semantic graph is a set of concept nodes connected to each other by relations. Comparing to Conceptual Graph, the notion of **arch** is defined as a couple of concept nodes labeled by a relation type.

2.2 Semantic graph

A semantic graph is a 4-tuple $Gs = (C, A, \mu, \nu)$ related to a semantic thesaurus *M,* such that :

- C is a set of concept nodes[3] contained in *Gs*.
- $A \subset C \times C$ is a set of arches contained in *Gs.*
- μ: $C \rightarrow T_C$, $A \rightarrow T_R$, μ is a mapping, which associated for each concept node, $c \in C$, a label $\mu(c) \in T_C$, $\mu(c)$ is also called the ***type*** of *c*. μ associated for each arch, $a \in A$, a label $\mu(a) \in T_R$. $\mu(a)$ is also called the ***type*** of *a.*
- ν is a set of mapping $\nu = \{\nu^{VL1}, \ldots \nu^{VLj}, \ldots, \nu^{VLP}\}$ such that the mapping ν^{VLj}: $C \cup A \rightarrow V_{Lj}$ associates an arch, $a \in A$ or a concept node, $c \in C$, with a term of the language *Lj*. $\nu(a)^{VLj} \in V_{Lj}$ is called the ***term*** of *a* for the language *Lj*. $\nu(c)^{VLj} \in V_{Lj}$ is called the ***term*** of *c* for the language *Lj*.

Thanks to previous definitions, there exist different representations for the same semantic graph depending of the label used.

1. The first representation of semantic graph labels each graph component with its type. So for a concept node *c,* its label is $\mu(c)$.
2. The second kind of semantic graph representation labels each graph component with a term chose in a vocabulary defined in the semantic thesaurus. So for a concept node *c,* its label is $\nu(c) = \lambda(\mu(c))$. Indeed, there exist several representations of the same semantic graph depending of the chosen vocabulary.

Now, the pseudo-projection operator comparing semantic graph is presented.

2.2.1 Pseudo-projection operator

The pseudo projection operator is an extension of the projection operator of Sowa conceptual graph. A pseudo projection defines morphism between graphs with less constraints than the Sowa original operator does. The pseudo projection of a graph *H* in a graph *G* means that, *H* is "comparable" to *G*. The formalization of the pseudo-projection operator is as follows:

Pseudo projection operator: A pseudo projection from a semantic graph $H = (C_H, A_H, \mu_H, \nu_H)$ to a semantic graph $G = (C_G, A_G, \mu_G, \nu_G)$ is a mapping Π: $A_H \rightarrow A_G$, $C_H \rightarrow C_G$ which associates an arch of *H* with an arch of *G* and a concept node of *H* with a set of concept nodes of *G*. Π has the following properties:

1. Arches are preserved but concept nodes cannot be preserved.
2. Types can be restricted or increased.

[3] In this article, "concept node" and "concept" are equivalent expressions.

Remark: A concept node can have several images by Π. As a consequence, the pseudo-projection operator makes no differences between a graph containing several concept nodes typed by t_c, for example, and another graph containing a unique concept node typed by t_c. That is the reason why, a semantic graph is considered to contain a unique concept node by type. That is defined as the *normal form* of a graph.

2.3 Similarity functions

The result of the pseudo-projection operator between graphs is Boolean: pseudo-projection exists or does not exist. Often, a matching function of IR system orders the result documents. Thus, a similarity function between graphs is defined. To this end, various similarity functions are presented. Each similarity function returns a normalized float value ranging between 0 and 1. First, thanks to the specialization relation, a similarity function between types will be defined.

2.3.1 Similarity function between types

The similarity function, noted *sim,* between types is an asymmetrical function. *sim* is defined as follows:

- If two types are not comparable then the similarity function returns 0.
- If two types are identical then the similarity function returns 1.
- If a type $t_{2.1}$ specializes another type t_2 directly, i.e. there is not intermediate type between $t_{2.1}$ and t_2 in the type hierarchy, then the similarity function returns a constant value lower than 1. For example, $sim(t_{C2.1}, t_{C2}) = V_G$ and $sim(t_{C2}, t_{C2.1}) = V_S$
  V_S and V_G are fixed arbitrary.
- If a type $t_{2.1.1}$ specializes another type t_2 not directly, i.e. there is an intermediate type $t_{2.1}$ between $t_{2.1.1}$ and t_2 in the type hierarchy then the similarity function between $t_{2.1.1}$ and t_2 is the product of the similarity functions between ($t_{2.1.1}$, $t_{2.1.}$) and ($t_{2.1.}$, t_2). For example, $sim(t_{C2.1.1}, t_{C2}) = sim(t_{C2.1.1}, t_{C2.1}) \times sim(t_{C2.1}, t_{C2})$
- If there is different lists of intermediate types between t_1 and t_2 in the type hierarchy then the similarity function between t_1 and t_2 is the maximum of the products of similarity functions between types. For example, if there exists two intermediate types t_{C3}, t_{C4} between t_{C1} and t_{C2} such that $t_{C1} \leq t_{C3} \leq t_{C2}$, $t_{C1} \leq t_{C4} \leq t_{C2}$ and $t_{C3} \neq t_{C4}$ *then* $sim(t_{C1}, t_{C2}) = Max_{i=3,4}\,(sim(t_{C1}, t_{Ci}) \times sim(t_{Ci}, t_{C2}))$

2.3.2 Similarity function between arches

The similarity function between two arches, noted Sim_A computes the average of the similarity type between each arch component.

For example, a_H is an arch such as $a_H = (c_H, c'_H)$ and $\mu(a_H) = t_{rH}$ and a_G is an arch such as $a_G = (c_G, c'_G)$ and $\mu(a_G) = t_{rG}$.

$$Sim_A(a_H, a_G) = \frac{sim(t_{rH}, t_{rG}) + sim(\mu(c_H), \mu(c_G)) + sim(\mu(c'_H), \mu(c'_G))}{3} \qquad \textbf{(1)}$$

2.3.3 Similarity function between graphs

The similarity function, noted sim_G, between a graph $H = (C_H, A_H, \mu_H, \nu_H)$ and a graph $G = (C_G, A_G, \mu_G, \nu_G)$ is the average of the similarity function between each arch and concept node of H and their images in G by Π. Because a concept node can have several images by Π, we take the maximum of the similarity function between a concept and their images.

$$\text{simG}(H,G) = \frac{\sum_{a \in A_H} sim_A(a, \Pi(a)) + \sum_{c \in C_H} Max(sim(\mu_H(c), \mu_G(\Pi(c))))}{|C_H| + |A_H|} \quad (2)$$

3 Algorithms

Now, we shall concentrate on the implementation of the matching function between graphs. Search algorithms evaluate all the pseudo-projections from the query graph to the index graphs stored in the database. During indexing, all the possible query subgraphs comparable with each index graph are memorized. Finding documents relevant for a query graph consists of identifying the subgraphs of the current query corresponding to possible query subgraphs, stored beforehand in the database. The semantic graphs are composed of arches and concept nodes. Thus, document content is represented by two different indices: a list of arches and a list of concepts, from which the normal form of semantic graph can be rebuilt. Following the works of I. Ounis [3], our algorithms are based on the association of inverted files and acceleration tables. The inverted file groups in the same entry all the documents indexed by an indexing entity. The acceleration tables store, for each indexing entity, the list of comparable entities as well as the result of the similarity function between comparable entity and indexing entity. The acceleration tables pre-compute all possible generalizations or specializations of the indexing entities. The construction of the inverted file and the acceleration table is done off-line, as part of the indexing procedure. There is a search algorithm for each kind of indexing entities. Because they are similar, only the search algorithm for arches is presented.

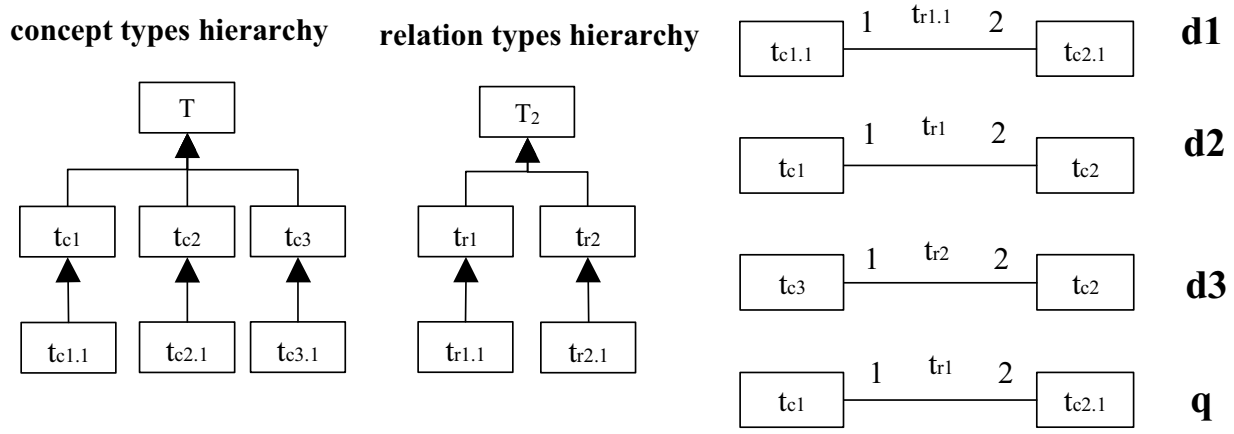


Fig. 2. Examples of query graph and index graphs and the support related to them.

To illustrate the search algorithm about arch, we will use the example of query graph (*q*) and index graphs (*d1, d2, d3*) presented in Figure 2. To compute the similarity functions, the constant *VG* and *VS* are initialized at *0.7* and *0.9* respectively.

In the inverted file (*InvertedFileArc*), the arches are grouped together in the same entry whenever they are syntactically equal. Then, for each entry corresponds a list of documents whose index contains the arch. Table 1 presents the inverted file built from the index graphs of the Figure 2. Because all arches are different, there is a different entry for each arch.

Arch identifier	*Relation type*	*Concept type of the First argument*	*Concept type of the second argument*	*Document*
a1	$t_{r1.1}$	$t_{c1.1}$	$t_{c2.1}$	*d1*
a2	t_{r1}	t_{c1}	t_{c2}	*d2*
a3	t_{r2}	t_{c3}	t_{c2}	*d3*

Table 1. an example of I*nvertedFileArc*

To build the acceleration tables, rows of the inverted file are analyzed in turn. There is an acceleration table for each arch component (first argument, second argument, relation). Their content serves to give all the specializations or generalizations for each possible arch that could appear in the query and the similarity between them.

For example, the *FirstArgValue* acceleration table of Table 2 is dedicated to the first arguments of arches. The type of the first column is the first argument type of possible query arch. The index arches whose first argument type is comparable, is associated with each of them. Each index arch is weighed by the value of the similarity function between the type of the possible query and the type of the index arch considered. For example, the type t_{C1} is associated with the arch a_1 and the similarity between t_{C1} and $t_{C1.1}$ ($t_{C1.1}$ is the concept type of the first argument of a_1) equals to $simC(t_{C1}, t_{C1.1}) = VS = 0.9$. The concept type hierarchy is used to build this table.

FirstArgValue		*SecondArgValue*		*RoleArgValue*	
Type	*Liste d'arc*	*Type*	*Liste d'arc*	*Type*	*Liste d'arc*
T	*{(0.81, a1), (0.9, a2), (0.9, a3)}*	*T*	*{(0.81, a1), (0.9, a2), (0.9, a3)}*	*Tr*	*{(0.81, a1), (0.9, a2), (0.9, a3)}*
t_{c1}	*{(0.9,a1), (1, a2)}*	t_{c2}	*{(0.9, a1), (1, a2), (1, a3)}*	t_{r1}	*{(0.9, a1), (1, a2)}*
$T_{c1.1}$	*{(1, a1), (0.7, a2)}*	$t_{c2.1}$	*{(1, a1), (0.7, a2), (0.7, a3)}*	$t_{r1.1}$	*{(1, a1), (0.7, a2)}*
t_{c3}	*{(1, a3)}*			t_{r2}	*{(1, a3)}*
$t_{c3.1}$	*{(0.7, a3)}*			$t_{r2.1}$	*{(0.7, a3)}*

Table 2. examples of acceleration tables for arch

For example, we would like to find document indices similar to the query arch *q* in Figure 2. The algorithm finds the couples (similarity, arch) in the acceleration tables

corresponding to the *q* component. In *FirstArgValue*, we select the row whose type is the type of first argument of the query arch, that is to say t_{C1}. The list of weighed arches is *{(0.9,a1), (1, a2)}*. In *SecondArgValue*, we select the row which type is $t_{C2.1.}$ The list of weighed arches is *{(1, a1), (0.7, a2), (0.7, a3)}*. In *RoleArgValue*, we select the row whose relation type is t_{r1}.the list of weighed arches is *{(0.9, a1), (1, a2)}*. We intersect these lists of arches and we weigh each arch by their weight average. Thus, the index arches similar to the query arch are *{(0.93, a1), (0.9, a2)}*. Then for each index arch, we find the list of document whose index contains it, in *InvertedFileArc*. The document list is built by replacing each index arches list by the document ID. The document list is *{(0.93, d1), (0.9, d2)}*. A document is weighed by the maximum of the weight of its index arches. Each document is added in the result list. The result list weight is equal to the sum of arches list weight divided by the number of arches and concepts contained in query arch. The result list is *{(0.31, d1), (0.3, d2)}*. At this point, the result list contains all the documents whose index contains the comparable arches of query arches.

Usually, the cost of a projection operator between graphs is prohibitive, because in graph theory, it is equivalent to find a morphism between indefinite structures. To overcome this problem, the graph structure is limited, that is to say a semantic graph is supposed to contain a unique concept node.

4 SyDoM Prototype

We have developed an information retrieval system based on semantic graph. This documentary system is called SyDoM for Multilingual Documentary System [4]. SyDoM is composed of three modules:

1. The semantic thesaurus module is presented in 4.1 chapter.
2. The indexing module indexes and annotates XML documents with semantic graphs using a set of metadata associated to the semantic thesaurus. Explanations are given in chapter 4.2.
3. The retrieval module performs multilingual retrieval. The users choose their query component in the semantic thesaurus presented in their native language. This module is presented in chapter 4.3.

4.1 The Semantic Thesaurus Module

The semantic thesaurus is the core element of the SyDoM system. Thus, the creation of this thesaurus is very important for the indexing and research processes. Using this module, indexers, who are in charge of a specific domain, represent their domain conceptualization by creating a hierarchy of concepts types and one of relation types. Moreover the type definition is associated to the vocabularies creation. Thus each type can be linked to a set of terms.

4.2 The Indexing Module

The goal of this module is to index a document with a semantic graph describing its content. In our system, indexes are stored directly inside the document content. So, indexing module enriches XML documents by adding new semantic tags inside the document content. As shown in figure 3, this module is composed of several parts:

- The right part is composed of a navigator. So users can navigate inside the semantic thesaurus in order to choose a specific type.
- The left part is composed of a tabs list. Each tab is an editor that presents a specific view of the XML document. The first tab presents the tree architecture of the XML document; the second one is composed of the XML tags. The third one presents the semantic graph associated to the document contents etc…

The semantic thesaurus and semantic graphs are presented in a language chosen by the user. At any time users can change the presentation language by selecting a new language using the top left button located above the editor.

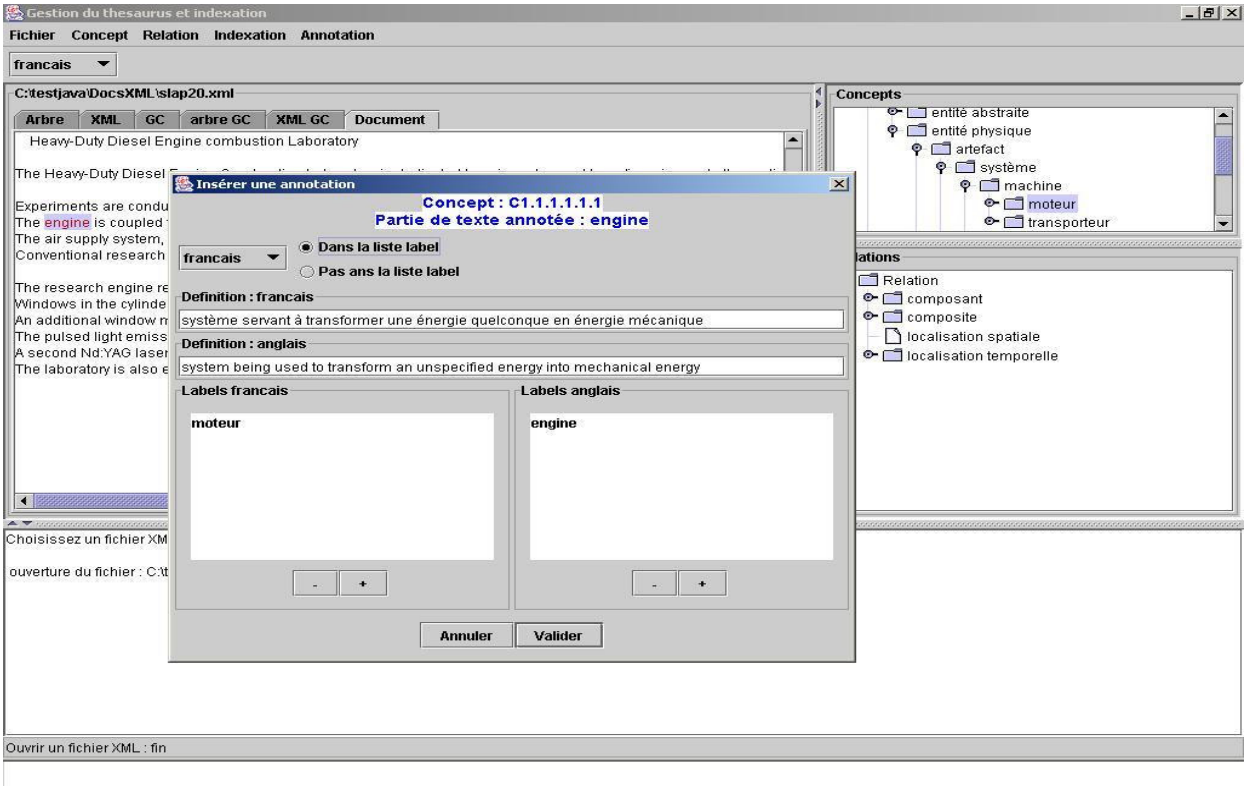


Fig. 3. The annotation module

The first stage of indexing process is to annotate document content. An annotation links a term or an expression taken inside the document content with a concept type. To build an annotation, the user selects a part of text in the document and a type of concept in the semantic thesaurus. Then, he selects in the annotation menu the function: insert annotation. As shown in figure 3, a window appears to present information related to a new annotation. In this example, the indexer annotates the part of text "engine" with the concept represented by the French term "moteur". This annotation is represented in XML documents by a TERM tag encapsulating the term "engine". This tag has an attribute semkey indicating the concept type C1.1.1.1.1.1

associated with the term. Annotations will be used to update lexical knowledge of the semantic thesaurus. Indeed, if the term selected in the annotation does not exist has a term of the concepts type in the semantic thesaurus, this new term can be added in the term list of the concepts type. To update the term list of the concepts type, user needs to select the button “dans la liste des labels” inside the annotation windows.

After annotation, annotated parts of texts are highlighted in the document. Thus, indexer can visualize rapidly significant terms in the document. In the graph editor, a concept node represents annotation. Using this editor and the thesaurus navigator, indexer can add new concepts nodes, link concept node with relations in order to improve the index graph. Once the semantic graph is completed, a set of semantic tags is inserted in the document beginning (For more details see [5]). Then, the database containing the indices is updated.

4.3 The retrieval module

The retrieval module contains the same parts as the indexing module, i.e. a tabs list of editors and a thesaurus navigator. The graph editor allows the user to build a query in the language of his choice by selecting types, in the semantic thesaurus. For example, figure 4 presents a query graph dealing with "the noise generation in diesel engines". The user visualizes the graph nodes using the vocabulary terms, while the system works only on nodes types. In the last version of SyDoM prototype all research and indexing algorithms are implemented, i.e. the comparison function between graphs takes into account the similarity between arcs and the similarity between concepts type. In SyDoM, these comparison functions evaluate the similarity between indexing graphs and query graphs label with type. Thus SyDoM can find the most relevant documents for query thank to the similarity calculation and order them. This document list is presented in figure 4.

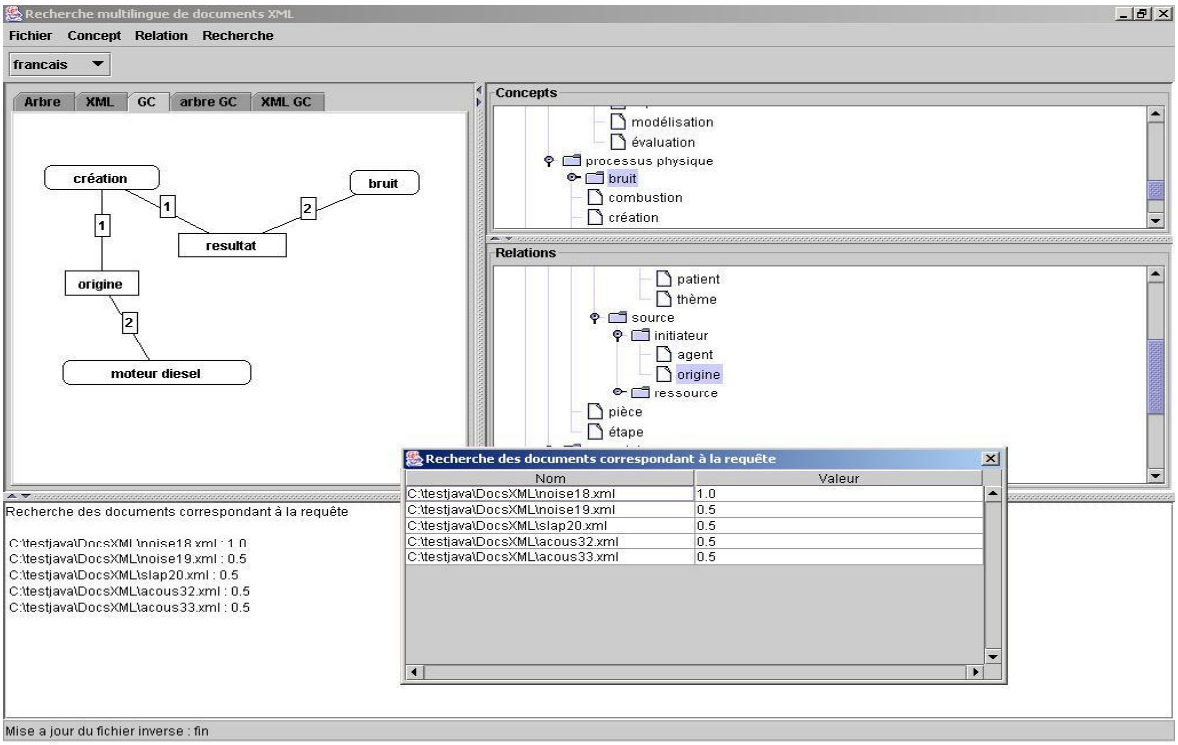


Fig. 4. A query example and its result

When a list of relevant document is presented to the user, he can select a document for visualization. SyDoM adapts automatically the document presentation to the user language using the lexical knowledge stored in the semantic thesaurus.

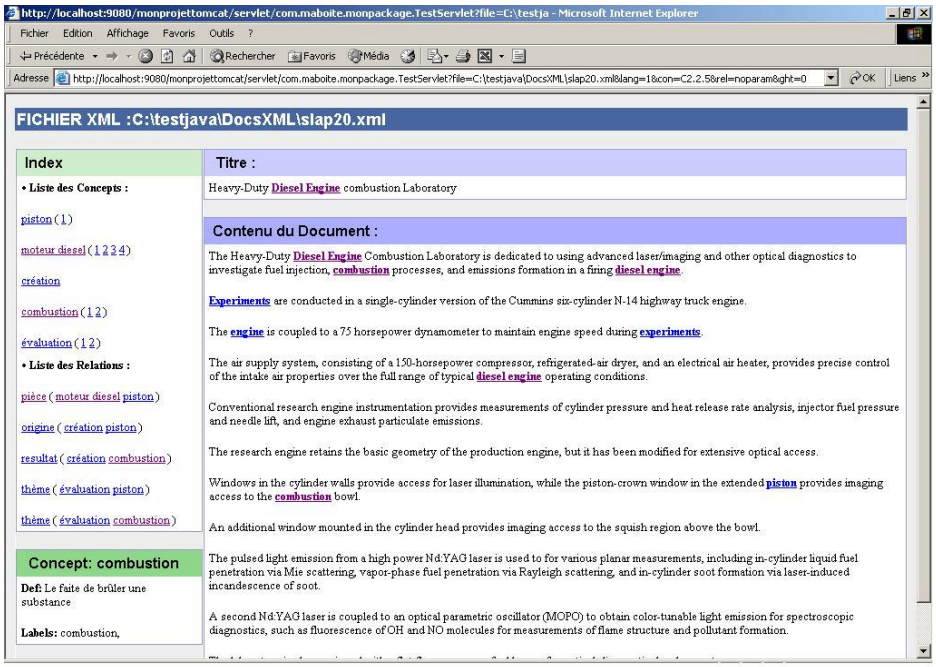


Fig. 5. An example of resulting document

The semantic tags representing the document index graph are replaced by terms chosen in the user language. This enables to form the document summary presented in figure 6. The annotations contained in the document point by hypertext links to the concept type definitions presented in the user language. An example is given in figure 7.

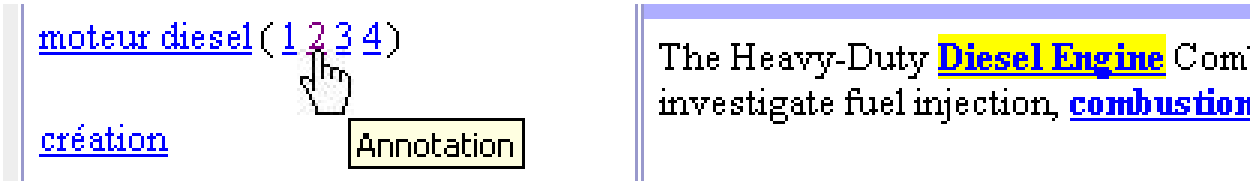


Fig. 6. Summary usage

If user clicks on "2" in the document summary, SyDoM highlights the second annotation of the concept type in the text.

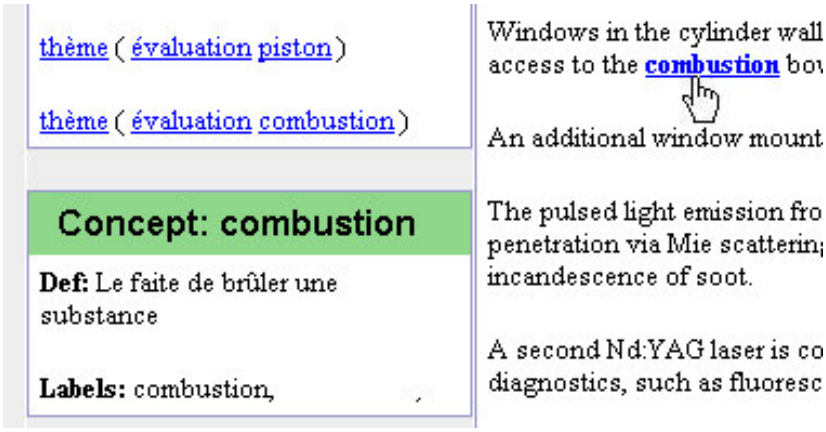


Fig. 7. Hypertext link example

If user clicks on hypertext link inside the text document, SyDoM displays the concept definition in the user language.

5 Experiments

The library Doc'INSA associated to the National Institute of Applied Science of Lyon gives us a test base of English articles. These articles deal with mechanics and they are called *pre-print of the Society of Automotive Engineers (SAE)*. The first step of the experiment was to build a semantic thesaurus for mechanics. Thanks to a mechanical thesaurus, we selected around one hundred mechanical concepts to supply our semantic thesaurus. On top of the concept hierarchy, we added the thirty-five relations proposed by Sowa in his Knowledge Book [8]. During manual indexing, only titles are taken in account. For our first experiments, approximately fifty articles were indexed manually and ten queries were performed.
Our system was compared to the Boolean system used at Doc'INSA. Indices of Doc'INSA system were generated automatically from those of SyDoM, to avoid

variability. Figure 8 presents this evaluation. The average precision was computed for ten recall intervals. We can notice that relation treatments and hierarchy inference improve significantly the quality of the answer even for manual indexing.

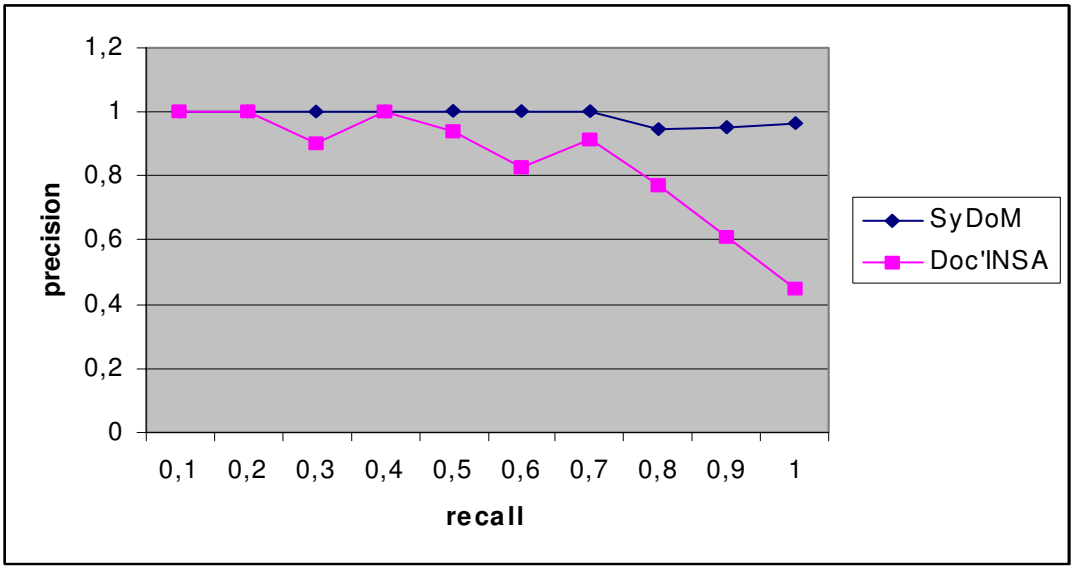


Fig. 8. Evaluation of SyDoM (threshold = 0.6) and Doc'INSA system.

6 Related works

Several IR systems, based on CG formalism, have been developed. None of them can perform multilingual retrieval:

The Ontoseek system [2] use manual indexing to retrieve yellow pages or products catalogues thanks to Wordnet hierarchy. It defines the match as a projection of the query graph onto a subgraph of index graph. Our matching function improves rappel because it retrieves more subgraph of index graph than those that specialized query graph.

The Relief system [3] enriches index graphs by adding new arches thanks to relation properties like similarity, transitivity, etc… One of the contributions of this work is to propose a fast implementation of the projection operator based on inverted file and acceleration tables. Our works extends the Relief method in order to compute rapidly similarity between graphs.

D. Genest [1] has noted that the projection operator is not adapted to IR purpose. First, matching functions based on projection give Boolean results. Secondly, document is not relevant for a query, if its index graph contains only one node, which is a generalization of the query node or if graph structures are different. In order to take such problems into account, D. Genest defines some transformations on conceptual graph. Moreover a mechanism is proposed to order sequences of transformations. As a consequence, the matching function based on projection becomes a ranking function and orders relevant documents for a query. We take in account D. Genest remarks in order to develop our matching function for IR purpose.

J. Zhong and all [9] propose a matching function between graphs very similar to ours. They propose an advance similarity function between types taking in account the type position in the hierarchy. Their algorithm complexity is the drawback of their approach because it uses recursive calculation. Our approach is only based on intersection sets.

7 Conclusion and perspectives

In this paper, we have presented an extension of the Sowa formalism of Conceptual Graphs in order to share a conceptualization in different languages. This new graph formalism called semantic graph enables to describe the semantics of document contents in a multilingual context. Moreover, a new comparison operator between graphs is proposed. Our contribution is validated by the SyDoM prototype dedicated to digital libraries. SyDoM has several useful functionality: Indexing knowledge are modeled in a semantic thesaurus; Users can chose the language used to present indexing knowledge, Annotation enables to update the vocabulary of the semantic thesaurus, users can compose their queries in the language of their choice. Relevant document are adapted to user language.

The measuring of type similarity is far from perfect and it needs further study based on type position in the hierarchy for example. Moreover, recent works [6] have shown that automatic indexing mixed with manual indexing improve effectiveness. So we are currently in studying automatic indexing for the SyDoM prototype.

8 References

1. D. Genest. Extension du modèle des graphes conceptuels pour la recherche d'information. PhD Thesis, Montpellier University, Montpellier, France (2000).
2. N. Guarino, C. Masolo, G. Vetere. OntoSeek: Content-Based Access to the Web. In IEEE Intelligent Systems Volume 14, Issue 3 (1999) p 70, 80.
3. I. Ounis, M. Pasça. RELIEF: Combining Expressiveness and Rapidity into a Single System ». Proceedings of 18th SIGIR, Melbourne, Australia, (1998), 266-274,.
4. B. Pivano, S. Calabretto, C. Roussey, J. M. Pinon. SyDoM : un système de recherche d'information multilingue basé sur des connaissances. Proceedings of IC, Lyon, France, (2004), p103-114.
5. C Roussey, S. Calabretto, J-M Pinon, A new model of Conceptual Graph Adapted for Multilingual Information Retrieval Purposes. Proceedings of DEXA, Munich, Germany. LNCS Vol 2113 Springer, (2001). p 92-101.
6. J. Savoy. Indexation manuelle et automatique : une évaluation comparative basée sur un corpus en langue française . Proceedings of CORIA, Grenoble, France (2005), p 9-23
7. J. Sowa. Conceptual Structures: Information Processing in Mind and Machine. The System Programming Series, Addison Wesley publishing Company, (1984).
8. J. Sowa. Knowledge Representation: Logical, Philosophical, and Computational Foundations. Brooks Cole Publishing Co., Pacific Grove, CA., (2000).
9. J. Zhong, H. Zhu, J. Li, Y. Yu: Conceptual Graph Matching for Semantic Search. Proceedings of ICCS, Borovets, Bulgaria, (2002). LNCS Vol 2393 Springer (2002) p 92-196

ACOSys: An Experimental System for Automated Content Organization

Guo-Qiang Zhang and Ye Tian

Department of Electrical Engineering and Computer Science
Case Western Reserve University, Cleveland, Ohio 44106, U.S.A.

Abstract. Menu-hierarchy is an important mechanism for human-computer interaction. In this paper we report the design, prototyping, and experiences of an experimental system called ACOSys for automated organization of contents by menu/folder hierarchies. ACOSys is built on a similar principle of our other work, FcAWN – formal concept analysis for web navigation-menu design [32], where lattice theory [1] provides the mathematical basis for menu-structures while the user-interface remains unchanged. Implemented in Java and Perl, our experimental prototype has the following features: (a) automated generation of folder hierarchy; (b) automatic folder-name assignment; (c) pruning for global folder-depth control; (d) lattice-unfolding to obtain trees that allow for backing-up. An experiment study is performed for ACOSys to generate an organization of the Medlars collection, an archive of 1039 medical articles commonly used for information retrieval benchmarking. ACOSys created a hyper-link navigation hierarchy in a couple of minutes based on a ten attribute set; a desktop web-browser is used to test and validate the hierarchy. The basic idea of ACOSys comes from our recognition that *menu-structures can be regarded as mathematical structures in the abstract sense and, in return, they can benefit from an order-theoretic investigation.* An additional recognition is that tree-structures are important to facilitate the backing-up operation for folder hierarchies, but the tension between trees and lattices can be resolved by unfolding lattices to trees.

1 Introduction

Considerable advances in information retrieval (IR) have not only made search-engines a routine part of our day-to-day interactions with the web but also put them on personal desktops. However, although current search-engines are effective in finding specific items (those a user can precisely define), search results for less-specific information tend to be off-target, overwhelming, and less useful. “Finding needles in a haystack” captures the overwhelming information amount, but it does not accurately reflect the issue of partial information – it assumes that the user knows what is to be found (i.e., the “needles”).

Browsing through a menu-hierarchy remains an indispensable modality in human-computer interaction. Not only can it help us navigate to contents that are unknown or imprecisely defined, an equally important reason is the simplicity for access. For example, even when search capabilities on personal computers are available, few users actually launch search-engines to locate documents as a routine. The reason is economy: most files on a personal desktop can be accessed in two to four mouse clicks,

while searching involves the launching of an application program (which may involve a slight wait), typing in keywords, and finding the targeted item from tens or hundreds of linearly-structured responses. The number of mouse and key strokes involved in searching can be far greater than that of navigation. Searching is useful in the event that an item cannot be found in the logical location it should reside.

An automated process that organizes documents according to their contents is one of the goals of our experimental system ACOSys. The resulting organizational structure should enhance user's experience in addition to the economical incentive indicated above. We use Formal Concept Analysis (FCA [1]) as the mathematical principle to help us construct both the organizational structure as well as put documents in their correct logical locations in this structure.

In addition to a systematic process suitable for automation[1], a benefit of the lattice structures derived from FCA is that they can sometimes provide better "logical" coverage of contents than the standard tree structures. For example, in the context of web-menu design [32], items related to *research facilities* should be accessible from either the *Research* upper menu, or the *Facilities* upper menu – see left of Fig. 1. This entails a diamond-shaped structure which is part of a lattice and not part of any tree because sub-branches in a tree do not overlap. Using a simple-minded tree structure (see right of Fig. 1), items related to research facilities are forced to reside in either *Research* or *Facilities*, but not both. Such a tree structure can cause confusion and miss-hit for items related to research facilities and prompt the user to use a search engine unnecessarily.

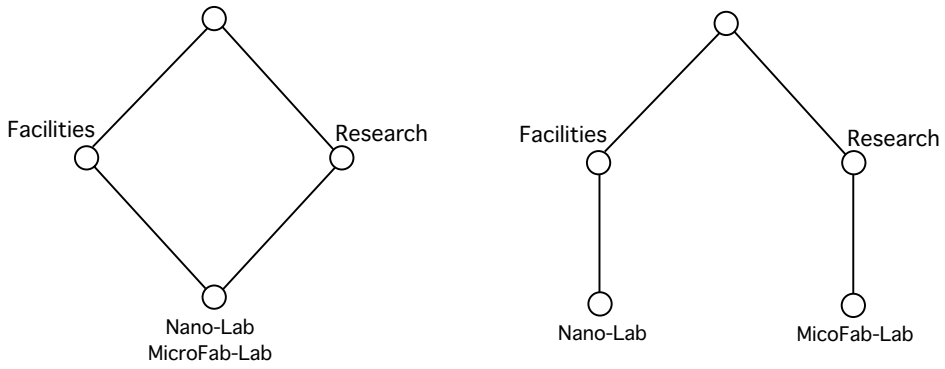


Fig. 1. Left: a lattice structure which allows research facilities such as Nano-Lab to be accessible by navigating down from both Research and Facilities. Right: a tree structure which forces research facility items to reside either in Research or in Facilities, but not both.

The situation pictured in Fig. 1 is not an isolated incident. *Documents in general have multiple attributes, and should be accessible from the root through multiple paths.* Yet multiple paths to a node are precisely what is not allowed in a tree. Manual remedies are possible, once the problem has been identified (*e.g.*, the symbolic-link mechanism in

[1] We would like to point out the subtle differences in our use of *automated* instead of *automatic*. "Automatic" is a stronger form of automation, while "automated" refers to a process which may involve limited user intervention here and there.

UNIX has been around for a while). However, what is needed is a *principled, systematic solution* so that multiple uplinks are not added as an after-thought based on an ad hoc manual process but become an integrated part of the design.

This paper introduces an experimental system called ACOSys, for Automated Content Organization. The basic idea of ACOSys comes from our recognition that menu-structures can be regarded as mathematical structures in the abstract sense, and in return, they can benefit from an order-theoretic investigation. Thus ACOSys shares the common algorithmic basis derived from FCA as FcAWN (and hence the benefits of lattices), but differs in the application domain, as well as the degree of automation in that document-classification is to be performed automatically. While FcAWN provides a formal method for web-menu design, the construction of formal context and the labeling of menus were semi-automatic. Key features of ACOSys include:

- Automated organization of electronic contents using a navigation-menu hierarchy
- Automatic menu-label and folder-name generation
- Pruning for application-dependent menu and folder nesting depth control
- Lattice-unfolding to obtain trees for backing-up in the navigation hierarchy

An experiment study has been carried out for ACOSys to generate an organization of the Medlars collection [2], an archive of 1039 medical articles commonly used for information retrieval benchmarking. ACOSys created a hyper-link navigation hierarchy in a couple of minutes based on a 10 attribute set. One of our observations is that although lattices are essential for providing the correct menu hierarchy, tree-structures are important to make the familiar backing-up operation available. However, the tension between the two can be resolved by unfolding lattices to trees.

Related work. An extensive array of research [9, 15, 16, 22, 24, 26, 28] on document organization and web navigation sets a good background for our work. Specially relevant to our results are creative FCA based works on *document classification and information retrieval*; however, most of them are innovative in either using lattice diagrams as an explicit *augmented* part of a user-interface modality [5–7, 14], or as an aid in clustering and in formulating queries for document retrieval [23]. Knowledge acquisition techniques based on FCA for the incremental improvement of a browsing mechanism for domain specific document-retrieval systems were proposed in [19]. We would also like to point out CREDO [4], an FCA-based online exploration system which combines concept visualization and user querying in a web browser. Our approach emphasizes the ubiquitous value of FCA *for the general content organization problem*, where a user navigates a standard menu or folder hierarchy without submitting any "queries". For both FcAWN and ACOSys, the clicking of menu or folder icon (with proper names attached) provides the simplest mechanism for user input during exploration and browsing. To account for this general approach, ACOSys needed to provide the backing-up facility to the user, achieved by our introduction of the lattice unfolding technique. ACOSys provides the organizational backbone of an electronic collection through FCA *without any altering of the familiar user modality*. ACOSys is also significantly different from FcAWN [32] in its use of lattices through their unfolding to trees, in the degree of automation, and in its application for organizing desktop or online collections.

The rest of the paper is organized as follows. Section 2 introduces auxiliary notions associated with lattices and FCA that ACOSys relies on. Section 3 presents the design of ACOSys. Section 4 describes the main steps for the implementation of ACOSys. Section 5 reports the results of our preliminary experiments with ACOSys. Section 6 concludes with some future work.

2 Auxiliary Notions

We freely use terminology and notations associated with FCA without further intrudution; it is covered in detail elsewhere [1]. We introduce some auxiliary notions not usually covered by FCA but will be important for ACOSys.

Definition 1. *Let* $(L, \leq)$ *be a finite lattice. The depth of a node* $x \in L$ *is defined to be the minimum of the lengths of all paths from* x *to the root/top of* L*.*

The depth concept provides well-defined meaning for the menu-depth control/pruning feature in ACOSys, discussed later.

Note that the depth of a node in a lattice is a less intuitive concept than the one restricted to trees. The root has depth 0, and all the nodes immediate below the root have depth 1. However, in general, even if x is below y ($x < y$ in L) the depth of x may not be greater than that of y, because there may be a shorter path from x to the root/top that does not go through y. For example, in Fig. 2, the depth of e ($= 2$) is the same as the depth of g, even though g is strictly below e.

Our next definition provides one version of the definition of tree that leads to a most direct formulation of lattice-unfolding (Definition 3). We clarify the notations used in Definition 2 to avoid confusion. Given a finite, non-empty set S, we denote by S^* the set of all finite strings consisting of an ordered finite sequence of symbols/elements from S. Given strings $x, y \in S^*$, we denote by xy the concatenation of x and y. The empty string is denoted by λ. If $z = xy \in S^*$, then we call x a prefix of z. The empty string is a prefix of any string.

Definition 2. *A tree* T *over the label set* S *is a non-empty, prefix-closed subset of* S^**. In other words, a tree is a non-empty set of strings* T *such that* $\lambda \in T$ *and if* $xy \in T$*, then* $x \in T$*. Often* S *is chosen to be* $\{0, 1\}$*, in which case we obtain the binary trees; or* ω*, the general trees labeled by natural numbers.*

The next diagram is our illustrative example for Definition 3.

Definition 3. *Let* $(L, \leq)$ *be a finite lattice. The unfolding of* L *is defined to be the tree (over the label set* L*) obtained from* L *as the set of prefixes of all paths from the root/top to the leaf nodes, with the root labeled by the empty string.*

The distinction in the unfolded tree is that different nodes in the tree are represented by different *paths*, and not merely the last symbol of the path as in the original lattice.

The use of trees is motivated by the need to make an unambiguous "backing-up" operation available in ACOSys. However, it is important to note that this is different from a simple-minded, ad hoc use of tree structures as in Fig. 1 (right). Trees systematically derived from concept lattices through unfolding allow for backing-up while at the same time share the benefits of lattices obtained via FCA.

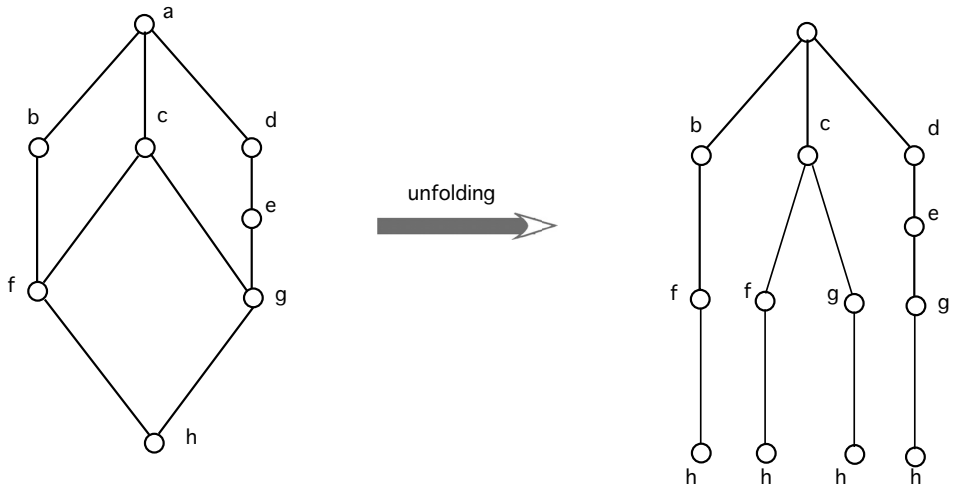


Fig. 2. Lattice unfolding.

3 Design of ACOSys

Architecturally, ACOSys consists of three modules and two algorithms. The three modules are: the Document Classification Module, the Concept Analysis Module, and the Folder Generation Module. The first of the two algorithms is used for automatic folder-name[2] assignment, and the second algorithm is for assigning documents to the corresponding folders (Fig.3).

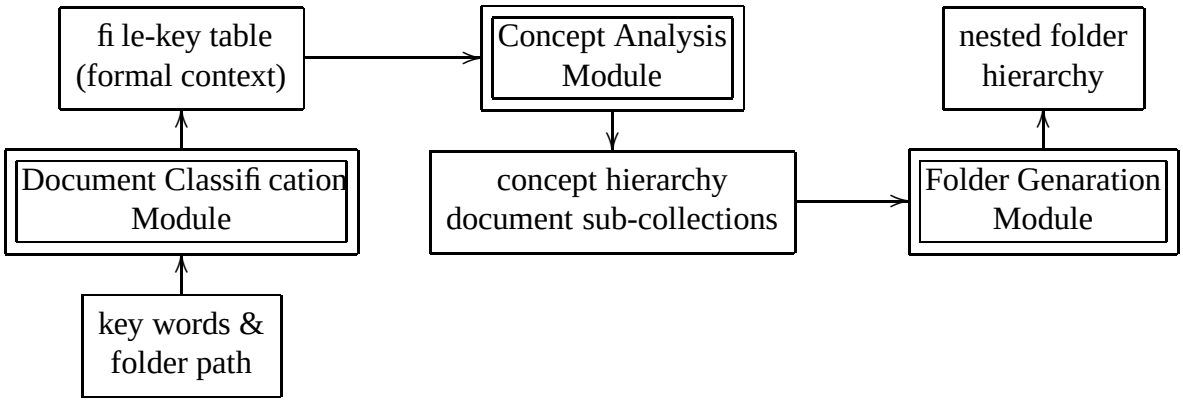


Fig. 3. ACOSys architecture: double-framed boxes represent modules and single-framed boxes represent input-output data.

Document Classification Module. The input of the Document Classification Module is a set of files (specified by a directory path) and a set of keywords (user inputs). The output of the module is a file-keyword matrix, in XML format. This matrix is the formal context – input to the Concept Analysis Module.

[2] In the rest of the paper we use "folder" and "menu" interchangeably.

Concept Analysis Module. This is used to generate a concept lattice hierarchy based on the file-keyword matrix generated from the Document Classification Module. It generates a concept collection in XML, consisting of both the attributes (keywords) and objects (files) of all associated concepts.

Folder Generation Module. This module reads the concept collection specified in the XML file given in the Concept Analysis Module and builds the corresponding folder/menu structure. Two algorithms are used. The *folder naming algorithm* automatically generates labels/names for each concept/folder, while the *file relocation algorithm* creates hyperlinks (or symbolic links) in their corresponding folders automatically. The separation of the two algorithms is for conceptual distinction only. In implementation they could well be combined.

4 Implementation

This section describes the implementation of each of the ACOSys components in more detail.

4.1 Document Classification Module

The input to the Document Classification Module consists of the path of the files to be processed and the user-defined keywords (Figure 4 contains a screen shot of sample inputs). Based on this input, the Document Classification Module generates a file-keyword matrix (i.e., "formal context" in FCA), in which the rows represent files, and the columns represent the keywords. The module searches all the keywords in every file by simple string matching. If it finds a keyword in a file, it marks "1" on the corresponding column in the matrix; otherwise "0" is entered. For interoperability, the matrix is specified in XML format following ConExp [31]: the keywords are defined in the ⟨Attribute⟩ tag and the files are defined in the ⟨Object⟩ tag. Their relationship is defined using the ⟨Intent⟩ tag.

Our Document Classification Module is implemented as a 500 line Perl program. Perl has an extensive string processing library which makes the task easier. A conversion of this program to Java might be helpful in the future to put all components in one package.

Note that any more "intelligent" document classification programs can replace our simple-minded string matching algorithm. Our initial goal in ACOSys is to demonstrate the system concept. Important as they may be, we made a choice in the first version not to focus on the issues of *semantics-based document classification and intelligent keyword selection*. These may be incorporated in future versions to make ACOSys more practically.

4.2 Concept Analysis Module

The Concept Analysis Module builds concept hierarchies in three steps based on a well-known algorithm given in [1]. We provide a brief account of these steps in the context of ACOSys to make the paper self-contained.

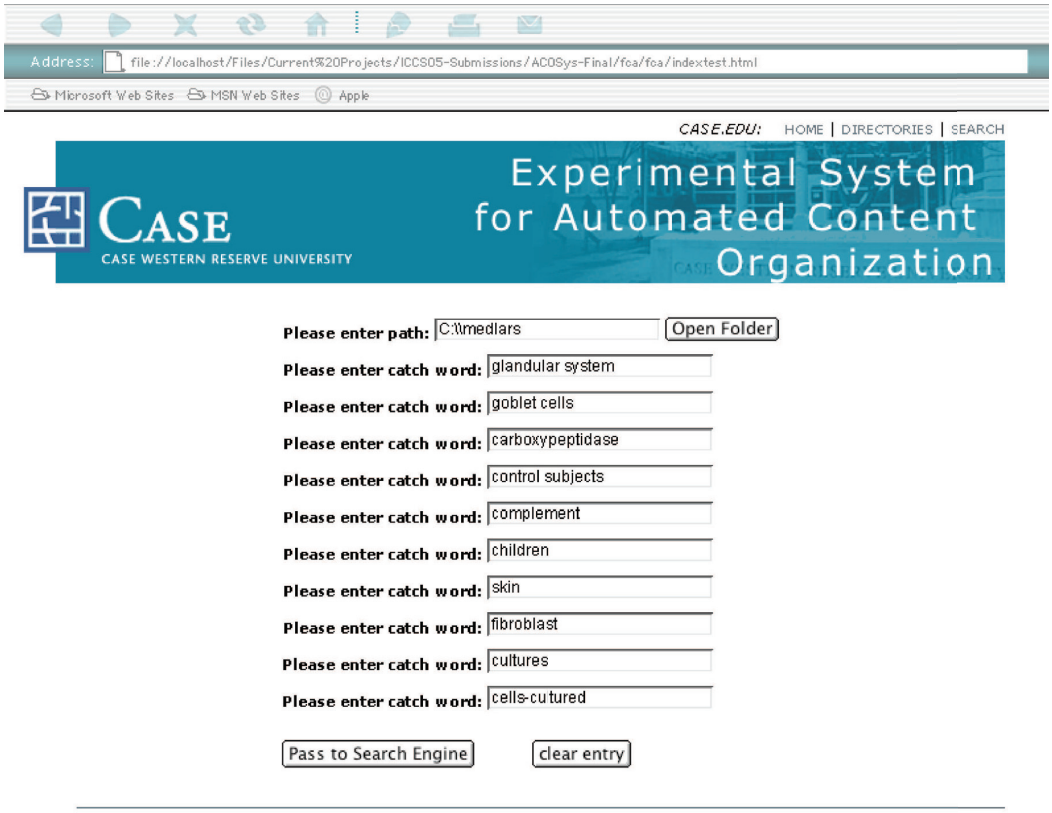


Fig. 4. Inputs for Medlars collection.

1. *Building object-concepts.* By a basic fact of FCA, each row in the file-keyword matrix will be a concept. We build a baseline concept table based on this fact. New concepts are inserted in the next step if they cannot be found in the table.
2. *Adding new concepts.* The concept table of "current concepts" is initialized in the previous step and repeatedly updated in this step. Here, (binary) intersections between any two concepts are computed. If the result is not found in the table, it is inserted. This step is repeated until no new concepts are found.
3. *Adjoining the top and the bottom concepts.* The top concept (concept-0) is constructed as a pair ({All Files}, {keywords}) of all the files together with the set of keywords, if any, which appear in all the files. The bottom concept (concept-N) is a pair ({Files}, {All keywords}) of all the keywords together with the set of files, if any, that contain all the keywords.

Note that step 2 and 3 correspond to the construction of a closure system generated from object concepts in step 1 which, according to FCA [1, 33, 34], is precisely the concept lattice determined by the file-keyword context.

The output is also formated as an XML file conforming to ConExp [31]. In addition to the XML file generated by the Document Classification module, the concepts are delineated by ⟨Concept⟩, and the concept-keyword instances are delineated by ⟨HasAttribute⟩.

4.3 Folder Generation Module

The Folder Generation Module builds a folder hierarchy based on Concept Analysis Module's output XML file in three steps: (1) constructing the folder structure; (2) assigning folder names; (3) creating file-folder memberships.

Constructing the folder structure. The XML file provided by the Concept Analysis Module contains a list of concepts. This list is represented as a Java vector, while the items in the vector are represented by a Java object capturing the data contained in a concept. Each Java concept object contains the concept label, the associated attribute set, and the number of attributes. The objects are not explicitly stored as part of the concept object; they can be derived by context table lookup.

The folder structure is captured by the coverage relation between concept objects. By pairwise comparison of concepts, we obtain each concept's subsumed concepts and covered concepts. This is represented as a tuple (concept-label, list of strict subconcepts, list of covered concepts).

From the example given in Fig. 5, we have a Java vector $(S_1, S_2, \ldots, S_8)$, where each S_i is a concept object:

$$
\begin{aligned}
S_1 &= (C_1, [C_4, C_7], [C_4]) \\
S_2 &= (C_2, [C_5, C_7], [C_5]) \\
S_3 &= (C_3, [C_2, C_5, C_7], [C_2]) \\
S_4 &= (C_4, [C_7], [C_7]) \\
S_5 &= (C_5, [C_7], [C_7]) \\
S_6 &= (C_6, [C_4, C_5, C_7], [C_4, C_5]) \\
S_7 &= (C_7, [\,], [\,]) \\
S_8 &= (C_0, [C_1, C_3, C_6, C_4, C_2, C_5, C_7], [C_1, C_3, C_6])
\end{aligned}
$$

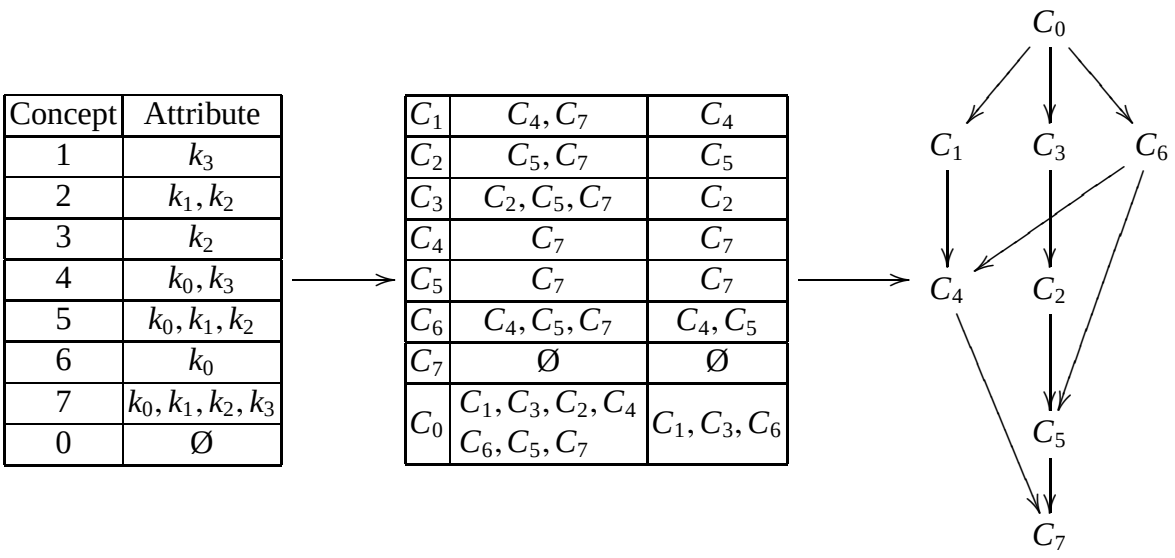


Fig. 5. Illustrative diagram for folder structure gneration.

Assigning folder names. As described above, each Java concept object contains the concept label, the associated attribute set, and the number of attributes. Also, concepts with less attributes generally appear on the top levels of the folder hierarchy, with the root folder containing attributes shared by all documents. Our naming scheme follows this simple strategy: let A be the attribute set of the current concept and B be the union of all the attributes of concepts along the path to the top/root, excluding the current concept. Name the current folder/concept by the concatenation of attributes in $A \setminus B$. Since A represents a new concept, it is necessarily different from B and hence the set is always non-empty and the name exists.

Note that Definition 3 provides the theoretical justification for our naming method. The generated folder structure, although conceptually a lattice, is in fact a tree. Just as different folders can contain identical file/folder names, our naming scheme guarantees that each folder and each document has a unique access path, either represented as a hyper-link or as a directory path.

Creating file-folder memberships. The last step is to put files into their corresponding folders and generate their symbolic links automatically. This will be based on the file-keyword matrix available from the Document Classification module. For example, suppose we have the entries (F_0, $\{k_0, k_3\}$), (F_1, $\{k_1, k_2\}$), (F_2, $\{k_0, k_1, k_2\}$), (F_3, $\{k_3\}$), (F_4, $\{k_2\}$) in the file-keyword matrix (rows). Suppose the Java concept vector gives us these concepts with their attributes: (Concept1, $\{k_3\}$), (Concept2, $\{k_1, k_2\}$), (Concept3, $\{k_2\}$), (Concept4, $\{k_0, k_3\}$), (Concept5, $\{k_0, k_1, k_2\}$), (Concept6, $\{k_0\}$). We then *pseudo-compose* the file-keywords relation (the formal context) with the (reverse of) the concept vector to obtain the following triples: (F_0, $\{k_0, k_3\}$, Concept4), (F_1, $\{k_1, k_2\}$, Concept2), (F_2, $\{k_0, k_1, k_2\}$, Concept5), (F_3, $\{k_3\}$, Concept1), (F_4, $\{k_2\}$, Concept3) where we only keep the largest concept (with the least set of attributes) in the third component (which always exists in a complete lattice).

In general, suppose R and S are binary relations on some *powerset*. The *pseudo-composition* of R and S, still denoted as $R \circ S$, is defined as

$$R \circ S = \{(A, B) \mid \exists X, Y\ (X \subseteq Y) \,\&\, (A, X) \in R \,\&\, (Y, B) \in S\}.$$

Each B in a tuple (A, B) can be required to be maximal (*i.e.*, Y to be the minimal one containing X) to eliminate redundancy.

4.4 Variable folder-nesting depth

An additional feature of ACOSys is variable menu-depth control, to account for different application needs and user preferences. Possible user selections are “all", or a particular positive integer. “All” means no menu-depth restriction is enforced the complete lattice, or its unfolded version, is constructed. If, for example, the user selects depth 3, then the resulting folder structure will contain folders of depth up to 3 (inclusive) according to Definition 1. The current implementation is achieved by first constructing the whole lattice, and then recursively merging documents and folders up to the desired depth. Techniques for constructing the iceberg concept lattice [27] may be helpful for a more efficient implementation, which will be considered for updated ACOSys versions.

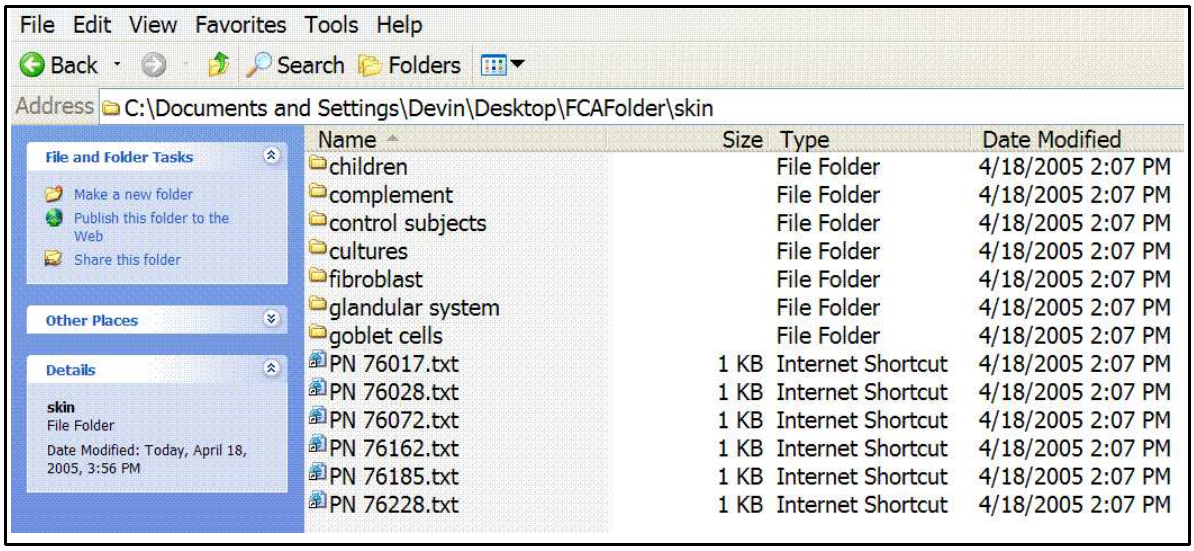


Fig. 6. Folders for Medlars collection.

5 Experiments

ACOSys is implemented on a Dell Latitude D800 laptop running Windows XP, with 512MB RAM, a 40G hard drive and a 1.6Ghz CPU. The programming languages used are Java (J2SDK1.4.2_03) and Perl (ActivePerl5.6.1.635). The Perl program, for document classification, consists of 500 lines. The rest of the system, written in Java, consists of more than 3000 lines of code. Because of the platform-independent nature of Java and Perl, ACOSys will be platform-independent: it can run under Mac OS X, Unix, and Linux.

ACOSys has a simple user interface. The user input consists of a source file path, a set of keywords filled in a GUI table, and menu depth. ACOSys then generates a hyperlinked menu-hierarchy, which can be accessed using a web-browser (Figure 7 contains a screen shot of a sample menu-hierarchy). Alternatively, ACOSys can construct a folder hierarchy residing in the local hard disk, to allow for navigation as a normal part of a user's desktop environment (Figure 6 contains a screen shot of a sample folder structure).

We have successfully used 10 keywords for organizing the Medlars collection of 1039 documents, in 133 seconds. Part of the generated folder lattice diagram is shown in Fig. 6.

Validation. To make sure that the folder structure is correct, we manually compared the lattice constructed using ConExp [31] with our ACOSys output, using the same XML context file generated by the Document Classification Module. Although the whole structure is compared and found to be correct, our initial depth-control algorithm contained an error in menu-depth, which was then corrected. We found that update in-place, to save memory, is trickier to implement and we resorted to a more straightforward but more memory intensive algorithm.

Fig. 7. Menus for Medlars collection.

6 Concluding Remarks

ACOSys is an experimental system aimed to demonstrate the feasibility of automated content organization, seamlessly integrated into *existing navigation modalities*. Our goal is to bring the benefits of FCA to menu and folder hierarchy structuring in a stealthy way (the user does not interact with Hasse diagrams explicitly and needs to know neither FCA nor the underlying algorithms in order to use the system). ACOSys achieved its design goals but more work needs to be done to make it a practical application. We briefly mention a few of the immediate directions.

Incremental concept hierarchy construction. The initial construction of menu/folder hierarchy can be carried out in batch mode (overnight, for example) for large collections and a substantial set of keywords. Additional documents/contents should be added incrementally, and not as an overhaul, as they become available. Techniques for incremental lattice construction in FCA will be relevant.

Keyword selection. The selection of the keyword set is an important aspect which determines the efficacy of the organization structure and its usage. We feel that this can at most be a semi-automated process involving decision-making by a user based on measures such as word-distribution statistics about a collection. Existing ontologies may be a good place to look for guidance in an established area.

Document classification. Simple string-match, as used here, are not ideally suited for document classification because it ignores the underlying semantics. A document may well be highly relevant to a topic area (given by keywords) and yet the keywords

may not appear anywhere in the document. Although many techniques are available to make document classification more intelligent, this is still a wide-open area.

We feel that the contribution of experimental systems such as ACOSys lies in the creative synthesis of existing techniques for specific application scenarios. Each component of the system, considered individually, may represent a simple, existing idea, but the system taken together as a whole may provide a value greater than the sum of its parts.

References

1. B. Ganter and R. Wille. Formal Concept Analysis. Springer, Berlin, 1999.
2. R. Baeza-Yates and B. Ribeiro-Neto (eds.). *Modern Information Retrieval*, Addison-Wesley Longman Publishing Company, 1999.
3. P. Becker and P. Eklund. Prospects for Formal Concept Analysis in Document Retrieval. *Australian Document Computing Symposium (ADCS01)*, pages 5-9, University of Sydney, Basser Department of Computer Science, 2001.
4. C. Capineto and G. Romano. *Concept Data Analysis: Theory and Applications*. John Wiley, England, 2004.
5. R. J. Cole. *The Management and Visualization of Document Collections Using Formal Concept Analysis*. Ph.D. Thesis, Griffith University, Australia, 2000.
6. R. Cole and P. Eklund. Browsing semi-structured web texts using formal concept analysis. In *Proceedings of the 9th International Conference on Conceptual Structures*, pp. 319 - 332, 2001.
7. R. Cole, P. Eklund, and G. Stumme. Document retrieval for email search and discovery using formal concept analysis. *Applied Artificial Intelligence*, Vol. 17, pp. 257 - 280, 2003.
8. R. Cole and G. Stumme. CEM - A Conceptual Email Manager. *Conceptual Structures*, LNCS 1867, pages 438-452. Springer, 2000.
9. M. Crampes and S. Ranwez. Ontology-supported and ontology-driven conceptual navigation on the World Wide Web. In: *Proceedings of the 11th ACM conference on Hypertext and Hypermedia*, San Antonio, Texas, USA, pp. 191 - 199, 2000.
10. W. Chuang, A. Tiyyagura, J. Yang, G. Giuffrida. Fast Algorithm for Hierarchical Text Classifi cation. In Proc. Of the 2nd Int. conf. On *Data Warehousing and Knowledge Discovery (DaWaK'00)*, pages 409-418. London-Greenwich, UK, 2000.
11. Email Analysis Pty Ltd: Mail-Sleuth homepage. *http://www.mail-sleuth.com*.
12. P. Eklund and R. Cole. *Structured Ontology and Information Retrieval for Email Search and Discovery*. Int. (ISMIS02), pages 11-27, LNAI 2393, Springer Verlag, 2002.
13. P. Eklund and R. Cole. HierMail homepage. `http://hiermail.com`.
14. P. ÊEklund, J.ÊDucrou,Ê and P.ÊBrawn. Concept lattices for information visualization: can novices read line-diagrams? In Peter Eklund (Eds.), Proceedings of ICFCA 2004, *Concept Lattices: Second International Conference on Formal Concept Analysis*, LNAI 2961, Sydney, Australia, pp. 57 - 73, 2004.
15. M. Hearst. User interfaces and visualization. Chapter 10, *Modern Information Retrieval*, Edited by Ricardo Baeza-Yates and Berthier Ribeiro-Neto, Addison-Wesley Longman Publishing Company, pp. 257 - 323, 1999.
16. M. Ivory, and M. Hearst. Improving web site design. *IEEE Internet Computing*, Vol. 6 (2), pp. 56 - 63, 2002.
17. M. Ivory, R. Sinha, and M. Hearst. Empirically validated web page design metrics. In *ACM Conference on Human Factors in Computing Systems*, CHI Letters 3(1), pp. 53 - 60, 2001.

18. M. Ivory and M. Hearst. The state of the art in automated usability evaluation of user Interfaces. *ACM Computing Surveys*, Vol. 33 (4), pp. 173 - 197, 2001.
19. M. Kim and P. Compton. Evolutionary document management and retrieval for specialized domains on the web. *Int. J. Hum.-Comput. Stud.*, Vol 60 (2), pp. 201 - 241, 2004.
20. R. Korfhage. *Information Storage and Retrieval*. New York, Wiley (1997).
21. C. Lindig. Fast Concept Analysis. In Gerhard Stumme, editors, *Working with Conceptual Structures* - Contributions to *ICCS 2000*, Shaker Verlag, Aachen, Germany, pp. 152 - 161, 2000.
22. Ê R. Pollard. A hypertext-based thesaurus as a subject browsing aid for bibliographic databases. *Information Processing and Management: an International Journal*, Vol. 29 n.3, pp. 345 - 357, 1993.
23. U. Priss. Lattice-based information retrieval. *Knowledge Organization*, Vol. 27, pp.132 - 142, 2000.
24. R. Rizzo, G. Fulantelli, M. Allegra. Browsing a document collection as an hypertext. *Proceedings of World Conference on the WWW and Internet*, San Antonio, Texas, USA, pp. 454 - 458, 2000.
25. G. Salton. *Automatic Text Processing: the Transformation, Analysis, and Retrieval of Information by Computer*. Reading, Massachusetts, Addison-Wesley, 1989.
26. E. Stoica, M. Hearst. Nearly-automated metadata hierarchy creation. *In the Companion Proceedings of HLT-NAACL'04, Boston*, pp. 117 - 120, 2004.
27. G. Stumme, R. Taouil, Y. Bastide, N. Pasquier, L. Lakhal. Computing iceberg concept lattices with TITANIC. *Data and Knowledge Engineering*, Vol. 42 (2), pp. 189 - 222, 2002.
28. D. Tudhope, D. Cunliffe. Semantically indexed hypermedia: linking information disciplines. *ACM Comput. Surv.* Vol. 31:4, 1999.
29. W. Wibovo and H. Williams. Simple and Accurate Feature Selection for Hierarchical Categorisation. In Proc. Of the 2002 ACM symposium on *Document Engineering*, pages 111-118. McLean, Virginia, USA, 2002.
30. R. Wille. Restructuring Lattice Theory: An Approach Based on Hierarchies of Concepts. In: I. Rival (ed.); *Ordered Sets*, pages 445-470. Reidel, Dordrecht-Boston, 1982.
31. S. Yevtushenko. ConExp. `http://sourceforge.net/projects/conexp`
32. Guo-Qiang Zhang, G. Shen, J. Staiger, A. Troy, J. Sun. FcAWN: Concept Analysis as a Formal Method for Automated Web-Menu Design. *Conceptual Structures at Work, Shaker Verlag*, pages 141-145, 2004.
33. Guo-Qiang Zhang. Chu spaces, formal concepts, and domains. *Electronic Notes in Computer Science*, Vol. 83, 16 pages, 2003.
34. Guo-Qiang Zhang and G. Shen. Approximating concepts, Chu spaces, and information systems. In de Paiva and Pratt (Guest Editors), *Special Issue on Chu Spaces and Applications, Theory and Applications of Categories*, to appear.

FLIPP: A User-Preferred Interface for Knowledge Sharing

David J. Cox djcox@fuse.net

Abstract: A way to make logic patterns visually obvious for readers is demonstrated. A more detailed description at a web site is referred to.

Introduction: Traditional forms of describing the branching-rejoining logic patterns of complex systems are user-unfriendly. They include (1) linear forms which are unpatterned (text, syllogisms, formulas, computer program listings, and symbolic logic); (2) charts analogous to convoluted wiring diagrams (flow charts, tree structures).

The method presented here uses frameworks which avoid linear-only forms, symbols, signs, words, formula-forms, and which are subject-language independent. Experience with FLIPP comes from years of practical applications made at a large company where users universally preferred it to conventional methods of explaining and understanding.

Discussion: Researchers (see accompanying references) have long sought underlying patterns which would be free of specific language and topical dependencies. For example, Simon Polovina and Richard Hill address this in a working paper and in comments at the ICCS 2004 Conference on Conceptual Structures. They propose basing such patterns around "… *transactions* [which] *recognize the optimal relationships that occur through a mutually beneficial exchange in resources."* Examples of transaction patterns are listed from biology, physics, economics, business, ecology, psychology, and sociology.

Further, Frithjof Dau proposes *type-structures* and *token-structures* as part of diagrammatic systems for precise mathematical reasoning. He also cites researchers like Frege, Euler, Venn, Sowa, and Peirce who have proposed diagrammatic systems of representation that are not mathematical.

FLIPP (Format for Logical Information Planning and Presentation) is a way of chunking information into border frames like picture frames. These frames are directly self-connected in patterns which represent any logical relationships like: If...Then…Otherwise; Or; Not; Uncertain; Always. It has a single rule for readers: *start at the top, don't cross any vertical lines*. Subject information, which is within the connecting frames, can be in any language and in any form.

Since the arrangement of frames is so obvious, the learning curve for navigating through all FLIPP diagrams ('Explainers') is essentially flat.

Demonstration: The problem addressed by FLIPP is how to make sense out of information that is complex because its parts are not connected in logical form for use. Sentences, for example, may be grammatically acceptable but their always-linear form encodes any non-linear logic they hide.

The following demonstration compares the line-like form of text sentences with FLIPP patterns. Samples of both formats are shown below and compared. In both cases, the question being asked is, "What solutions (paths or scenarios) exist that make sense for any situation addressed by this system?"

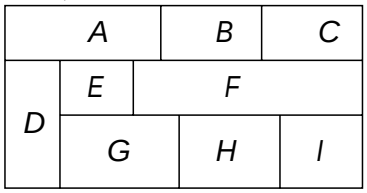
The Baseline: Text. Below are complex instructions in regular sentence format (text). The upper case letters represent different pieces or chunks of information about the subject. They can be in any format – words, symbols, drawings, etc.. There are 3 sentences forming 1 paragraph with 1 starting point and 1 ending point. *Only by reading and mentally figuring out the text*, can we learn there are 11 paths (logical arguments) like AFH, for example. There are 125 characters. This exemplifies text's logical complexity.

Text Format

If you start with A, as opposed to B or C, use D or use E then G, or F. If you choose F, then use G or H or I. If you start with either B or C, use F, then use G or H or I.

The FLIPP Method. The diagram below contains 9 characters, 7 % of the 125 used above in the text display. We see *without reading* that there are 9 frames which form 11 top-to-bottom explanation paths (scenarios or arguments), with 3 starting frames and 4 concluding frames. The content and logic are the same as the text passage but clearer. Notice that frames can overlap: The upper left frame is a member of 5 paths. Adding up the memberships of all 9 frames, we find that the board effectively represents 32 frames. Notice that we would see these 11 patterns even if no subject matter were shown. Notice also that the frames are not symbols since they represent only themselves. This is a vast simplification vs. flow charts and Modern Symbolic Logic among other tools of representation.

FLIPP Format (Read down. Don't cross verticals)



DEMO SUMMARY. We have looked at how the FLIPP format replaces text with simple, structured patterns of connection (logic). Subjective measures such as ease of explaining, learning, and improved team environment are described by user comments in the FLIPP web site.

Summary: Comparison of Conventional Text vs. FLIPP Format

Format	Text	FLIPP
Number of characters to be read	**125**	**9 (7% of 125)**
Number of scenario patterns visible by form	0	11
Starting points visible by form	1	3
Ending points visible by form	1	4
Number of frames effectively active	na	32
Verbal OR connections	7	0
Backtracking due to noncontiguous paths	a lot	none

Conclusions: The simplicity of FLIPP, hinted at here, is described at the web site referred to below. It documents the experience off users in about 100 actual applications in a large company starting in the late 1950s.

The site concludes that FLIPP frameworks are non-symbolic, non-linear, subject-independent, language-independent, are navigated easily, and increase teamworking. Pages of instructions are normally reduced in number by 90%. Families of scenarios for any systems are economically represented. Translation is vastly simpler since only 10% of the original subject material remains.

References

Barwise, Jon & Etchemendy, John – *Computers, Visualization, and The Nature of Reasoning.*
Cox, D. – http://www.flipp-explainers.org *Explanation by Pattern Means Massive Simplification*
Dau, Frithjof - *Types and Tokens for Logic with Diagrams.*
Felleisen, M.; Findler, R.; Flatt, M.; & Krishnamurthi, S. – *How to Design Programs* (MIT e-book).
Polovina, S. & Hill, R. - *Transactions Framework for Effective Enterprise Management.* (In process.)
Sowa, John F. - *Knowledge Representation: Logical, Philosophical, and Computational Foundations.*

Goal integration for service interoperability of engineering systems

Richard Dapoigny, Patrick Barlatier, Nacima Melal, Laurent Foulloy and Eric Benoit

Université de Savoie, ESIA Laboratoire d'Informatique, Systèmes, Traitement de l'Information et de la Connaissance
B.P. 806, F-74016 ANNECY Cedex, France
Phone: +33 450 096529 Fax: +33 450 096559 listic@esia.univ-savoie.fr

The design and implementation of service-oriented applications is known as *Service Engineering* (SE) and its prominent goal is to propose technology-independent models [5]. Such behavioral models using services are widely applied in the domain of web services (OWL-S) or telecommunications and more recently, in the engineering domain (e.g., automotive applications) [6]. SE must be able to capture descriptions of services which are machine-interpretable as well as user-understandable in order to facilitate the realization of Service-oriented tasks. A major challenge in this domain, is the design of services for use in distributed applications, provided that systems consist of dynamic sets of services.

The design process must support the aggregation of sub-services (i.e., to find a composition of services that achieve a given task), while the runtime behavior requires both to define the dynamic acquisition of service components and to check if their composition satisfies the requirements. Having formal requirements models makes the service composition problem manageable and gives rise to the need for a methodology for composing services. Motivated by the intelligent control of distributed physical systems, we investigate a modelling where the concept of service is divided in two parts, an intensional part described by a goal structure and an extensional one expressed by goal objects. On the one hand, we adopt the *teleological* approach in which the structure and behavior of the designed system are related to its goals [3]. In this approach, purposes can be ascribed to each component of the system and to achieve a global goal, we only have to describe how each sub-goal can be connected [4]. On the other hand, since the use of ontologies may favor the semantic heterogeneity, the ontological engineering is the prime requisite for information and service aggregation. Therefore, an important aspect of the design process is the ability, given users'specifications about the physical structure of the system and mathematical graph equations describing physical and control laws, to synthesize a goal-centered ontology. The domain ontology which is represented by a *part-of hierarchy*, addresses the problem of non-deterministic behavior through a goal decomposition. This results into an automatization of the design process together with a tractable type system where part-of hierarchies can be merged at run-time. The application focusses on the automatic acquisition of goal hierarchies in the engineering domain, based on *Formal Concept Analysis* (FCA). The goal structure involves first generic

goal types defined as pairs including an action verb (*intensional* part) and a physical role, then generic goal tokens expressed as triples requiring an action verb, a physical role and entities (*extensional* part). The sub-context dedicated to generic goals is automatically derived with respect to functional dependence highlighted by physical and/or control equations. A goal g_i influences functionally g_j if the only way to achieve g_j is to have already achieved g_i. A similar approach has been investigated in Software Engineering [1]. The resulting formal context uses generic goal tokens as objects and a *hierarchical scale* of both generic and composite goal tokens as attributes [7]. The conceptual hierarchy relies on the functional inclusion of goals and illustrates their functional dependencies in the λ-*Calculus* style. Finally, pruning techniques based on a recent work [2], extracts the goal part-of hierarchy. Given a goal request concerning the control of a physical system, the FCA-based mechanism is able to i) interactively construct a context of basic goals from user inputs and local databases ii) construct a hierarchical scale including a set of generic goals and goal(s) request(s) expressed by user, which is translated into a goal hierarchy.

In order to work together in distributed applications without the need for human interaction, services must be able to merge their information systems in a sound manner. This is known as the problem of distributed service composition. Our work in progress addresses this problem through a hierarchical structure with two layers. The top layer includes a part-of hierarchy of goals where the intensional aspect is predominant. The sub-layer maps the goal hierarchy to an action hierarchy where each action is determined with pre-conditions related to each goal. It results in an action classification where instances are temporal and ordered through the *Event Calculus* formalism. The event ordering provides the resulting planning for the considered system. Once the structure is well defined, we plan to explore the *IF-based* approach at the goal level where the negotiation of distributed goals can be solved in a sound way.

References

1. Ball T.: The concept of dynamic analysis. in Procs. of the ACM SIGSOFT Symposium on the Foundations of Software Engineering (1999) 216–234
2. Cimiano P., Hotho A., Stumme G., Tane J.: Conceptual knowledge processing with formal concept analysis and ontologies. ICFCA LNAI **2961** Springer (2004) 189–207
3. Dapoigny R., Benoit E., Foulloy L.: Functional Ontology for Intelligent Instruments. Foundations of Intelligent Systems. LNAI 2871 (Springer) (2003) 88–92
4. Lind M.: Modeling Goals and Functions of Complex Industrial Plant. Journal of Applied Artificial Intelligence **8** (1994) 259–283
5. Quantel D., Dijkman R., Sinderen M.V.: Methodological support for service-oriented design with ISDL, 2nd Int. Conf. on Service Oriented Computing (2004)
6. Schätz B.: Towards Service-based Systems Engineering: Fomalizing and checking service specifications, Tech. Report TUMI-0602 München (2002)
7. Stumme G.: Hierarchies of conceptual scales. Procs. of the Workshop on Knowledge Acquisition, Modeling and Managenent **2** (1999) 78–95

Propositional Theorem Prover for Peirce-Logic

David P. Hodgins and Pavel Kocura

Department of Computer Science, Loughborough University P.Kocura@lboro.ac.uk

Abstract. This paper briefly surveys the theory and implementation of a theorem proving system for Peirce's propositional Existential Graphs(EG).

Introduction. Over the past 2 years we developed a propositional theorem prover for Alpha EG. It uses a linear EG syntax[1] and the EG inference rules[2], to test thorem validity[3].

The system is implemented as a distributed automated propositional EG inference engine server that: a) Proves relations between premises and conclusions (theorems), b) Checks user entered proofs, c) Translates between classical and EG propositional logic notations, d) Automatically draws existential graphs.

One of the motivations for the system was to improve upon the performance of the Peirce logic components of a CGs theorem proving system developed by Heaton[4]. The current system comprises:

Translator. The system translates between Peirce and conventional notations by replacing the logical connectives of one with those of the other, and inserting symbols around the antecedents and consequents of the operators where necessary. This replacement strategy is shown in tables 1 and 2. Equivalence requires special treatment.

Table 1. Peirce to conventional

Peirce	)(	ab	(	)
Conventional	)∧(	a∧b	~(	)

Table 2. Conventional to Peirce

Conventional	a∧b	a∨b	a⇔b	a⇒b
Peirce	ab	((a)(b))	(a(b))(b(a))	(a(b))

Inference Engine. Peirce's inference rules are truth preserving[5]. However, a *false* graph can be transformed into a true graph[5], therefore conclusions can only be drawn by transforming graphs to *false*,"()", as a valid graph cannot be transformed into a unsatisfiable graph. The inference engine combines premises and negated conclusions using conjunction. The system uses Iteration along with the three reducing rules: Double Negation Elimination (DNE), Deiteration and Erasure. Erasure and Iteration are combined as their only purpose in reducing a graph to *false* is to split a complex structure into two simpler ones. For example, "((ab)(c))" would be simplified to "((a)(c)) ((b)(c))". The reason for not simply discarding one of the variables is to ensure that unsatisfiable graphs are always shown as such. If variables were simply discarded then this would not be possible. This strategy has the following stages:

1. (a) Iteratively perform all possible DNEs on the graph;
2. (a) Iteratively perform all possible Deiterations on the graph;
   (b) If at least one Deiteration was performed in step 2.a, then go back to step 1 (as double negations may have be generated);
3. (a) Perform a single combined Erasure/Iteration operation, then loop back to stage 1;

At the end of part *a* of each stage the algorithm checks if the graph is unsatisfiable, and finishes if it is. It always finishes, and if it does without finding the graph to be unsatisfiable then it isn't. The results are then interpreted and displayed in either linear notation or drawn as existential graphs.

Interface. The systems graphical interface is accessed via a web page server. This can be accessed by multiple users, or users can transparently access different servers. Each user's theorems can be distributed amongst different servers for improved performance.

Graph drawing. The automated EG drawing tool parses an existential graph in linear form, from left to right, and replaces each open and close bracket with html open and close table tags. Cuts are represented as rectangles, and displayed with either rounded or square corners, depending on user preference.

Testing and Evaluation. The test data was provided by translating from conventional notation: the rules of a conventional propositional logic system, various proofs including each of the steps used in them, and prime number examples generated from HeerHugo[6]. So far, extensive testing of the system hasn't revealed any inconsistencies.

Performance. Deiteration, DNE, and Erasure operations run in O(n) time. However, whilst the first two reduce the size of the graph, the latter increases it. The Erasure operation is thus the key element in deciding the performance of the system, and therefore various heuristics are used to optimise it. One such heuristic is to stop the inference process once an empty cut is found on the sheet of assertion, instead of erasing everything except the cut. Currently the system can only handle inputs of a maximum size of 30 KB, assuming the log of the input size approximates the maximum number of cuts that enclose a variable.

This system shows that the Alpha EG inference rules can be used on their own to automatically prove theorems, and provides the basis for a predicate system currently being developed.

References

1. http://pages.cpsc.ucalgary.ca/∼kremer/courses/CG/8.2.
2. Zeman. (2002) http://web.clas.ufl.edu/users/jzeman/graphicallogic/alpha.htm.
3. http://www.oursland.net/aima/propositionApplet.html.
4. Heaton, J.: Goal driven theorem proving using conceptual graphs and Peirce logic. PhD thesis, Loughborough University (1994)
5. Roberts, D.D.: The existential graphs. Computers Math. Applic. **Vol. 23** (1992) pp. 639–663
6. Groote, J.F. (1998) http://www.win.tue.nl/∼jfg/heerhugo.html.

Generating Diseases Concept and Antonyms

Christian Jacquelinet[1], Anita Burgun[2]

1. Département Médical et Scientifique,
Etablissement français des Greffes, 1, avenue du Stade de France,
93321 Saint-Denis-La plaine Paris, France
2. Laboratoire d'informatique médicale,
Université de Rennes 1, 35000, Rennes, France.
christian.jacquelinet,anita.burgun@univ-rennes1.fr

1 The Medical Context

The Etablissement français des Greffes maintains a decisional information system (EfG-IS) used to aggregate information, to perform evaluation studies and to propose strategic decision in the field of organ failure public health policies. The use of several terminologies is managed through a terminological component providing ontology editing functionalities and tools for mapping terminological standards onto a diseases ontology. The foundations of EfG terminological system have been reported in [1]. A first component supports a medical ontology centered on diseases with relevant editing functionalities. A second component comprises terminologies to integrate and relevant mapping functionalities. The ontology is supported by a concept type lattice with multiple inheritance. Each concept can be related to a Description Model: a set of conceptual relations linking the concept to refining concepts. A Description Model behaves like a meta-model comprising all potential slots describing a class and its sub-classes. Some "slots" defining the class are inheritable by hyponyms, some are local and not inheritable. A new concept is always generated from a current concept as a brother or as an hyponym of the current concept, using a Description Model as a template. At the generation of the new concept, slots inherited of the current concept can be over-specified, some new slots will be instantiated by specifying a refining concept, some won't be instantiated. The description of each concept by a conceptual structure (its Description Graph) makes the ontology formal: the *is_a* relation is derived from the definitional conceptual structure. It is not based on type labels, their lexical meanings and an arbitrary type lattice.

2 Improving the Initial Description Model

Previous works dealing with the semantic structure of medical terms, led us to build an initial Disease Description Model comprising six main slots relating a given disease to its discriminating characteristics, thus beginning with a: has_for_discriminating completed with : _Location, _Evolution, _Semiology, _Etiological-Process, _Lesion, and Occurs-during/after. The experimental use of this model to integrate medical terms showed us that it required to be more sophisticated to support opposition principles and antonyms. Negation is indeed

a common phenomenon in medical terminologies. Some terms are defined by opposition to the others (e.g.: non otherwise specified, non Hodgkin's Lymphoma), some comprise negative slots (e.g: non genetic hemochromatosis, non A non B hepatitis, non glomerular lesion). To represent such entities, we found that an efficient solution was to render more explicit the type and the quantification of refining concepts as above:

[**Disease**]-
(00: has for EfG label)→*["disease"]*
(10: has a location quantified as)→[*]
(11: has for location's type)→[**Body_Component**]
(20: has an etiological process quantified as)→[*]
(21: has for discriminating etiological process type)→[**Etiological Process**]
(22: has for etiological process agent's type)→[**Etiological Process Agent**]
(30: has an evolution quantified as)→[*]
(31: has for discriminating evolution's type)→[**Evolution**]
(40: has a discriminating semiolgy quantified as)→[**Quantifier**]
(41: has for discriminating sign's type)→[**Semiology**]
(50: has a causative disease quantified as)→[**Quantifier**]
(51: has a discriminating causative disease whose type is)→[**Disease**]
(60: has a lesion quantified as)→[**Quantifier**]
(61: has for discriminating lesion's type)→[**Lesion**]

Any disease has intrinsically an existing location, an existing evolution profile or an existing etiological process. Indeed, there is no disease without an involved body component, evolution or etiological process, even if it is unknown (cryptogenic disease) or unspecified. In contrast, the existence of a discriminating sign, causative disease or lesion is to specify: a disease can be discriminated according to its semiology, a disease can be primitive or secondary, a disease can be organic or functional.

3 Discussion

This position paper summarizes the impact of antonymy in diseases terminologies and its consequences on the Disease Description Frame we use to generate new concepts within the hierarchy by creating a new subtype from a current concept according (or not) to the current concept schema. Constraints due to the disease's schema in combination with the anatomical, etiological process, semiology and lesions type hierarchy precludes artificial and excessive disease's concept generation. This procedure appears as a first step toward a generative ontology for diseases, in contrast with classifying approaches used in Description Logics where all concepts are to be described before their automated classification and in contrast with arbitrary type lattices whose consistency is difficult to assess. Such an approach permits the mapping of various medical terminologies, which is a first step to medical information sharing.

References

1. Jacquelinet C, Burgun A, Delamarre D, Strang N, Djabbour S, Boutin B, Le Beux P. Developing the ontological foundations of a terminological system for end-stage diseases, organ failure, dialysis and transplantation. Int J Med Inf. 2003 Jul;70(2-3):317-28.

Representing relations between individuals and universals

Gábor Rédey

Hungarian Atomic Energy Authority, H-1036 Budapest, Fényes A. u. 4., Hungary
redeyg@iif.hu

Expressing individuality and universality in English *Unity* and *multiplicity* is a pair of concepts reflecting probably the most essential structure of existence, and are likely among the most ancient abstractions in human thinking. Formal ontology, a description of concepts and their relationships for communicating agents, also focuses on the topic how highly general, domain independent categories can be developed. According to this approach, the most general ontologically basic categories, the entities the real world are partitioned into, are *individuals* and *universals* [1]. The subsequent analysis tries to show that this dichotomic partition is relative to some extent.

Natural languages often use the same expression or a derivative phrase to distinguish a collection of things having the same property from the name of that property e.g. *black – the colour black, long – length, swim – swimming, John (a complexity) – John (an individual), European – Europe* etc.

$$\begin{array}{ll} \textit{A black ball rolls.} & \textit{Black is a colour.} \\ \textit{John walks.} & \textit{The hand is John's.} \end{array} \tag{1}$$

"Black" is used in (1) in two different meanings: as an adjective referring to the collection of black things, and as a noun naming the property of being black. The other two sentences demonstrate the reverse function, assigning properties to names. Although "John" is an individual name in both sentences in (1), the second sentence asserts a *part of* relation treating "John" like a property.

Consequently, the human way of thinking is twofold. Reality is generally considered as being built up from, or partitioned into individuals, however, individuals might also be decomposed into their constituents, and similarly, collections of individuals can also be considered as a whole.

Representing individuality and universality in iCTRL Although many knowledge representation systems have been published since the mid-80s, intensional text representation language (iCTRL) is utilised which models the basic natural language relations rather consistently [2].

$$\begin{array}{ll} \textit{roll } x, \exists\ x, \textit{ black } x, \textit{ ball } x. & \textit{colour } x, \langle \textit{black} \rangle x. \\ \textit{walk } x, \textit{ John } x. & [(\forall\ x)y, \textit{ John } y], \forall\ x : z, \textit{ hand } x. \end{array} \tag{2}$$

The formula *walk x*, *John x*. expresses the same content by, the same subject-predicate relation, as the English sentence, or in this particular case first-order logic does: there is an individual, *John x*, which is included in the extension of the predicate, *walk x*. The same informal description of syntax and semantics is sufficient to interpret the formula rendering the other simple sentence. In this instance the compound subject extension, $\exists$ *x*, *black x*, *ball x*, is supposed to be included in the extension of the predicate: *roll x*. The compound extension of

subject is a successive intersection of the extensions of *ball* x and *black* x, then the result and the extension of $\exists\ x$. The existential quantifier is represented by a predicate extending to the non-empty parts of the universe of discourse.

The other two statements exceed the frames of classical first-order logic, requiring a modal extension of the representation language [2]. Carnap's definition of meaning (intension) is formalised by the following syntax. Provided that α denotes an arbitrary formula and x a context variable parameter referring to the possible worlds, $(\alpha)x$ designates the function mapping any possible world index, x, to the current extension/truth-value of α.

Since $\forall\ x$ denotes the universal quantifier, with an extension that coincides with the whole universe of discourse, $(\forall\ x)y$ can represent any possible worlds, restricted by a certain respect. Hence, the individual *John* x can act as a restriction: $(\forall\ x)y$, *John* y, refers to the corresponding parts of the universe of discourse which effectively compose John. The variable parameter $x : z$ in (2) points back to the hand that is supposed to have been mentioned previously in context by the variable z [3].

Referring to properties by their naming needs some new device in the language. Natural languages form these names generally by nominalisation, while logical languages do the same by quotation. Quoting "black", however, seems rather ambiguous at first sight. One can state that something is black by the predicate *black* x, but its quoted form, $\langle$*black* $x\rangle y$, which is an individual, represents the class name of black individuals. Still, our statement is not of this sort, it tells something about the colour black itself: $\langle$*black*$\rangle x$. Consequently: *colour* x, $\langle$*black*$\rangle x$. is the correct representation of the sentence in (2).

Conclusion Formalising concepts need a fine distinction between individuals and universals. Human reflection resorts to both of them, however, natural language word-classes indicate that some things are more often referred to as properties or relations while others as individuals. Providing a concept p is common as a property or a proper name N is common for naming an individual object, then its nominalisation which refers to that multiplicity as a whole or a formal common name can be associated, respectively:

$$p\ x \rightarrow \langle p\rangle x \rightarrow (\forall\ x)y, \langle p\rangle y \quad N\ x \rightarrow (\forall\ x)y, N\ y \rightarrow \langle(\forall)y, N\ y\rangle x. \qquad (3)$$

Conceiving as a universal or an individual is not permanent, the same thing can always be considered from both aspects depending on which part of this dichotomy is emphasised. Natural languages are able to express both angles, and following this capability an exact formal representation of that phenomenon has been tried to present here.

References

1. Degen, W.; Heller, B.; Herre, H.; Smith, B.: GOL: Towards an Axiomatized Upper-Level Ontology, in Smith, B.; Guarino, N. eds. Proc. of the 2nd Int. Conf. on Formal Ontologies and Information Systems, Ogunquit, Maine, USA, 2001.
2. Rédey G.: iCTRL: Intensional conformal text representation language, *Artificial Intelligence*, **109/1-2**(1999)33-70.
3. Rédey G., Neumann A.: Self-referencing languages revisited, in Büchel, G.; Klein, B.; Roth-Berghofer, T. eds.: Proc. of the 1st Workshop on Philosophy and Informatics, Cologne, Germany, (2004), 105-111.

Author Index