Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

Document
D-94-10

International Workshop on Description Logics

Bonn, May 28/29, 1994

Organizers:

**Franz Baader, Maurizio Lenzerini,
Werner Nutt, Peter F. Patel-Schneider**

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

Postfach 20 80
67608 Kaiserslautern, FRG
Tel.: (+49 631) 205-3211/13
Fax: (+49 631) 205-3210

Stuhlsatzenhausweg 3
66123 Saarbrücken, FRG
Tel.: (+49 681) 302-5252
Fax: (+49 681) 302-5341

International Workshop on Description Logics

Organizers: Franz Baader, Maurizio Lenzerini, Werner Nutt, Peter F. Patel-Schneider

DFKI-D-94-10

This work has been supported by a grant from The Federal Ministry for Research and Technology (FKZ ITW-9201).

ISSN 0946-0098

International Workshop on Description Logics

Franz Baader
Lehr- und Forschungsgebiet Theoretische Informatik
RWTH Aachen
Aachen, Germany
baader@informatik.rwth-aachen.de

Maurizio Lenzerini
Dipartimento di Informatica e Sistemistica
Universita di Roma, "La Sapienza"
Rome, Italy
lenzerini@assi.dis.uniroma1.it

Werner Nutt
German Research Center for Artificial Intelligence
Saarbrücken, Germany
nutt@dfki.uni-sb.de

Peter F. Patel-Schneider
AT&T Bell Labs
Murray Hill, NJ, U.S.A.
pfps@research.att.com

This collection of papers forms the permanent record of the 1994 Description Logic Workshop, that was held at the Gustav Stresemann Institut in Bonn, Germany on 28 and 29 May 1994, immediately after the Fourth International Conference on Principles of Knowledge Representation and Reasoning. The workshop was set up to be as informal as possible, so this collection cannot hope to capture the discussions associated with the workshop. However, we hope that it will serve to remind participants of their discussion at the workshop, and provide non-participants with indications of the topics that were discussed at the workshop.

The workshop consisted of seven regular sessions and one panel session. Each regular session had about four short presentations on a single theme, but also had considerable time reserved for discussion. The themes of the sessions were Foundations of Description Logics, Architecture of Description Logics and Description Logic Systems, Language Extensions, Expanding Description Logics, General Applications of Description Logics, Natural Language Applications of Description Logics, Connections between Description Logics and Databases, and the Future of Description Logics and Description Logic Systems.

The session on Foundations of Description Logics concentrated on computational properties of description logics, correspondences between description logics and other formalisms, and on semantics of description logics, Similarly, there is discussion on how to develop tractable desription logics, for some notion of tractable, and whether it is useful to worry about achieving tractability at all.

Several of the participants argued in favour of a very expressive description logic. This obviously precludes tractability or even decidability of complete reasoning. Klaus Schild proposed that for some purposes one could employ "model checking" (i.e., a closed world assumption) instead of "theorem proving," and has shown that this is still tractable for very large languages. Maurizio Lenzerini's opinion was that it is important to have decidable languages. Tractability cannot be achieved in several application areas because there one needs very expressive constructs: e.g., axioms, complex role constructors, and cycles with fixed-point semantics. For Bob MacGregor, not even decidability is an issue since he claims that Loom's incomplete reasoner is sufficient for his applications.

The discussion addressed the question of whether there is still need for foundations, and whether the work on foundation done until now really solved the problems that the designers of early DL systems had. Both questions were mostly answered in the affirmative, with the caveat that new research on foundations should make sure that it is concerned with "real" problems, and not just generates new problems.

In the session on Architecture of Description Logics and Description Logic Systems the participants considered different ways of putting together description logics and description logic systems. One way of doing this is to have a different kind of inference strategy for description logics, such as one based on intuitionistic logics or one based directly on rules of inference—thus allowing variant systems. Another way of modifying description logic systems is to divide them up in different ways, such as making a terminology consist of a schema portion and a view portion. Some discussion in this session concerned whether architectures should be influenced by application areas, or even by particular applications.

There was considerable discussion at the workshop on how Description Logics should be extended or expanded to make them more useful. There are several methods to do this. The first is to extend the language of descriptions, e.g., to represent n-ary relations, temporal information, or whole-part relationships, all of which were discussed at the workshop. The second is to add in another kind of reasoning, such as default reasoning, while still keeping the general framework of description logic reasoning. The third is to incorporate descriptions or description-like constructs in a larger reasoner, such as a first order reasoner. This was the approach taken in OMEGA and is the approach being taken in the Loom project.

There have been many extensions of the first two kinds proposed for description logics, including several presented at the workshop. One question discussed at the workshop was whether these extensions fit in well with the philosophy of description logic. Another question was whether the presence of many proposals for extensions means that description logics are easy to expand, or that description logics are inadequate representation formalisms?

The general consensus was that description logics adequately capture a certain kind of core reasoning and that they lend themselves to incorporation with other kinds of reasoning. Care must be taken, however, to keep the extended versions true to the goals of description logics.

The sessions on Applications of Description Logics had presentations on applications of description logics in various areas, including configuration, tutoring, natural language processing, and domain modeling. Most of these applications are research applications, funded by government research programs. There was discussion of what is needed to have more fielded applications of description logics.

The session on Connections between Description Logics and Databases considered three kinds of connections between Description Logics and Databases:

1. using Description Logics for expressing database schemas, including local schemas, integrated schemas, and views, integrity constraints, and queries;
2. using Description Logic reasoning for various database-related reasoning, including schema integration and validation, and query optimization, and query validation and organization; and
3. making Description Logic reasoners more like Database Mangagement Systems via optimization.

All three of these connections are being actively investigated by the description logic community.

The panel session on the Future of Description Logics and Description Logic Systems discussed where the future of descripion logics will lie. There seems to be a consensus that description logics must forge tighter connections with other formalisms, such as databases or object-oriented systems. In this way, perhaps, description logics will find more real applications.

Contents

Foundations of Description Logics

On the Correspondence between Description Logics and Logics of Programs 1
Giuseppe De Giacomo, *Maurizio Lenzerini*, Università di Roma "La Sapienza"

Which Semantics for Individuals in the TBox? 5
Andrea Schaerf, Università di Roma "La Sapienza"

Tractable Reasoning in a Universal Description Logic 9
Klaus Schild, German Research Center for Artificial Intelligence

Integrating Terminologies into F(rames)-Logic Knowledge Bases 14
Mira Balaban, Ben-Gurion University of the Negev

Architecture of Description Logics and Description Logic Systems

YAEF: Yet Another Epistemic Formalism 19
Philipp Hanschke, sd&m GmbH & Co. KG

Refining the Structure of Terminological Systems: Terminology = Schema + Views 22
M. Buchheit, German Research Center for Artificial Intelligence; *F. M. Donini*, Università di Roma "La Sapienza"; *W. Nutt*, German Research Center for Artificial Intelligence; *A. Schaerf*, Università di Roma "La Sapienza"

Flexible Inference Strategies for DL Systems 27
J. Joachim Quantz, *Guido Dunker*, Technische Universität Berlin; *Véronique Royer*, Onera DES/SIA

Intuitionistic Query Answering in Description Bases 31
Véronique Royer, Onera DES/SIA; *J. Joachim Quantz*, Technische Universität Berlin

Language Extensions

Homogeneous Concepts in a Temporal Description Logic 36
Alessandro Artale, *Enrico Franconi*, Knowledge Representation and Reasoning Lab., IRST; *Claudio Bettini*, University of Milano

Extending Standard Description Logics 42
Patrick Lambrix, Linköping University

CLASSIC Extended with Physical Whole-part Relations 45
Piet-Hein Speel, University of Twente; *Peter F. Patel-Schneider*, AT&T Bell Laboratories

Using Description Logics for Software Reuse and Case-based Reasoning 51
John Yen, *Swee Hor Teh*, *Xiaoqing Liu*, Texas A&M University

Useful Defaults in Description Logics 55
W. Nutt, German Research Center for Artificial Intelligence; *Peter F. Patel-Schneider*, AT&T Bell Laboratories

Expanding Description Logics

Beyond Description Logics 58
Robert MacGregor, USC/Information Sciences Institute

A Relation-Based Description Logic 62
Enrico Franconi, *Vania Rabito* Knowledge Representation and Reasoning Lab., IRST

Studies about the Integration of Classification-Based Reasoning and Object-Oriented Programming 67
Amedeo Napoli, CRIN CNRS—INRIA Lorraine

General Applications

Where are all the Users? 70
Deborah L. McGuinness, AT&T Bell Labs
Description Logics as Core of a Tutoring System 73
S. Bergamaschi, Università di Modena—CIOC-CNR; *S. Lodi*, *C. Sartori*, Università di Bologna—CIOC-CNR
Using Description Logics in Intelligent Tutoring Systems 75
Gunnar Teege, Technische Universität München

Natural Language Applications

Description Logics for Natural Language Processing 79
D. Fehrer, *U. Hustadt*, *M. Jaeger*, *A. Nonnengart*, *H. J. Ohlbach*, *R. A. Schmidt*, *Ch. Weidenbach*, *E. Weydert*, Max-Planck-Institut für Informatik
Lexical Semantics, Description Logics and Natural Language Systems 84
Peter Forster, Universität Stuttgart
Using Description Logics for Disambiguation in Natural Language Processing 87
J. Joachim Quantz, *Birte Schmitz*, *Uwe Küssner*, Technische Universität Berlin
Construction of a Domain Model for Speech Translation 93
Manfred Gehrke, Siemens AG

Connection to Databases

Description Logics for Querying Databases 95
Alex Borgida, Rutgers University
Subsumption for Semantic Query Optimization in OODB 97
Domenico Beneventano, Università di Bologna; *Sonia Bergamaschi*, Università di Modena; *Claudio Sartori*, Università di Bologna
Constraints in Complex Object Database Models 101
D. Beneventano, Università di Bologna—CIOC-CNR; *S. Bergamaschi*, Università di Modena—CIOC-CNR; *S. Lodi*, *C. Sartori*, Università di Bologna—CIOC-CNR
Advantages of using a Terminological system for integrating Relational Databases 104
J. M. Blanco, *A. Illarramendi*, *A. Goñi*, *J. Bermúdez*, Universidad del País Vasco
Persistent Maintenance of Object Descriptions using BACK 108
Albrecht Schmiedel, Technische Universität Berlin

General Papers and Position Papers

Position Paper for Description Logic Workshop 110
Lin Padgham, Linköping University
Knowledge Formal Specifications for Formal Verification: a Proposal based on the Integration of Terminological Logics with Different other Logical Formalisms 112
Marie-Christine Rousset, Universite Paris-Sud
Generalized Feature Graphs: A Kernel for Description Logic Implementations 116
Gunnar Teege, Technische Universität München

On the correspondence between description logics and logics of programs (position paper)

Giuseppe De Giacomo and Maurizio Lenzerini
Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, 00198 Roma, Italia
e-mail: [degiacom,lenzerini]@assi.dis.uniroma1.it

1 Introduction

In the last decade, many efforts have been devoted to an analysis of the epistemological adequacy, and the computational effectiveness of Description Logics (DLs). In particular, starting with [4], the research on the computational complexity of the reasoning tasks associated with DLs has shown that in order to ensure decidability and/or efficiency of reasoning in all cases, one must renounce to some of the expressive power [17; 19; 20; 12; 13; 11]. These results have led to a debate on the trade-off between expressive power of representation formalisms and worst-case efficiency of the associated reasoning tasks. This issue has been one of the main themes in the area of DLs, and has led to at least four different approaches to the design of knowledge representation systems.

- In the first approach, the main goal of a DL is to offer powerful mechanisms for structuring knowledge, as well as sound and complete reasoning methods (not necessarily realized by means of terminating procedures), while little attention has to be paid to the (worst-case) computational complexity of the reasoning procedures. Systems like OMEGA [1], LOOM [18] and KL-ONE [6], can be considered as following this approach.
- The second approach advocates a careful design of the DLs so as to offer as much expressive power as possible while retaining the possibility of sound, complete, and efficient (often polynomial in the worst case) inference procedures. Much of the research on CLASSIC [5] follows this approach.
- The third approach, similarly to the first one, advocates very expressive languages, but, in order to achieve efficiency, accepts incomplete reasoning procedures. There is no general consensus on what kind of incompleteness is acceptable. Perhaps, the most interesting attempts are those resorting to a non-standard semantics for characterizing the form of incompleteness [22; 3; 14].
- Finally, the fourth approach is based on what we can call "the expressiveness and decidability thesis", and aims at defining DLs that are both very expressive and decidable, i.e. designed in such a way that sound, complete, and terminating procedures exist for the associated reasoning tasks. Great attention is given in this approach to the complexity analysis for the various sublogics, so as to devise suitable optimization techniques and to single out tractable subcases. This approach is the one followed in the design of KRIS [2].

This position paper presents an ongoing research project that adheres to the fourth approach, and aims at both identifying the most expressive DLs with decidable associated decision problems, and characterizing the computational complexity of the corresponding reasoning problems.

2 The expressiveness and decidability thesis

In order to clearly characterize the expressiveness and decidability thesis, let us point out that by "very expressive DL" we mean the following:

1. The logic offers powerful mechanisms to describe/render classes.
   - It includes concept constructs for boolean connectives $C \sqcap D$, $C \sqcup D$, $\neg C$, and existential and universal quantifications $\forall R.C$, $\exists R.C$.
   - It may include role constructs for inverse role R^-, chaining of roles $R \circ Q$, union of roles $R \sqcup Q$, and the identity role projected on a concept *id*(C). It may also include *functional restrictions*[1] on atomic roles $(\leq 1\ P)$ and possibly on their inverse $(\leq 1 P^-)$, and (qualified) *number restrictions*[2] again on atomic roles $(\leq\ n\ P)$, $(\geq\ n\ P)$ $((\leq\ n\ P.C), (\geq\ n\ P.C))$ and possibly on their inverse $(\leq\ n\ P^-)$, $(\geq\ n\ P^-)$ $((\leq\ n\ P^-.C), (\geq\ n\ P^-.C))$.
   - It possibly includes suitable mechanisms to describe concepts which are not first-order defin-

[1]Functional restrictions impose the functionality of a role in the context of a concept.

[2](Qualified) number restrictions state the minimum and/or the maximum number of instances of a role (restricted by means of concept) in the context of a concept.

able. The most common example of such mechanisms is a construct for the transitive closure of roles R^*. More sophisticated ones are those to capture *inductively* and *co-inductively* defined classes (i.e. classes defined as the smallest class such that ..., or the biggest class such that ...).

2. The logic provides suitable means for imposing mutual dependencies among concepts (TBox). The basic mechanisms for supporting this feature are *inclusion assertions* of the form $C \sqsubseteq D$ where C, D can be any concepts, stating that C is to be interpreted as a subset of D. Observe that the assumption of acyclicity of TBoxes is not enforced. Indeed, with this assumption, the power of inclusion assertions vanishes.
3. The logic allows one to assert properties of individuals (ABox) in term of *membership assertions.* These can be of two forms: $a : C$, stating that an individual a is an instance of the concept C, and $a\ R\ b$, stating that the individual a is related via the role R to the individual b.

Note that, the presence of inverse of roles allows the logic to subsume most of the frame-based representation systems, semantic data models and object-oriented database models proposed in the literature. The constructs for functional restrictions on both atomic roles and their inverse greatly enhance the power of the logic, e.g. they allow the logic to correctly represent n-ary relations among classes. Note, also, that the ability to describe non-first-order definable classes is often needed, for example to model the most common data structures used in computer science, such as LISTs and TREEs.[3]

3 The correspondence between DLs and logics of programs

Two main approaches have been developed following the "expressiveness and decidability thesis". The first approach relies on the tableaux-based technique proposed in [25; 12], and led to the identification of a decision procedure for a logic which fully covers points (2) and (3) above, but only partially point (1) in that it does not include the construct for inverse roles [7], and has no mechanism to describe concepts that are not first-order definable.

The second approach is based on the work by Schild [23], which singled out a correspondence between some DLs of the kind described above and a certain class of logics of programs: the Propositional Dynamic Logics (PDLs), which are modal logics specifically designed for reasoning about program schemes. The correspondence is based on the similarity between the interpretation structures of the two logics: at the extensional level, objects in DLs correspond to states in PDLs, whereas connections between two objects correspond to state transitions. At the intensional level, classes correspond to propositions, and roles corresponds to programs. The correspondence provides an invaluable tool for studying very expressive DLs. Indeed, it makes it clear that reasoning about assertions on classes is equivalent to reasoning about dynamic logic formulae (e.g., logical implication wrt a TBox, in any of the above logics, is equivalent to satisfiability of a specified dynamic logic formula), so that the large body of research on decision procedures for PDLs (see [16] for references) can be exploited in the setting of DLs.

However, in order to fully exploit this correspondence, at least three problems left open in [23] need to be solved, namely, how to fit functional restrictions on both atomic roles and their inverse, number restrictions, and assertions on individuals, respectively, into the correspondence. Note that these problems refer to points (1) and (3) above.

The work we have been carrying out on this subject [9; 10] has the explicit goal of providing suitable solutions to the above problems. Regarding point (1), we have investigated the following DL, named $\mathcal{CIF}$:

$$\begin{array}{lcl} C & ::= & A \mid C_1 \sqcap C_2 \mid C_1 \sqcup C_2 \mid \neg C \mid \forall R.C \mid \exists R.C \mid \\ & & (\leq 1\ P) \mid (\leq 1\ P^-) \\ R & ::= & P \mid R_1 \sqcup R_2 \mid R_1 \circ R_2 \mid R^* \mid id(C) \mid R^- \end{array}$$

where A and P denote the generic atomic concept and role respectively. The main feature of $\mathcal{CIF}$ is the presence of functional restrictions on both atomic roles and their inverse. The decidability of the corresponding PDL, named $\mathcal{DIF}$, was not known. We have proved that satisfiability in $\mathcal{DIF}$ and logical implication for $\mathcal{CIF}$-TBoxes are EXPTIME-complete problems. The above decidability/complexity result holds also for $\mathcal{CIN}$ ($\mathcal{DIN}$), obtained from $\mathcal{CIF}$ ($\mathcal{DIF}$) by including the constructs for qualified number restrictions on both the atomic roles and their inverse. Moreover it is possible to polynomially encode n-ary relations among concepts in such logics. With respect to point (3), we have proved that for knowledge bases (TBox and ABox) expressed in two sublanguages of $\mathcal{CIF}$, namely $\mathcal{CI}$ (no functional restrictions) and $\mathcal{CF}$ (no inverse roles), satisfiability and logical implication are EXPTIME-complete. It is worth noting that, from the PDLs' point of view, an ABox has a natural counterpart: it can be regarded as a specification of partial computations.

Recently, both Schild and ourselves [24; 8] have pointed out that the correspondence between DLs and PDLs, can be extended to another logic of programs called (modal/propositional) mu-calculus [15] (see [26] for more references). This logic has the salient property of including explicit constructs for least and greatest fixpoints of formulae, which makes it more expressive than comparable PDLs. Indeed, the presence of the fixpoint constructs enables the logic to fully express inductive and co-inductive definitions, as well as to model, in a single framework, terminological cycles interpreted according to Least Fixpoint Semantics, Greatest Fixpoint Semantics, and Descriptive Semantics (see [21]). We have studied an

[3]In fact to correctly represent these data structures, the logic must also include constructs for inverse roles and for functional (or number) restrictions on both atomic role and their inverse.

extension of mu-calculus that includes qualified number restrictions on atomic roles, showing that satisfiability is EXPTIME-complete for it. Currently we are developing a method to reason with knowledge bases (ABox and TBox) expressed in a DL corresponding to mu-calculus extended with functional restrictions on atomic roles. We conclude remarking that a mu-calculus with a construct corresponding to inverse roles, though of great interest, has not been studied yet.

References

[1] Giuseppe Attardi and Maria Simi. Consistency and completeness of omega, a logic for knowledge representation. In *Proc. of the Int. Joint Conf. on Artificial Intelligence IJCAI-81*, pages 504–510, 1981.

[2] Franz Baader and Bernhard Hollunder. A terminological knowledge representation system with complete inference algorithm. In *Proc. of the Workshop on Processing Declarative Knowledge, PDK-91*, Lecture Notes in Artificial Intelligence, pages 67–86. Springer-Verlag, 1991.

[3] Alexander Borgida and Peter F. Patel-Schneider. A semantics and complete algorithm for subsumption in the CLASSIC description logic. Submitted for publication, 1993.

[4] Ronald J. Brachman and Hector J. Levesque. The tractability of subsumption in frame-based description languages. In *Proc. of the 4th Nat. Conf. on Artificial Intelligence AAAI-84*, pages 34–37, 1984.

[5] Ronald J. Brachman, Deborah L. McGuinness, Peter F. Patel-Schneider, Lori Alperin Resnick, and Alex Borgida. Living with CLASSIC: when and how to use a KL-ONE-like language. In John F. Sowa, editor, *Principles of Semantic Networks*, pages 401–456. Morgan Kaufmann, 1991.

[6] Ronald J. Brachman and James G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2):171–216, 1985.

[7] Martin Buchheit, Francesco M. Donini, and Andrea Schaerf. Decidable reasoning in terminological knowledge representation systems. In *Proc. of the 13th Int. Joint Conf. on Artificial Intelligence IJCAI-93*, pages 704–709, 1993.

[8] Giuseppe De Giacomo and Maurizio Lenzerini. Concept language with number restrictions and fixpoints, and its relationship with mu-calculus. To appear in ECAI-94.

[9] Giuseppe De Giacomo and Maurizio Lenzerini. Boosting the correspondence between description logics and propositional dynamic logics (extended abstract). To appear in AAAI-94.

[10] Giuseppe De Giacomo and Maurizio Lenzerini. Description logics with inverse roles, functional restrictions, and n-ary relations. To appear in JELIA-94.

[11] Francesco M. Donini, Bernhard Hollunder, Maurizio Lenzerini, Alberto Marchetti Spaccamela, Daniele Nardi, and Werner Nutt. The complexity of existential quantification in concept languages. *Artificial Intelligence*, 2–3:309–327, 1992.

[12] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Werner Nutt. The complexity of concept languages. In *Proc. of the 2nd Int. Conf. on Principles of Knowledge Representation and Reasoning KR-91*, pages 151–162, 1991.

[13] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Werner Nutt. Tractable concept languages. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence IJCAI-91*, pages 458–463, 1991.

[14] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, Werner Nutt, and Andrea Schaerf. Adding epistemic operators to concept languages. In *Proc. of the 3rd Int. Conf. on Principles of Knowledge Representation and Reasoning KR-92*, pages 342–353, 1992.

[15] Dexter Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science* 27, pages 333–354, 1983.

[16] Dexter Kozen and Jerzy Tiuryn. Logics of programs. In *Handbook of Theoretical Computer Science – Formal Models and Semantics*, pages 789–840, Elsevier, 1990.

[17] Hector J. Levesque and Ron J. Brachman. Expressiveness and tractability in knowledge representation and reasoning. *Computational Intelligence*, 3:78–93, 1987.

[18] Robert MacGregor. Inside the LOOM description classifier. *SIGART Bulletin*, 2(3):88–92, 1991.

[19] Bernhard Nebel. Computational complexity of terminological reasoning in BACK. *Artificial Intelligence*, 34(3):371–383, 1988.

[20] Bernhard Nebel. Terminological reasoning is inherently intractable. *Artificial Intelligence*, 43:235–249, 1990.

[21] Bernhard Nebel. Terminological cycles: Semantics and computational properties. In John F. Sowa, editor, *Principles of Semantic Networks*, pages 331–361. Morgan Kaufmann, 1991.

[22] Peter F. Patel-Schneider. A hybrid, decidable, logic-based knowledge representation system. *Computational Intelligence*, 3(2):64–77, 1987.

[23] Klaus Schild. A correspondence theory for terminological logics: Preliminary report. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence IJCAI-91*, pages 466–471, 1991.

[24] Klaus Schild. Terminological cycles and the propositional μ-calculus. In *Proc. of the 4th Int. Conf. on Principles of Knowledge Representation and Reasoning KR-94*. To appear, 1994.

[25] Manfred Schmidt-Schauß and Gert Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26, 1991.

[26] Colin Stirling. Modal and temporal logics. In *Handbook of Logic in Computer Science*, Vol. 2, pages 477–563, Clarendon Press, Oxford, 1992.

Which Semantics for Individuals in the TBox?

Andrea Schaerf
Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, 00198 Roma, Italia
email: aschaerf@assi.dis.uniroma1.it

1 Introduction

In this paper we investigate terminological systems that allow for the presence of the individuals in the TBox. Such presence is obtained by introducing ad hoc constructors in the concept-defining language. In particular, one of these constructors is obtained by building a concept from a set of enumerated individuals. This constructor, called ONE-OF and written $\{a_1, \ldots, a_n\}$ allows one to express many natural concepts. For example, the concept `Permanent_onu_member` can be defined as {`china, france, russia, uk, usa`}, where `china`, ..., `usa` are individuals.

The demand for the constructor ONE-OF in terminological systems is due to the significant increase in the expressiveness it provides. It is also confirmed by the fact that it is included in both the proposals for a standard system in [Patel-Schneider and Swartout,1993] and [Baader *et al.*,1991], and in various actual systems (e.g. CLASSIC).

The semantics employed for ONE-OF in actual systems is generally a variant of the first-order one (e.g. [Borgida and Patel-Schneider,1993]). In this paper instead, we explore on the consequences of using the pure first-order semantics for ONE-OF, and we perform an analysis of the various issues related to ONE-OF in this case. This analysis gives an insight of the problem of reasoning with individuals and allows us to understand the intuitive aspects which makes reasoning difficult, and why developers of actual system left such semantics.

In [Schaerf,1993b] we also consider another concept constructor of the above kind, called FILLS, that denotes the class of objects that have a specific individual as a filler for a given role.

For our analysis we make use of the language $\mathcal{ALE}$ [Donini *et al.*,1992; Schmidt-Schauß and Smolka,1991] augmented with ONE-OF. In such a language, concepts (denoted by the letters C and D) are built up by means of the following syntax rule (where A denotes a concept name and R denotes a role name):

$$
\begin{array}{lll}
C, D \longrightarrow & A \mid & \text{(concept name)} \\
& \top \mid \bot \mid & \text{(top, bottom)} \\
& \neg A \mid & \text{(name complement)} \\
& C \sqcap D \mid & \text{(intersection)} \\
& \forall R.C \mid \exists R.C \mid & \text{(quantifications)} \\
& \{a_1, \ldots, a_n\} & \text{(one-of)}
\end{array}
$$

Very briefly, we call ABox a set of assertions of one of the forms

$$C(a) \text{ or } R(a, b)$$

and TBox is a set of axioms of the forms

$$A \doteq C \text{ or } A \dot{\leq} C,$$

with the usual acyclicity restrictions and the condition that the negation appears only in front of atomic concepts (which are the concepts that do not appear in the left-hand side of an axiom).

A knowledge base Σ is a pair $\Sigma = \langle \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{T}$ is a TBox and $\mathcal{A}$ is an ABox. In this paper we do not consider the TBox and we focus only on the ABox. Consequently, we will consider only the following reasoning services:

- Concept Satisfiability: $C \not\equiv \bot$
- Subsumption: $C \sqsubseteq D$
- ABox-Consistency: $\mathcal{A} \not\models \bot(b)$
- Instance Checking: $\mathcal{A} \models C(b)$
- Hybrid Subsumption: $\mathcal{A} \models C \sqsubseteq D$

For the sake of brevity, we do not supply the semantics of concepts, knowledge bases and the above reasoning services. They can be found, for example, in [Schaerf,1993b].

Example 1.1 Let $\mathcal{A}$ be the following ABox:

$\mathcal{A}$ = { ∃`FRIEND.{susan,peter}(john)`,
∀`FRIEND.Married(john)`
¬`Married(peter)`}

It is easy to see that $\mathcal{A}$ is consistent. Moreover, some non-trivial conclusions can be drawn from $\mathcal{A}$. For example, we can prove that $\mathcal{A}$ ⊨`Married(susan)`. Intuitively, they can be explained by the following reasoning: Due to the last two assertions, Peter cannot be a friend of John; therefore, according to the first assertion, the friend of John must be Susan and, consequently, she must be married, too. □

2 On the Role of ONE-OF in Reasoning

In this section we give an overview of the most relevant issues related to **ONE-OF** and to the individuals in the TBox in general. We do not claim that the list below is exhaustive. Some other issues can be found in [Schaerf,1993b].

2.1 Implicit Assertions

One of the characteristics of concept languages is the ability of describing incomplete knowledge. In particular, by means of existential quantification, it is possible to express information about objects that exist but whose identity is not known to the knowledge base. With regards to these unknown objects, it is also possible to state their membership to some concept. In particular, when **ONE-OF** is used, it is possible to state the membership of an unknown object to a set of individuals. A consequence of this is that the unknown object is forced to be one of the individuals of the set.

For example, consider the following assertion:

$$\exists R.(A \sqcap \{b\})(d).$$

It explicitly states the membership of d in $\exists R.(A \sqcap \{b\})$, but it also implicitly states that b must be in the extension of A. In fact, it says that there exists an object in $A \sqcap \{b\}$, therefore this object must be b and it must be in the extension of A, that is equivalent to stating the assertion $A(b)$.

If we consider collection formed by more than one element then the resulting implicit assertion can be disjunctive. For example, if we state the existence of an object in the concept $A \sqcap \{b, c\}$, then the resulting implicit assertion is $A(b) \vee A(c)$.

Consider the following concept formed by a conjunction of three existential quantifications

$$\exists R.(A \sqcap \{b, c\}) \sqcap \exists R.(\neg A \sqcap \{b\}) \sqcap \exists R.(\neg A \sqcap \{c\}).$$

Suppose that we want to check its satisfiability. The standard approach (e.g. [Schmidt-Schauß and Smolka,1991; Baader and Hollunder,1991]) to this problem is to separately check for the satisfiability of the three concepts involved in the existential quantifications, namely $A \sqcap \{b, c\}$, $\neg A \sqcap \{b\}$, and $\neg A \sqcap \{c\}$. It is easy to see that this technique fails to recognize that the whole concept is unsatisfiable. In fact, although each of the conjuncts is separately satisfiable, the conjunction of their implicit assertion (i.e. $A(b) \vee A(c)$, $\neg A(b)$, and $\neg A(c)$) is unsatisfiable.

2.2 Mixing Terminological and Assertional Knowledge

Another important characteristic of concept languages is that the reasoning process in the terminological component in general is not influenced by the assertional knowledge.

More precisely, the following theorem holds for a large class of languages (in [Nebel,1990], here simplified slightly from the original version):

Theorem 2.1 *Given a consistent ABox $\mathcal{A}$, for every pair of concepts C, D:*

$$\mathcal{A} \models C \sqsubseteq D \quad \textit{iff} \quad C \sqsubseteq D.$$

The above property is crucial for the efficiency of reasoning in concept-based knowledge representation systems. In fact, it allows for the maintenance of a static hierarchy of concepts; which is not influenced by the evolution of the assertions. Unfortunately, such nice property does not hold when the language includes **ONE-OF** as shown in the following example.

Example 2.2 Let $\mathcal{A} = \{E(a), E(b)\}$. It is easy to see that

$$\forall R.\{a, b\} \not\sqsubseteq \forall R.E.$$

In fact, given an interpretation $\mathcal{I}$ such that $R^{\mathcal{I}} = \{(d_1, d_2)\}$, $E^{\mathcal{I}} = \emptyset$, and $a^{\mathcal{I}} = d_2$ we have that $d_1 \in (\forall R.\{a, b\})^{\mathcal{I}}$ and $d_1 \notin (\forall R.E)^{\mathcal{I}}$. On the other hand

$$\mathcal{A} \models (\forall R.\{a, b\} \sqsubseteq \forall R.E).$$

That is because, in every model of $\mathcal{A}$ all the objects related only with a and b by means of R are obviously related only to object in E

2.3 Abstraction

Abstraction is a well known mechanism in reasoning about individuals in concept-based systems. It consists in retrieving all the assertions relevant to a given individual a and collecting them into a single concept. Such concept has the property of being the most specific concept (expressible in the language) such that the individual a is an instance of. For this reason it is generally indicated by $MSC(a)$ (Most Specific Concept).

Abstraction, together with subsumption, allows one to perform instance checking. In fact, an algorithm for checking whether $\mathcal{A} \models C(a)$ can work as follows: Step 1. compute $MSC(a)$. Step 2. check whether C subsumes $MSC(a)$. This algorithm, called Abstraction/Subsumption, has been broadly exploited in actual systems (see [Quantz and Kindermann,1990; Nebel,1990]).

However, the problem of exploiting this algorithm is that, in general, it is not possible to completely fit in the information relevant to an individual into a single concept of the language. Let us clarify this point by means of an example: Let $\mathcal{A}$ be the following ABox $\mathcal{A} = \{R(a, a), B(a)\}$. Now consider the individual a; the abstraction for a in $\mathcal{ALE}$ returns $MSC(a) = B \sqcap \exists R.B$. In $MSC(a)$, the information that the individual related to a is exactly a itself is lost. This fact has an impact on the completeness of the algorithm; for instance the algorithm fails to draw the conclusion that $\mathcal{A} \models \exists R.\exists R.B(a)$.

In general, any time an individual is referred more than once in the knowledge base, the connection between the different occurrences may be lost. For this reason, the algorithms for instance checking based on abstraction, for expressive languages, are in general incomplete.

Nevertheless, if the language includes ONE-OF it is possible to make a *lostless* abstraction. In the previous example, if the language is $\mathcal{ALE}$ plus ONE-OF instead of $\mathcal{ALE}$, the abstraction for a gives $MSC(a) = \{a\} \sqcap B \sqcap \exists R.\{a\}$, and it is easy to see that the inference $\mathcal{A} \models \exists R.\exists R.B(a)$ is captured because $\{a\} \sqcap B \sqcap \exists R.\{a\} \sqsubseteq \exists R.\exists R.B$ holds. In conclusion the use of ONE-OF gives the possibility of doing complete reasoning using the Abstraction/Subsumption algorithm.

3 Complexity of Reasoning with ONE-OF

We start our complexity analisys by showing that when ONE-OF is in use, the reasoning problems involving the ABox have always the same complexity than the ones involving only concepts (which is not the case for $\mathcal{ALE}$, as shown in [Schaerf,1993a]). In order to achieve this result, we present the transformation Φ, from an ABox to a concept,

$$\Phi : ABox \longrightarrow Concept$$

defined as follows. Let $\mathcal{A}$ be an ABox, C a concept, and a, b two individuals, then:

$$\begin{aligned} \Phi(\mathcal{A}) &= \sqcap_{(\alpha \in \mathcal{A})} \Phi(\alpha) \\ \Phi(C(a)) &= \exists Q.(\{a\} \sqcap C) \\ \Phi(R(a,b)) &= \exists Q.(\{a\} \sqcap \exists R.\{b\}) \end{aligned}$$

where Q does not appear in $\mathcal{A}$. Intuitively, Φ "encodes" the ABox $\mathcal{A}$ in the implicit assertions of the concept $\Phi(\mathcal{A})$. The following proposition states the relation between the $\mathcal{A}$ and $\Phi(\mathcal{A})$.

Proposition 3.1 *Given an ABox $\mathcal{A}$, an individual a, and a concept C then:*

(i) $\mathcal{A}$ is consistent iff $\Phi(\mathcal{A})$ is satisfiable,

(ii) $\mathcal{A} \models C(a)$ iff $\Phi(\mathcal{A}) \sqcap \{a\} \sqsubseteq C$.

In [Schaerf,1993b] we also investigate on the complexity of the reasoning services in the specific languages. In that paper, we consider various languages that do not use ONE-OF and the corresponding languages obtained by adding it. In particular, we focus on the languages $\mathcal{ALC}$ (i.e. $\mathcal{ALE}$+ general negation), $\mathcal{ALE}$, and $\mathcal{AL}$ (i.e. $\mathcal{ALE}$ with existential quantifications only of the form $\exists R.\top$), which are a good representative of the various degrees of expressiveness (and complexity), and we achieve some complexity results on the corresponding languages with ONE-OF. We summarize the results we have obtained in that paper in Table 1, which also contains the previous known results for the corresponding languages without ONE-OF.

4 ONE-OF in the Query Language

In the previous section we have shown that the use of ONE-OF generally increases the complexity of reasoning. Despite this negative result, there exists one possibility of exploiting ONE-OF in a useful way without increasing the complexity of reasoning: Given the instance checking problem $\mathcal{A} \models C(a)$, we admit ONE-OF only in the query language, i.e. we allow ONE-OF in the expression of the query concept C but not in the assertions of $\mathcal{A}$.

Using ONE-OF it is possible to express various forms of selection that are usually admitted in database query languages, but are usually missing in standard concept languages. For example, it is possible to ask for the books whose author is Newton and whose subject is mathematics:

$$\texttt{Book} \sqcap \exists \texttt{AUTHOR}.\{\texttt{newton}\} \sqcap \exists \texttt{SUBJECT}.\{\texttt{math}\}$$

In [Lenzerini and Schaerf,1991b], it is shown that it is possible to query an ABox in $\mathcal{AL}$ using $\mathcal{AL}$+ONE-OF concepts in polynomial time, in spite of the fact that reasoning in $\mathcal{AL}$+ONE-OF is in general coNP-complete.

5 Discussion and Conclusions

We have shown an analysis of the various issues related to the use of a concept constructor involving individuals. Moreover, we have presented a set of complexity results which formally confirms that reasoning with individuals is generally hard. In fact, in some languages, they increases the complexity of reasoning ($\mathcal{AL}$,$\mathcal{ALE}$). Whereas, in those cases in which reasoning is in the same complexity class as the underlying language ($\mathcal{ALC}$), as shown in [Schaerf,1993b] the algorithms are generally more complex and less efficient (in term of both time and space) than in the corresponding language without individuals.

We have also identified an intuitive explanation of this intractability: On one side, it is related to the implicit disjunction carried by the use of sets with more then one object. On the other side, it is due to the implicit equality associated with individuals in concept expressions.

In our opinion, the solutions proposed in actual systems to overcome the computational intractability are not completely satisfying. Therefore a deeper insight of the problem can also be useful for the development of better incomplete reasoners and/or variant semantics.

References

[Baader and Hollunder, 1991] Franz Baader and Bernhard Hollunder. A terminological knowledge representation system with complete inference algorithm. In *Proc. of the Workshop on Processing Declarative Knowledge, PDK-91*, Lecture Notes in Artificial Intelligence, pages 67–86. Springer-Verlag, 1991.

[Baader *et al.*, 1991] Franz Baader, Hans-Jürgen Bürkert, Jochen Heinsohn, Bernhard Hollunder, Jürgen Müller, Bernard Nebel, Werner Nutt, and Hans-Jürge Profitlich. Terminological knowledge representation: A proposal for a terminological logic. Technical Report TM-90-04, Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI), Kaiserslautern, Germany, 1991.

[Borgida and Patel-Schneider, 1993] Alexander Borgida and Peter F. Patel-Schneider. A semantics and complete algorithm for subsumption in the CLASSIC description logic. Submitted for publication, 1993.

	without ONE-OF			with ONE-OF
	subsumption	instance checking		subsumption & inst. ch.
$\mathcal{AL}$	P [Schmidt-Schauß and Smolka,1991]	P [Lenzerini and Schaerf,1991a]	$\mathcal{AL}$+ ONE-OF	coNP
$\mathcal{ALE}$	NP [Donini *et al.*,1992]	PSPACE [Donini *et al.*,1994]	$\mathcal{ALE}$+ ONE-OF	PSPACE
$\mathcal{ALC}$	PSPACE [Schmidt-Schauß and Smolka,1991]	PSPACE [Baader and Hollunder,1991]	$\mathcal{ALC}$+ ONE-OF	PSPACE

Table 1: Complexity of reasoning

[Donini *et al.*, 1992] Francesco M. Donini, Bernhard Hollunder, Maurizio Lenzerini, Alberto Marchetti Spaccamela, Daniele Nardi, and Werner Nutt. The complexity of existential quantification in concept languages. *Artificial Intelligence*, 2–3:309–327, 1992.

[Donini *et al.*, 1994] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Andrea Schaerf. Deduction in concept languages: From subsumption to instance checking. *Journal of Logic and Computation*, 4(92–93):1–30, 1994. To appear.

[Lenzerini and Schaerf, 1991a] Maurizio Lenzerini and Andrea Schaerf. Concept languages as query languages. In *Proc. of the 9th Nat. Conf. on Artificial Intelligence AAAI-91*, pages 471–476, 1991.

[Lenzerini and Schaerf, 1991b] Maurizio Lenzerini and Andrea Schaerf. Querying concept-based knowledge bases. In *Proc. of the Workshop on Processing Declarative Knowledge, PDK-91*, Lecture Notes in Artificial Intelligence, pages 107–123. Springer-Verlag, 1991.

[Nebel, 1990] Bernhard Nebel. *Reasoning and Revision in Hybrid Representation Systems.* Lecture Notes in Artificial Intelligence. Springer-Verlag, 1990.

[Patel-Schneider and Swartout, 1993] P. F. Patel-Schneider and Bill Swartout. Working version (draft): Description logic specification from the KRSS effort, June 1993. Unpublished Manuscript.

[Quantz and Kindermann, 1990] Joachim Quantz and Carsten Kindermann. Implementation of the BACK system version 4. Technical Report KIT-Report 78, FB Informatik, Technische Universität Berlin, Berlin, Germany, 1990.

[Schaerf, 1993a] Andrea Schaerf. On the complexity of the instance checking problem in concept languages with existential quantification. *Journal of Intelligent Information Systems*, 2:265–278, 1993. An abridged version appeared in *Proc. of the 7th Int. Symp. on Methodologies for Intelligent Systems ISMIS-93.*

[Schaerf, 1993b] Andrea Schaerf. Reasoning with individuals in concept languages. Technical Report 07.93, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", 1993. An abridged version appeared in *Proc. of the 3rd Conf. of the Italian Association for Artificial Intelligence AI*IA-93.*

[Schmidt-Schauß and Smolka, 1991] Manfred Schmidt-Schauß and Gert Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26, 1991.

Tractable Reasoning in a Universal Description Logic: Extended Abstract*

Klaus Schild
German Research Center for Artificial Intelligence
Stuhlsatzenhausweg 3, D-66123 Saarbrücken, FRG
e-mail: schild@dfki.uni-sb.de

1 Introduction

Description logics (also called *terminological logics* or *concept languages*) have been designed for the logical reconstruction and specification of knowledge representation systems descending from KL-ONE such as BACK, CLASSIC, $\mathcal{KRIS}$, and LOOM.[1] These systems are used to make the terminology of an application domain explicit and then to classify these definitions automatically into a taxonomy according to semantic relations like subsumption and equivalence. More precisely, automatic classification refers to the ability to insert a new concept into the taxonomy in such a way that it is directly linked to the most specific concept it is subsumed by and to the most general concept it in turn subsumes. Terminological knowledge representation systems thereby support the task to formalize an application in at least two respects. On the one hand, they urge the user to isolate the intrinsic concepts of the application; on the other hand they may detect hidden subsumption and equivalence relations between definitions or may even detect that a definition is incoherent.

A model of the application is then given by associating special objects of the domain with the concepts of the terminology. The systems mentioned above in turn automatically classify these objects with respect to the given terminology and to those membership relations which have been asserted explicitly. In this case, however, automatic classification refers to the ability to find the most specific concept the object is a member of.

Terminologies comprise two different kinds of terms, viz. so-called *concepts* and *roles*. The former are intended to represent classes of objects of a given domain, while the latter represent binary relations over this domain. Concepts can either be simple *concept names*, representing not further specified classes of objects, or structured by means of a fixed set of *concept structuring primitives*. Common concept structuring primitives are *concept conjunction* $\sqcap$ and universal quantification $\forall R{:}C$ over a role R. Concept conjunction is to be interpreted as set intersection, while the concept $\forall R{:}C$ denotes all those objects d of the domain for which each object related to d by the role R is a member of the concept C. Although there exist many other concept structuring primitives, it is commonly accepted that these two should be part of each concept language. In contrast to concepts, roles are often taken to be atomic, i.e., there are no roles other than *role names*. The standard concept language $\mathcal{ALC}$, for instance, does not comprise any role structuring primitives. However, in addition to those mentioned above, this language comprises *concept disjunction* $\sqcup$, *concept negation* $\neg$ as well as existential quantification $\exists R{:}C$ over a role R as concept structuring primitives. For details the reader is referred to [Schmidt-Schauß and Smolka,1991].

Definitions are given by associating a concept or role T with a concept name (resp., role name) TN. Such a definition is represented by the expression $TN \doteq T$ and is called *concept* and *role introduction* respectively. *Terminologies* are just finite sets of concept and role introductions such that each concept and role name is defined at most once, i.e., for every concept and role name TN there exists at most one concept or role introduction the left-hand side of which is TN.

As already mentioned, a model of application domain is described in terms of the given terminology. More precisely, specific objects of the domain and pairs of objects can be associated with concepts and roles of the terminology, where these objects are syntactically represented by so-called *individual names*. It can either be asserted that an individual name a is an instance of a concept C or that it is related to another individual name, say, b, by a role R. Such assertions are called *assertional axioms* and are represented by the expressions $a{:}C$ and $(a,b){:}R$ respectively. A finite set of assertional axioms forms a *knowledge base*.

From a theoretical point of view, the computational service provided by terminological knowledge representation systems can be reduced to answer queries of the following form with respect to a knowledge base $\mathcal{KB}$ and to a terminology $\mathcal{T}$: a *query* can be an assertional axiom or an *inclusion axiom* of the form $T_1 \sqsubseteq T_2$, where T_1 and T_2 are either two concepts or two roles. The meaning of such a query Q posed with respect to $\mathcal{KB}$ and $\mathcal{T}$ is usual-

*This work was supported by a grant from the *Deutsche Forschungsgemeinschaft* (DFG).

[1]For a good overview of the so-called KL-ONE family the reader is referred to [Woods and Schmolze,1992]; for KL-ONE itself cf. [Brachman and Schmolze,1985].

ly given in terms of so-called interpretations and models. An *interpretation* $\mathcal{I}$ consists of a *domain* $\Delta^{\mathcal{I}}$ and a *valuation* $\mathcal{V}$ over $\Delta^{\mathcal{I}}$ along with an *interpretation function* $.^{\mathcal{I}}$. The valuation $\mathcal{V}$ over $\Delta^{\mathcal{I}}$ maps each concept name to a subset of $\Delta^{\mathcal{I}}$ and each role name to a binary relation over $\Delta^{\mathcal{I}}$. Individual names, however, are mapped to singleton sets containing exactly one element of $\Delta^{\mathcal{I}}$. The interpretation function $.^{\mathcal{I}}$, on the other hand, just extends $\mathcal{V}$ to deal with arbitrary concepts and roles in such a way that all concept and role structuring primitives are interpreted properly. The concept structuring primitives $\sqcap$, $\sqcup$, $\neg$, for instance, are to be interpreted as the corresponding set operations on $\Delta^{\mathcal{I}}$, while the interpretation of the concept $\forall R{:}C$ is defined inductively as follows: if $C^{\mathcal{I}}$ and $R^{\mathcal{I}}$ have already been defined, then $(\forall R{:}C)^{\mathcal{I}}$ is $\{d \in \Delta^{\mathcal{I}} : \forall e(\langle d, e\rangle \in R^{\mathcal{I}}), e \in C^{\mathcal{I}}\}$.

An interpretation $\mathcal{I}$ is then said to be a *model* of the inclusion axiom $T_1 \sqsubseteq T_2$ just in case that $T_1^{\mathcal{I}} \subseteq T_2^{\mathcal{I}}$ and, if a and b are individual names such that $a^{\mathcal{I}}$ is $\{\underline{a}\}$ and $b^{\mathcal{I}}$ is $\{\underline{b}\}$, then $\mathcal{I}$ is a model of the assertional axiom $a{:}C$ (resp., of $(a,b){:}R$) just in case that $\underline{a} \in C^{\mathcal{I}}$ (resp., $\langle \underline{a}, \underline{b}\rangle \in R^{\mathcal{I}}$). Not very surprising, an interpretation is a *model* of $\mathcal{KB}$ and $\mathcal{T}$ if it is a model of each of the elements of $\mathcal{KB}$ and $\mathcal{T}$. Now, Q is said to be *entailed* by $\mathcal{KB}$ and $\mathcal{T}$, written $\mathcal{KB} \models_{\mathcal{T}} Q$, if and only if every interpretation which is a model of $\mathcal{KB}$ and $\mathcal{T}$ is a model of Q as well. Moreover, we say that T_2 *subsumes* T_1 with respect to $\mathcal{T}$ if and only if it holds that $\emptyset \models_{\mathcal{T}} T_1 \sqsubseteq T_2$.

2 Terminological Reasoning is Inherently Intractable

Unfortunately, answering such queries is in most cases provably intractable, at least in terms of computational worst case complexity. This applies, for instance, to the basic inference of KL-ONE, although originally claimed to be computationally tractable. In fact, Schmidt-Schauß [1989] proved that there exists no algorithm at all which decides whether one concept of KL-ONE subsumes another one or not, even with respect to empty terminologies.

Moreover, in [Schild,1993, 94a], it is proved that in case of the standard concept language $\mathcal{ALC}$, every algorithm capable of deciding whether one concept subsumes another one or not uses more than polynomial time in the worst case if at least one (possibly recursive) concept introduction is taken into account. Notably, this result holds no matter which of the usual kinds of semantics for recursive concept introductions is presupposed, viz. either *descriptive semantics* or *least* or *greatest fixed point semantics*, as Nebel [1991] called them.

It is also known that even in case of the minimal concept language (comprising no concept and role structuring primitives other than concept negation and universal quantification over role names), there exists no polynomial time algorithm which decides with respect to acyclic terminologies whether one concepts subsumes another one or not, unless P = NP [Nebel,1990].

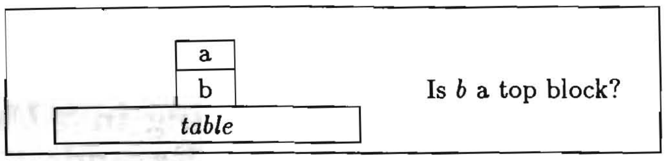

Figure 1: A sample blocks world.

$$\left\{\begin{array}{lll} \forall x.block(x) \Leftrightarrow x = a \vee x = b, & & \\ a \neq b, a \neq table, b \neq table, & & \\ \forall x \forall y.on(x,y) & \Leftrightarrow & (x = a \wedge y = b) \\ & \vee & (x = b \wedge y = table) \end{array}\right\}$$
$$\overset{?}{\models} \; block(b) \wedge \neg\exists x.block(x) \wedge on(x,b)$$

Figure 2: Representing the sample blocks world by first-order formulae.

3 Model Checking Versus Theorem Proving

In the previous section, we have seen that, as Woods and Schmolze [1992] put it, "the surfeit of intractability results seems to have reached its logical end with the conclusion that practically everything of any use is intractable (in the worst case)." Recently, Halpern and Vardi [1991] proposed a possible solution to this very problem of knowledge representation. As a starting point, they re-examined the traditional approach to knowledge representation, going back to McCarthy [1968]. According to this approach the world to be modeled should be represented by a finite set of formulae of some given logic, preferably first-order logic. If a question to be answered is then formulated within the same logic, the answer depends on whether this formula is a *logical consequence* of the collection of formulae representing the world or not. In other words, it is checked whether *every* semantic structure which is a model of each of the formulae representing the world is also a model of formula corresponding to the question.

We shall illustrate this traditional approach to knowledge representation by means of an example, drawn from the famous blocks world. Suppose, for instance, we would like to represent a blocks world involving two blocks, say, a and b, where a lies on b and the latter in turn lies on a table. Suppose, furthermore, we would like to know whether b is a top block or not. Figure 1 depicts exactly this situation, while Figure 2 gives its representation in terms of first-order logic in the traditional way just described.

McCarthy's approach, however, gives rise to the problem that the need to represent all facts about the world in terms of some logic necessitates the use of very expressive logics such as full first-order logic. This, in fact, gives rise to difficulties because it is known that there exists no algorithm at all which generally decides logical consequence in full first-order logic [Church,1936], and this

$$\begin{array}{lcl} \mathcal{D}om & = & \{a, b, table\} \\ [\![block]\!] & = & \{a, b\} \\ [\![on]\!] & = & \{\langle a, b\rangle, \langle b, table\rangle\} \end{array}$$
$$\overset{?}{\models} block(b) \wedge \neg\exists x.block(x) \wedge on(x, b)$$

Figure 3: Representing the sample blocks world by a semantic structure.

remains true even when only finite interpretation domains are taken into consideration [Trahtenbrot,1963].

At this very point Halpern and Vardi stressed that in many cases the natural representation of a world to be modeled is a *semantic structure* rather than a collection of formulae. If, as in the traditional approach, queries are represented by formulae of a given logic, a query can be answered in this case depending on whether the formula representing the query is true in the *given* semantic structure or not. That is to say, it is checked whether the semantic structure is a model of the formula corresponding to the query. The fact that a (closed) formula α is true in a semantic structure $\mathcal{M}$ is usually indicated by $\mathcal{M} \models \alpha$. Resorting to this convention, Figure 3 gives such an alternative representation of the blocks world considered above.

In many cases this model checking approach has tremendous benefits, at least in terms of computational complexity. For instance, checking the truth of an arbitrary closed first-order formula[2] α in a finite semantic structure fixing the interpretation of all predicates and constants occurring in α is known to be *decidable* using at most polynomial space [Chandra and Merlin,1977]. Recall that in contrast to this, there exists no algorithm at all which is able to decide whether an arbitrary formula of this kind is a logical consequence of a finite set of first-order formulae, even with only finite interpretation domains taken into account. However, it is also known that first-order model checking is still at least as hard as *any other* problem solvable using at most polynomial space, hence this problem is still very hard [Chandra and Merlin,1977]. Anyway, Halpern and Vardi's intention was to forge a new approach to knowledge representation rather than to give concrete instances which allow for tractable inferences.

4 The Model Checking Approach to Terminological Reasoning

It should be clear that terminological knowledge representation, as described in the introduction, is committed to the traditional approach to knowledge representation rather than to the model checking approach. In [Schild,1994b] we investigated the consequences of adapting Halpern and Vardi's model checking approach to terminological reasoning. It turned out that even in case

[2]This formula should involve no function symbols other than constants.

$$\left\{ \begin{array}{l} a{:}Block, b{:}Block, table{:}\neg Block, \\ (a, b){:}on, (b, table){:}on, \\ a{:}(\neg\exists on^{-1}{:}Block), table{:}(\neg\exists on{:}Block) \end{array} \right\}$$
$$\mathcal{T} = \{TopBlock \doteq Block \sqcap \neg\exists on^{-1}{:}Block\}$$
$$\overset{?}{\models}_{\mathcal{T}} b{:}TopBlock$$

Figure 4: Representing the sample blocks world by an $\mathcal{ALC}^{-1}$-KB.

$$\begin{array}{lcl} \mathcal{D}om & = & \{a, b, table\} \\ [\![Block]\!] & = & \{a, b\} \\ [\![on]\!] & = & \{\langle a, b\rangle, \langle b, table\rangle\} \end{array}$$
$$\mathcal{T} = \{TopBlock \doteq Block \sqcap \neg\exists on^{-1}{:}Block\}$$
$$\overset{?}{\models}_{\mathcal{T}} b{:}TopBlock$$

Figure 5: Representing the sample blocks world by a physical $\mathcal{ALC}^{-1}$-KB.

of the most powerful description logic considered in the literature, answering queries become tractable just by replacing the usual kind of knowledge bases with single finite semantic structures fixing the interpretation of all *primitive* concepts and roles (i.e., those concept and role names which are mentioned somewhere in the terminology or in the query, but which are not defined).

But before engaging into details, have a look at Figure 4, which shows how to represent the already familiar blocks world in terms of $\mathcal{ALC}$ together with the inverse of roles $^{-1}$, as it would be done traditionally. Observe, however, that this representation is *incomplete* in that it solely states that block a lies on block b, while the latter in turn lies on the table, but it is left open whether there is any other block lying on b or on the table. As a matter of fact, there is no way at all to give an *accurate* representation of our blocks world in terms of $\mathcal{ALC}$, even when augmented by the inverse of roles. This means, in this case the so-called *open world assumption*,[3] traditionally made for terminological reasoning, is a nuisance rather than an advantage.

Figure 5 modifies the just considered representation in the spirit of the model checking approach. A finite semantic structure is shown there which fixes the interpretation of each primitive concept and role of $\mathcal{T}$, that is, it fixes the interpretation of *Block* and *on*. Such a semantic structure is obviously nothing but a valuation along with a domain. When taken together with a domain, the syntactic representation of such a valuation is called *physical knowledge base*, emphasizing the fact that they are intended to replace customary knowledge bases.

[3]In contrast to the *closed world assumption*, usually made for databases, the open world assumption does *not* assume that all those facts that are not explicitly mentioned (or that cannot be inferred) are taken to be false.

Now, suppose $\mathcal{V}$ is such a physical knowledge base with domain $\mathcal{D}om$, $\mathcal{T}$ is an arbitrary terminology, and Q is a query. Then $\mathcal{V} \models_{\mathcal{T}} Q$ is intended to mean that every interpretation extending $\mathcal{V}$ which is a model of $\mathcal{T}$ is a model of Q as well, where an interpretation $\mathcal{I}$ is said to *extend* a physical knowledge base $\mathcal{V}$ with domain $\mathcal{D}om$ just in case that $\Delta^{\mathcal{I}} = \mathcal{D}om$ and, moreover, $\cdot^{\mathcal{I}}$ interprets all those concept and role names handled by $\mathcal{V}$ in exactly the same way as $\mathcal{V}$ does.

In [Schild,1994b] we investigated the computational complexity of answering such queries with respect to physical knowledge bases in the description logic $\mathcal{U}$, introduced by Patel-Schneider [1987] as a universal description logic. This concept language is *universal* in the sense that it encompasses all others considered in the literature, except for those which comprise nonstandard facilities like defaults, for instance. In addition to those of $\mathcal{ALC}$, this language comprises *number restrictions* of the form $\exists^{\geq n} R{:}C$ and $\exists^{\leq m} R{:}C$ as well as *role value maps* of the form $R \leq S$ as concept structuring primitives. Number restrictions restrict the number of role fillers (i.e., those objects which are related to an object by a role), while role value maps impose restrictions on the fillers of two roles. The concept $R \leq S$ states that all fillers of the role R are also fillers of the role S. In addition, $\mathcal{U}$ admits of individual names to occurring in concepts. The role structuring primitives of $\mathcal{U}$ are the *identity role* ϵ, Boolean operations $\sqcap$, $\sqcup$, $\neg$ on roles, the inverse R^{-1} of a role, the composition $R \circ S$ of two roles, as well as the *transitive closure* R^{+} and the *reflexive-transitive closure* R^{*} of a role. For details cf. [Schild,1994b] or [Patel-Schneider,1987]. Notably, it is known that there cannot exist any algorithm which is capable of deciding subsumption between two concepts (or two roles) of $\mathcal{U}$, even with respect to empty terminologies [Schild,1988].

The main result of [Schild,1994b] is that even in this language $\mathcal{V} \models_{\mathcal{T}} Q$ can be decided in polynomial time provided that each of the following conditions is satisfied:

(a) $\mathcal{V}$ has a finite domain and specifies all concept and role names occurring in $\mathcal{T}$ and Q except for those which are defined in $\mathcal{T}$;

(b) Roles are not defined recursively;

(c) Concepts can be defined recursively, but then they must occur in their definition[4] *positively*, i.e., they must occur in the scope of an even number of negations, where $\exists^{\leq m} R{:}$ counts also as a negation. Moreover, each recursive definition must be given either least or greatest fixed point semantics, not necessarily in a uniform way.

Of course, each of these conditions calls for some comment. Condition (b) is commonly presupposed for terminological reasoning, while condition (c) constitutes the most liberal restriction on recursive concept definitions considered in the literature. The most important condition, however, is the first one in that it ensures all primitive concepts and roles to be specified extensionally. This restriction does make sense as these concepts and roles are exactly those which are not further specified according to the semantics. It can easily be verified that the sample query of Figure 5 obeys each of the three conditions above.

The employed algorithm capable of deciding $\mathcal{V} \models_{\mathcal{T}} Q$ in polynomial time just mimics the semantics of the concept and role structuring primitives of $\mathcal{U}$, storing already evaluated ones. To deal with recursive concept definitions, however, we exploited a technique for computing least and greatest fixed points due to Emerson and Lei [1986].

It turned out that even when relaxing condition (a) in such a way that $\mathcal{V}$ is solely required to have a finite domain, $\mathcal{V} \models_{\mathcal{T}} Q$ is still decidable in the universal description logic $\mathcal{U}$. In fact, we proved that in this case the computational complexity is essentially the same as the one of deciding ordinary subsumption between two concepts with respect to acyclic terminologies in the *minimal* concept language.[5]

We also investigated the consequences of incorporating some limited kind of incomplete knowledge by means of Reiter's *null values* [Reiter,1984]. It turned out that, when presupposing P $\neq$ NP, admitting of null values causes intractability, even in case of $\mathcal{ALC}$. Thus our results suggest that the main source of computational complexity of terminological reasoning seems to be the ability to express *incomplete* knowledge.

5 Description Logics as Tractable Query Languages for Databases

Another interpretation of our results is that, when taken together with the least and greatest fixed point semantics, the universal concept language $\mathcal{U}$ can serve as a powerful but tractable query language for relational databases comprising solely unary and binary relations.[6] From this point of view terminologies are to be thought of as defining so-called *views*, possibly defined recursively.

At this very point, it is important to note that the universal description logic $\mathcal{U}$ is so strong in expressive power that it is even capable of *accurately* defining concepts such as directed acyclic graphs (*DAG*s), trees, or binary trees. The powerful role forming primitives of $\mathcal{U}$ actually admit of plausible and nonrecursive definitions of these concepts. As every finite graph can uniquely be represented by a physical knowledge base in a completely straightforward manner, these concepts provide views which can be used to extract from a huge collection of (connected) directed graphs exactly those which are acyclic or those which are trees or binary trees. If we additionally have recursive concept introductions along

[4] In this context, a *definition* is meant to be the subterminology of $\mathcal{T}$ which contains exactly those concept introductions which are involved in the recursion.

[5] Technically speaking, in this case deciding $\mathcal{V} \models_{\mathcal{T}} Q$ in $\mathcal{U}$ is co-NP-complete.

[6] Note that unary and binary relations do suffice as far as only object-oriented databases are concerned.

$$
\begin{array}{rcl}
\mathit{DirectedGraph} & \doteq & \forall \mathit{connected}\colon \mathit{Vertex} \\
\mathit{connected} & \doteq & (\mathit{edge} \sqcup \mathit{edge}^{-1})^{*} \\
\mathit{Acyclic} & \doteq & \forall \mathit{connected}\colon(\mathit{edge}^{+} \leq \neg\epsilon) \\
\mathit{DAG} & \doteq & \mathit{DirectedGraph} \sqcap \mathit{Acyclic} \\
\mathit{Tree} & \doteq & \mathit{DAG} \sqcap \forall \mathit{edge}^{*}\colon\exists^{\leq 1}\mathit{edge}^{-1}\colon \mathit{Vertex} \\
\mathit{BinaryTree} & \doteq & \mathit{Tree} \sqcap \forall \mathit{edge}^{*}\colon\exists^{\leq 2}\mathit{edge}\colon \mathit{Vertex} \\
\mathit{AndOrGraph} & \doteq & \mathit{DirectedGraph} \\
 & \sqcap & \forall \mathit{connected}\colon \mathit{AndOrVertex} \\
\mathit{AndOrVertex} & \doteq & \mathit{AndVertex} \sqcap \neg \mathit{OrVertex} \\
 & \sqcup & \mathit{OrVertex} \sqcap \neg \mathit{AndVertex} \\
\mathit{Successful} & \doteq & \neg\exists \mathit{edge}\colon \mathit{Vertex} \\
 & \sqcup & \mathit{AndVertex} \sqcap \forall \mathit{edge}\colon \mathit{Successful} \\
 & \sqcup & \mathit{OrVertex} \sqcap \exists \mathit{edge}\colon \mathit{Successful}
\end{array}
$$

Figure 6: A terminology of $\mathcal{U}$.

with least fixed point semantics at our disposal, we may even extract from a finite and-or-graph G (or a collection of such) exactly the *successful* vertices, i.e., those vertices which are a root of an acyclic subgraph G_s of G such that every and-vertex of G_s has exactly those edges it has in G and, moreover, every or-vertex has at least one of those edges it has in G. Figure 6 gives the terminology of $\mathcal{U}$ defining all the concepts mentioned in this section, where the recursive concept introduction of *Successful* should be given least fixed point semantics. This is just to demonstrate that even though the model checking approach to terminological knowledge representation does make it possible to answer queries in polynomial time, there are actually nontrivial inferences to perform.

Acknowledgements

I would like to thank Martin Buchheit for valuable comments on earlier drafts of the abstract.

References

[Brachman and Schmolze, 1985] R. J. Brachman and J. G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2):171–216, 1985.

[Chandra and Merlin, 1977] A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational databases. In *Proc. of the 9th ACM Symposium on Theory of Computing*, pages 77–90, 1977.

[Church, 1936] A. Church. A note on the Entscheidungsproblem. *Journal of Symbolic Logic*, 1:40–41, 1936.

[Emerson and Lei, 1986] E. A. Emerson and Ch. -L. Lei. Efficient model checking in fragments of the propositional mu-calculus (extended abstract). In *Proc. of the 1st IEEE Symposium on Logic in Computer Science*, pages 267–278, Boston, Mass., 1986.

[Halpern and Vardi, 1991] J. Y. Halpern and M. Y. Vardi. Model checking vs. theorem proving: A manifesto. In *Proc. of the 2nd International Conference on Principles of Knowledge Representation and Reasoning*, pages 325–334, Cambridge, Mass., 1991.

[McCarthy, 1968] J. McCarthy. Programs with common sense. In M. Minsky, editor, *Semantic Information Processing*, pages 403–418. MIT Press, Cambridge, Mass., 1968.

[Nebel, 1990] B. Nebel. Terminological Reasoning is Inherently Intractable. *Artificial Intelligence*, 43:235–249, 1990.

[Nebel, 1991] B. Nebel. Terminological cycles: Semantics and computational properties. In J. Sowa, editor, *Formal Aspects of Semantic Networks*, pages 331–361. Morgan Kaufmann, San Mateo, Cal., 1991.

[Patel-Schneider, 1987] P. F. Patel-Schneider. *Decidable, Logic-Based Knowledge Representation*. PhD thesis, University of Toronto, Toronto, Ont., 1987. Computer Science Department, Technical Report 201/87.

[Reiter, 1984] R. Reiter. Towards a logical reconstruction of relational database theory. In M. L. Brodie, J. Mylopoulos, and J. W. Schmidt, editors, *On Conceptual Modeling*, pages 191–233. Springer-Verlag, Berlin, FRG, 1984.

[Schild, 1988] K. Schild. Undecidability of subsumption in $\mathcal{U}$. KIT Report 67, Department of Computer Science, Technische Universität Berlin, Berlin, FRG, 1988.

[Schild, 1991] K. Schild. A correspondence theory for terminological logics: Preliminary report. In *Proc. of the 12th International Joint Conference on Artificial Intelligence*, pages 466–471, Sydney, Australia, 1991.

[Schild, 1993] K. Schild. Terminological cycles and the propositional μ-calculus. DFKI Research Report RR-93-18, German Research Center for Artificial Intelligence (DFKI), Saarbrücken, FRG, April 1993.

[Schild, 1994a] Klaus Schild. Terminological cycles and the propositional μ-calculus. In *Proc. of the 4th International Conference on Principles of Knowledge Representation and Reasoning*, Bonn, FRG, 1994. Forthcoming.

[Schild, 1994b] K. Schild. Tractable reasoning in a universal description logic. DFKI Research Report, German Research Center for Artificial Intelligence (DFKI), Saarbrücken, FRG, 1994. Forthcoming.

[Schmidt-Schauß and Smolka, 1991] M. Schmidt-Schauß and G. Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26, 1991.

[Schmidt-Schauß, 1989] M. Schmidt-Schauß. Subsumption in KL-ONE is undecidable. In *Proc. of the 1st International Conference on Principles of Knowledge Representation and Reasoning*, pages 421–431, Toronto, Ont., 1989.

[Trahtenbrot, 1963] B. A. Trahtenbrot. Impossibility of an algorithm for the decision problem in finite classes. *American Mathematical Society Translation Series*, 23(2):1–5, 1963.

[Woods and Schmolze, 1992] W. A. Woods and J. G. Schmolze. The KL-ONE family. In F.W. Lehmann, editor, *Semantic Networks in Artificial Intelligence*, pages 133–178. Pergamon Press, 1992.

Integrating Terminologies into F(rames)-Logic Knowledge Bases

Mira Balaban
Dept. of Mathematics and Computer Science
Ben-Gurion University of the Negev
P.O.B. 653, Beer-Sheva 84105, Israel
mira@bengus.bitnet (972)-57-461622

1 Introduction

In a previous paper ([1]) we suggested to use the F-logic approach of [6] as a basis for a rich description language that can be termed DFL (for Description F-logic Language). It was emphasized that DFL would enjoy the advantage of a full fledged logic that integrates notions of description, reasoning, object-orientation and logic programming. Three advantages were singled out:

1. DFL can serve as a unifying formalism for current description languages.
2. DFL can extend the expressivity of current DLs.
3. DFL provides a smooth integration with other frameworks of representation and reasoning.

In [1] we dealt, mainly, with the first two points. We showed that a DFL preserves the descriptional nature of DLs, since an F-Logic's ontology is already a standard DL's ontology. In particular, an F-Logic's ontology preserves the fundamental status of *concepts*, *objects*, *roles*, *terminological operations*, and *subsumption*. Moreover, fine terminological operations can be introduced, since F-logic enables fine distinctions among such notions as *applicability* of a role (i.e., appropriately typed), to *definability* of a role (i.e., a role is defined at an object, but its value may not be known), to *being defined and known*. For example, the standard operator **some**, can be distinguished from the usually equal **at-least1** operator, by adopting either the *applicable* meaning, or the *defined* meaning (the latter is equal to **at-least0**). In [1] it was also shown that replacing the DL's set-theoretic semantics by F-logic's semantics preserves subsumption, and examples for prospect expressivity of DFLs were presented.

In this paper we investigate the latter, i.e., the integration of a terminological component into an F-logic KB, yielding a DFL Knowledge Base (KB). We show that a DFL knowledge base can be viewed as having the standad two parts of a DL KB, i.e., a terminology (T-box) and an assertional (A-box) component. Viewed this way, the contribution of F-logic is in providing a rather expressive A-box language that smoothly integrates with the T-box language. The reason is that the T-box and the A-box share the powerful F-logic semantics. Moreover, the semantics and inferencing can build on common approaches. We claim that DFL gets closer to the idea of *rational management* of [3].

Most DL researchers believe that terminological reasoners should be used in complex applications ([10; 11; 7; 13; 4; 16; 3; 8]). A major issue in designing integration schemes is to avoid usual problems of **mismatch**. In [1] we proved the DFL correctness of DL subsumption algorithms, presented in [14; 12; 9]. Hence, computationally nothing is lost by considering a description language as a subset of F-logic, and standard computation methods apply without any meaningful changes. Based on these results we discuss below the nature of an integrated DFL knowledge base, define operational semantics for DFL, and suggest an integrated inference procedure. An Appendix that shortly describes F-Logic is also attached.

2 The Nature of an Integrated DFL Knowledge Base

A DL knowledge base has the following main parts:

1. A terminology – which includes definitions of concepts and roles, and definitions of primitive concepts and roles. The later are explicit introductions of primitive names into the texonomy.
2. Rules – which are declarations of subsumption between concept terms. They are not material implications in the regular meaning.
3. Descriptions – which are introductions of individual objects into the taxonomy.

An rule based F-logic KB, such as an F-logic program, might have the following components:

1. An IS-A hierarchy declaration – which includes rules (and facts) whose head is an IS-A assertion.
2. A signature (type) declaration – which includes rules (and facts) whose head is a signature expression.
3. An object base definition – which includes rules (and facts) whose head is either a predication, or data (method) assertions.
4. An equalities component.

A DFL knowledge base will have in addition:

- A definition of terminological operators.

DL KB	DFL KB
Terminological definitions	A terminology component
Primitive definitions	Part of the IS-A hierarchy declaration
Rules	Part of the IS-A hierarchy declaration
Descriptions	Part of the IS-A hierarchy declaration

Table 1: Correspondence between DL KB and DFL KB

- A terminology component.

Table 1 shows the tentative correspondence between the components of a DL KB to those of a DFL KB: It appears that a DFL KB is a DL KB with a very expressive A-box, in which the concepts and roles preserve their descriptional nature (and are not transformed into predicates). A DFL KB adds to a regular DL KB the signature component, the equalities component, and the rules for IS-A and role assertions. The correspondence is tentative since our experience shows that there is information that is more naturally expressed within an expressive assertional component. For example, the "typing" of a role, which can be expressed in BACK via **domain** and **range** restrictions, is more naturally expressed as a signature declaration for the role. In [2] an extensive DFL example is presented, following the industrial plants example from the BACK manual ([5]). We particularly concentrate on the contribution of the smooth integration of a terminology and rules within a single, uniform KB. We demonstrate several possible inferences. For example:

1. Inferencing using F-logic rules:
   Suppose that we have a rule saying that a location in which a waste produced by a dangerous plant is buried, is a risky place. Then any true ground instance of this rule, can extend the taxonomy with a new risky place. Using F-logic notation:
   $L :: risky\_place \longleftarrow$
   $Y :: dangerous\_plant[produces \twoheadrightarrow X]$,
   $X[buried\_at \rightarrow L]$
   If *dangerous_plant* is a defined concept (as in the BACK manual), and *buried_at* is a primitive role with signature: $product[buried\_at \Rightarrow (location)]$, then from the contingent knowledge:
   $p1 :: dangerous\_plant$, $p1[produces \twoheadrightarrow waste1]$, and $waste1[buried\_at \rightarrow l]$ we can infer:
   $l :: risky\_place$,
   and extend the taxonomy with this information.
2. Classification based on contingent knowledge:
   Suppose that *risky_place* is a defined concept:
   $risky\_place :=$
   $\mathbf{and}(place, \mathbf{some}(contains, toxic\_waste))$,
   and we have the contingent facts about the place *dump*:
   $dump :: place$, $dump[contains \twoheadrightarrow \{product1\}]$, and $product1 :: toxic\_waste$. Then based on the definition of the some terminological operator, *dump* can be classified as a *risky_place*:
   $dump :: risky\_place$.
3. Iteration between contingent and classification inference. In the full paper.

3 Semantics

The semantics of a DL knowledge base is, traditionally, defined as all models of the terminology (T-box) in which the assertional (contingent) information (A-box) holds ([9]). The T-box provides complete analytic definitions of concepts and roles, while the A-box includes contingently known facts. Hence, DL inferencing operates under the *open world assumption*. The semantics of logic programs, on the other hand, is traditionally defined under the *closed world assumption*, where information is assumed to be complete, and facts that are not implied are assumed false.

The difference between the two approaches is made clear when observing the definition of the **all** operator:

$$\begin{aligned} C_1 \; :: \; & \mathbf{all}(\; R,\; C\;) \;\equiv \\ & \forall C_2,\; (\; C_1[R\bullet\!\!\twoheadrightarrow \{C_2\}] \;\longrightarrow\; C_2 :: C\;). \end{aligned}$$

Under the closed world assumption, if all ground instances of the right hand side hold in a canonical model of a knowledge base, then also $C_1 :: \mathbf{all}(\; R,\; C\;)$ hold. This conclusion is in contradiction to the DLs approach, since it derives from contingent assertions about objects, and not from their essential terminological properties.

The question is what should be the semantics of an integrated DFL KB. Straightforward definitions such as: "all models of the terminology that are canonical models of the contingent part" are not useful, since the canonical model of the rule base will exclude facts that must hold in a model of the terminology. Yet, an operational semantics can be defined, following the intuition of *iterated inference*, where F-logic reasoning and classification are interleaved. We motivate the definition of the operational semantics with an example:

Terminology:
$a := \mathbf{and}(\; \mathbf{all}(r_1, b), d\;)$
$b := \mathbf{and}(\; \mathbf{all}(r_2, c), \mathbf{some}(r_3, d)\;)$
IS-A assertion component:
$o_1 :: b$
$X :: a \longleftarrow X :: b, X[Y \rightarrow Z]$
Roles' assertion component:
$o_1[r_1 \rightarrow o_2]$
$X[r_2 \rightarrow Y] \longleftarrow X :: b, Y :: d$
The canonical model of the A-box (the IS-A and the roles' components) is:
$o_1 :: b, \quad o_1[r_1 \rightarrow o_2], \quad o_1 :: a.$

First iteration: Using classification we can infer:
$o_2 :: b$, $o_1 :: d$, $o_1[r_3 \twoheadrightarrow sk_1]$, $sk_1 :: d$, $o_2[r_3 \twoheadrightarrow sk_2]$, $sk_2 :: d$.
Second iteration: Using A-box rules we can infer:
$o_2 :: a$, $o_2[r_2 \rightarrow o_1]$, $o_1[r_2 \rightarrow o_1]$.
Third iteration: Using classification we can infer:
$o_2 :: d$, $o_1 :: c$.
Forth iteration: Using A-box rules we can infer:
$o_2[r_2 \rightarrow o_2]$, $o_1[r_2 \rightarrow o_2]$, $o_1 \doteq o_2$, $o_2[r_1 \rightarrow o_2]$, $o_2[r_1 \rightarrow o_1]$, $o_1[r_1 \rightarrow o_1]$, $o_1[r_3 \twoheadrightarrow \{sk_2\}]$, $o_2[r_3 \twoheadrightarrow \{sk_1\}]$, $o_2 :: c$.
Note the following properties of the process:

1. No new definitions are entered into the terminology.
2. The classification step yields new A-box facts, but not new A-box rules.
3. The introduced circularity involves just the assertional component.
4. The new terms entered via the A-box reasoning are primitive terms of the taxonomy.

The inference process is summarized in the following definitions:
Let F be a set of F-logic facts, R be a set of F-logic rules, *T-box* be a terminology, and DEF be the definitions of terminological operators. Denote *A-box* $= F \cup R$, and $KB =$ *A-box*$\cup$*T-box*$\cup DEF$. Let L be a canonical model of *A-box*. An F-logic's H-structure (Herbrand structure) is a set of ground molecules (formulae without connectives and quantifiers), closed under logical implication (see [6]). Note that a canonical model is an H-structure.

Definition 3.1 *CLASSIFY is the following operator from H-structures to H-structures:*

$$CLASSIFY(L) = \{m \ \ / \ m \textit{ is a ground molecule}; \ L \cup T\textit{-box} \cup DEF \models m; \ m \textit{ does not include terminological operators}\}$$

Proposition 3.2 *For an H-structure L, CLASSIFY(L) is also an H-structure.*

Definition 3.3 T_A *is the following operator from H-structures to H-structures:* $T_A(L)$ *is the smallest set satisfying:*

1. $L \subseteq T_A(L)$.

2. $T_A(L)$ *includes the set:*

$\{m$ */ m is the head of some rule in R :*
$m \longleftarrow \langle body \rangle$, *and* $\langle body \rangle$ *is true in L*
(Precise definition depends on the syntax of rule bodies in R) $\}$

3. $T_A(L)$ *is closed under logical implication.*

Proposition 3.4 *For an H-structure L,* $T_A(L)$ *is also an H-structure.*

Definition 3.5 *The* operational semantics *of a KB, denoted* OP_{KB}, *is defined as the countably-infinite iteration:*

$$(\ (\ T_A \circ CLASSIFY\ )\ \uparrow \omega\ )(L)$$

Proposition 3.6 OP_{KB} *is a Herbrand model of KB.*

4 Inference

The interleaved operational semantics suggests a similar inference procedure, where terminological classification and A-box resoning are interleaved. In this subsection we demonstrate such inferencing, for the industrial plants example, using the *magic-sets* bottom up evaluation approach ([15]), for the A-box inferencing.

References

[1] M. Balaban. The f-logic approach for description languages. Technical Report FC 93-02, Department of Mathematics and Computer Science, Ben-Gurion University, Beer Sheva, Israel, 1993. To appear in Annals of Mathematics and Artificial Intelligence.

[2] M. Balaban. Integrating terminologies into f(rames)-logic knowledge bases. Technical Report in preparation, Department of Mathematics and Computer Science, Ben-Gurion University, Beer Sheva, Israel, 1994.

[3] J. Doyle and R. Patil. Two theses of knowledge representation: Language restrictions, taxonomic classification, and the utility of representation services. *J. of Artificial Intelligence*, 48(3):261–297, 1991.

[4] P. Hanschke. How to benefit from terminological logics. In *Working notes, AAAI Fall Symposium on Issues in Description Logics*, pages 45–48, 1992.

[5] T. Hoppe, C. Kindermann, J. Quantz, A. Schmiedel, and M. Fischer. Back v5: Tutorial and manual. Technical Report KIT - report 100, Technische Universitat Berlin, March 1993.

[6] M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. Technical Report #93/06, Dept. of Computer Sciencee, SUNY at Stony Brook, April 1993. To appear in JACM.

[7] R. MacGregor. What's needed to make a description logic a good kr citizen. In *Working notes, AAAI Fall Symposium on Issues in Description Logics*, pages 53–55, 1992.

[8] B. Mark. 10 years don't mean nothing. In *Working notes, AAAI Fall Symposium on Issues in Description Logics*, pages 59–60, 1992.

[9] B. Nebel. *Reasoning and Revision in Hybrid Representation Systems*. Dissertation, University of Saarlands, Saarbrücken, 1989.

[10] P. Patel-Schneider. Partial reasoning in knowledge representation systems based on description logics. In *Working notes, AAAI Fall Symposium on Issues in Description Logics*, pages 74–75, 1992.

[11] A. B. Pfahringer. The logical way to build a dl-bsed kr system. In *Working notes, AAAI Fall Symposium on Issues in Description Logics*, pages 76–77, 1992.

[12] L. Resnick, A. Borgida, R. Brachman, D. McGuinness, and P. Patel-Schneider. Classic description and reference manual for common lisp implementation. Technical Report Version 1.02, AT&T Bell Labs, 1990.

[13] A. Schmiedel. For a more expressive query language. In *Working notes, AAAI Fall Symposium on Issues in Description Logics*, pages 98–102, 1992.

[14] M. Schmidt-Schauß and G. Smolka. Attributive concept descriptions with complements. *J. of Artificial Intelligence*, 48(1):1–26, 1991.

[15] J. Ullman. *Principles of Database and Knowledge-base Systems.* Computer Science Press, 1989.

[16] W. Woods. Understanding subsumption and taxonomy: A framework for progress. In J. Sowa, editor, *Principles of Semantic Networks: Explorations in the Representation of Knowledge*, pages 45–94. Morgan Kaufmann, 1991.

5 Appendix: F-Logic

An F-logic domain U consists of *objects* and *methods*. In addition, there are *object constructors*, which are functions defined on objects, a *partial ordering* $\preceq_U$ on objects, that stands for the subset relationship, and a binary relation $\in_U$ on objects, that stands for the membership relationship. A condition set on $\preceq_U$ and $\in_U$ guarantees that membership in an object is extended to a super-class object. The underlying intensional approach assumes that the "essence" of an object lies in its behavior.

Methods are partial functions of objects. There are single-valued (scalar) and set-valued methods. Methods describe the *behavior* of objects, and provide *information* about objects. For example, *spouse-of*, *children-of*, and information on a bank-account, can be captured by methods. The *children-of* method is an example of a method that takes additional arguments: "children-of an object o_1 with another object o_2", is an application of the method on o_1, with o_2 as an extra argument (or in the *context* of o_2). A method like *spouse-of*, that does not take additional arguments (i.e., a function of one argument) is called an *attribute*. *Name-of*, *age-of*, *address-of*, are all attributes. Methods that describe actions, like *buy*, *meet*, *treat*, etc., can typically take additional arguments, for all the parameters of the actions. These are n-ary functions, ($n > 1$).

Methods are also classified into *inheritable* and *non-inheritable*. For example, *color* is an inheritable attribute of bears, *averageSalary* is a non-inheritable attribute of faculty, and *children-of* is a non-inheritable set-valued method of Mary (since various specializations of Mary, e.g., MaryAsChild need not inherit the value of *children-of* at Mary). The inference machinery uses the distinction between inheritable to non-inheritable methods for propagating values of inheritable methods, down the $\preceq$ hierarchy, as long as no overwriting is caused. Inheritance extends also into the $\in$ relationship, but is blocked after one step. This way, Mary, being a student, inherits the *registered-in-college* attribute-value pair, while its MaryAsChild specialization does not inherit this property; *studentCommittee* can inherit the *committee-size* property of *univCommittee*, but inheritance does not extend to Mary.

The "secret" behind F-logic is the *high-logization* of methods: Methods and types are **reified** by their object-names, and quantification over them is carried just over their object-names. Section 5.2 shortly summarizes the main ideas in the semantics of F-logic.

Observation: An F-logic ontology is already a description language ontology.

In the rest of this paper we use the DL terminology: An *attribute-feature* is a 1-ary single-valued method, an *attribute-role* is a 1-ary set-valued method, a *non-attribute-feature* is an n-ary single-valued method ($n > 1$), a *non-attribute-role* is an n-ary set-valued method ($n > 1$), a *feature* is a single-valued method of any arity, a *role* is a set-valued method of any arity. The term *method* refers to features and roles, indifferently.

5.1 F-logic Syntax

The terms of F-logic are expressions that denote objects in the domain. For example, *mary*, 3, **and**(*polygon, 3Sides*), **and**(*polygon, 3Angles*), *cars(employees(bgu))*, *cars-of(employees)*, are all, *id-terms*, denoting objects. The atomic formulae of F-logic, called F-molecules, are of three kinds: *is-a* F-molecules, *data* F-molecules, and *signature* F-molecules.

1. Is-a F-molecules map to the partial ordering, and to the membership relation on U. Table 2 presents some is-a F-molecules.

2. Data F-molecules are assertions about the values that methods (features, roles) get on objects. Table 3 presents some data F-molecules.

3. Signature F-molecules are assertions about the types of features and roles. For example,

   X[*husband-of* $\Rightarrow$ (*male*)]
   X[*children* @ Y $\Rrightarrow$ (*person*)]

   where X and Y are assumed to be universally quantified, assert that values of the *husband-of* feature must be males, and the values of the *children* role must be of type *person*. The rule:

   X : **and**(*female, married*) $\longleftarrow$ X[*husband-of* $\Rightarrow$ ()]

   where X is assumed to be universally quantified, asserts that if the feature *husband-of* applies to an object o denoted by X, then o must be a member of the **and**(*female, married*) class, i.e., a married female.

F-logic includes also regular atomic formulae of a first order language. Its formulae are constructed using connectives and quantifiers in the usual first order manner. An example of knowledge base browsing:

Is-a F-molecules	Meaning
mary : woman	The object denoted by *mary* is $\in_U$ related to the one denoted by *woman*.
woman :: person	The object denoted by *woman* is $\preceq_U$ related to the one denoted by *person*.
and(*polygon, 3Sides*) :: *polygon*	Subset relationship ($\preceq_U$) between the denotations of the two id-terms.

Table 2: Is-a F-molucules

Data F-molecules	Meaning	Explanation
mary[*husband-of* $\rightarrow$ *fred*]	Fred is Mary's husband.	*husband-of* is an *attribute-feature.*
bear[*color* $\bullet\!\!\rightarrow$ *grey*]	Bears are grey.	*color* is an *inheritable attribute-feature.*
mary[*teach* $\twoheadrightarrow$ {*aut.*, *graph.*}]	Mary teaches aut. & graph.	*teach* is an *attribute-role.*
bear[*color* @ north $\bullet\!\!\rightarrow$ *white*]	Northern bears are white.	*color* is an *inheritable non-attribute-feature.*
son(m)[*children*@j $\twoheadrightarrow$ {*pat*}]	Pat is a child of m's son with j.	*children* is a *non-attribute-role.*

Table 3: Data F-molecules

interestingAttributes(X)[*attributes* $\twoheadrightarrow$ *L*] $\longleftarrow$
X : *faculty*[*L* $\rightarrow$ *Z* : *person*]
interestingAttributes(X)[*attributes* $\twoheadrightarrow$ *L*] $\longleftarrow$
X : *faculty*[*L* $\twoheadrightarrow$ *Z* : *person*]
These rules define, for every member, o, of the *faculty* object, a new object, *interestingAttributes*(o), with a set-valued attribute, *attributes*, whose value at that object is the set of all attributes of *o* that have a *person* value.

5.2 F-logic Semantics

In the full paper.

YAEF: Yet Another Epistemic Formalism

Philipp Hanschke
sd&m GmbH & Co. KG
Am Schimmersfeld 7a
40880 Ratingen
Germany
Philipp.Hanschke@uni-duesseldorf.de

It is well known that terminological formalisms are both useful for representing terminological knowledge and limited with respect to their expressive power. The main issue of this paper is the homogeneous integration of the special-purpose reasoning power of a terminological formalism with the general-purpose representation and reasoning power of a semidecidable (computationally complete) rule formalism. Therefore a declarative generic rule scheme is developed that can be applied to a terminological formalism such as ALC or the extended formalisms with concrete domains as described in [1; 6].

1 Introduction

The proposed generic rule formalism is based on rules of the form

$$\phi_0 \rightsquigarrow \phi_1 | \ldots | \phi_n \qquad (1)$$

where $n \geq 0$ and ϕ_i are formulas of a first-order logic satisfying certain requirements and the symbol "$\rightsquigarrow$" stands for a weak form of implication explained later. The formalism is parametrized by the first-order logic which is referred to as the *condition formalism*. For instance, it has been shown in [5] that terminological formalisms as mentioned above can be seen as a condition formalism. Term equations (and negated term equations) induce another relevant condition logic. The operational semantics of the generic rule formalism generalizes the way productions rules are applied to a fact base. If a rule is triggered one of the ϕ_i, $1 \leq i \leq n$, in the head is (don't know) nondeterministically selected and added to the fact base.

Informally, such a rule says "if ϕ_0 is believed, then one of ϕ_1, ϕ_2, ...,ϕ_n is believed." For $n = 0$ the rule is a *denial* saying that whenever ϕ_0 is believed, the current state is inconsistent. For $n = 1$ the rules are very close to production rules. If ϕ_0 is very simple and $n > 1$, the operational semantics of these rules has much in common with SLD resolution.

Hence, this formalism combines deterministic, data-driven, bottom-up reasoning with non-deterministic, goal-directed, top-down search.

1.1 Operational Semantics

The operational semantics can be considered as production rule-like inferences combined with backtracking search. First of all, there is a fact base $\mathcal{A}_0$. Objects occurring in the fact base are substituted for variables in the premise of a rule. Then it is checked whether the fact base entails the instantiated premise with respect to the condition logic. If not, another instantiation of a premise of a rule with objects in the fact base is tried. Otherwise, one of the alternatives in the head of the instantiated rule is added to the fact base. Free variables in the head are considered as being existentially quantified and new objects are introduced to instantiate them. If the fact base gets inconsistent with respect to the condition logic, backtracking takes place: The computation resumes at the most recent point where another alternative in the head of a rule can be selected. If all instantiations of rules with a premise that is entailed by the current fact base $\mathcal{A}$ have been applied, and if the current fact base $\mathcal{A}$ is consistent, $\mathcal{A}$ is an answer computed by the set of rules for $\mathcal{A}_0$.

Note that the generic inference algorithm of the rule scheme just requires the functions consistent and entails to be provided by the condition formalism.

1.2 Logical Reading

The rules should not be regarded as logical implications in the classical sense. For example, the operational semantics of the formalism does not take care of contra positions: If there is a rule $\phi \rightsquigarrow \phi'$ and $\neg\phi'$ is believed, it will not derive $\neg\phi$. The operator $\rightsquigarrow$ also differs from classical implication in the following sense: If $\phi \vee \phi'$ holds and there are rules $\phi \rightsquigarrow \phi''$ and $\phi' \rightsquigarrow \phi''$, then ϕ'' is not derived by these rules. Finally, assume that there is a rule $\phi(x) \rightsquigarrow \phi'(x)$ with a variable x, which is implicitly universally quantified. Then the rule is only triggered if there is an object a in the current fact base such that $\phi(a)$ is implied by the fact base. Thus, all variables in the premise of a rule have to be instantiated by objects which occur explicitly in the fact base.

These restrictions enable efficient processing of the rules. The *trigger rules* in [2; 4; 9] have similar restrictions in their operational semantics. A trigger rule $A \rightsquigarrow B'$ can be regarded as a special case of (1) where ϕ_0 and ϕ_1 are concepts and $n = 1$. In [3] a semantics based on the epistemic operator K, standing for 'knows', is proposed which coincides with the operational semantics. Levesque has introduced the K operator in his ask and tell framework [7].

In [8] Lifschitz relates minimal believe logics to the semantics of some logic programming formalisms including general, disjunctive logic programs. He replaced the letter K by the letter B reflecting his preference of 'believe' in place of 'knowledge' as the intuition behind his logic.

The idea of a *minimal believe logic* is also the key to the semantics of the rule formalism introduced here. However, non of the mentioned formalizations of an epistemic logic is appropriate as the basis for a model-theoretic semantics of the rules. Compared to [3] the formalism considered here offers more complex premises, disjunctions in the conclusions, and variables occurring only in the head. It is also more general, because it tolerates the possible presence of equality "=" and the possible absence of a unique-name assumption.

Premises with more then one variable together with the absence of the unique-name assumption induce a major technical problem. Consider, for instance, a fact base just consisting of the fact $p(x,y)$, which is associated with the epistemic formula $\exists x,y(Bp(x,y))$, and a program consisting of a rule $p(x,x) \leadsto q$, which is associated with $\forall x(Bp(x,x) \Rightarrow Bq)$. Note, that the rule cannot be applied to the fact base. The soundness result below implies that each epistemic model of (the computed answer) $\exists x,y(Bp(x,y))$ satisfies the program and the fact base.

What is an epistemic model? Roughly, the formalizations of epistemic logics in [12; 8; 7; 3] all have the same structure. The following definitions can be seen as a simplification of the logic presented in [8] where an additional modal operator *not* is considered. An epistemic interpretation $(\mathcal{A},\mathcal{M})$, which is also referred to as a *structure*, consists of an interpretation $\mathcal{A}$ of an underlying first-order logic and a set $\mathcal{M}$ of such interpretations where all interpretations $\mathcal{J} \in \mathcal{M}$ and $\mathcal{A}$ share the same domain, D say. For the parameters $d \in D$ names $n_1, n_2, \cdots$ are introduced. These names are in a one-to-one correspondence to the parameters in D. The notion of satisfiability for structures is inductively defined as follows. If a structure $(\mathcal{A},\mathcal{M})$ satisfies an epistemic formula ϕ, this is written as $\mathcal{M},\mathcal{A} \models \phi$.

1. $\mathcal{M},\mathcal{A} \models \phi$:iff $\mathcal{A} \models \phi$, for a closed first-order formula ϕ.
2. $\mathcal{M},\mathcal{A} \models \phi \wedge \phi'$:iff $\mathcal{M},\mathcal{A} \models \phi$ and $\mathcal{M},\mathcal{A} \models \phi'$.
3. $\mathcal{M},\mathcal{A} \models \exists x(\phi(x))$:iff there exists a name n such that $\mathcal{M},\mathcal{A} \models \phi(n)$.
4. $\mathcal{M},\mathcal{A} \models \neg\phi$:iff not $\mathcal{M},\mathcal{A} \models \phi$.
5. $\mathcal{M},\mathcal{A} \models B\phi$:iff $\mathcal{J},\mathcal{M} \models \phi$, for all $\mathcal{J} \in \mathcal{M}$.

Then an *epistemic model* $(\mathcal{A},\mathcal{M})$ of ϕ is a structure with $\mathcal{M},\mathcal{A} \models \phi$ that is maximal with respect to $\leq$. Here $\leq$ is defined by $(\mathcal{A},\mathcal{M}) \leq (\mathcal{A}',\mathcal{M}')$:iff $\mathcal{M} \subseteq \mathcal{M}'$.[1]

In the example, let $(\mathcal{A},\mathcal{M})$ be a model of $\exists x,y(Bp(x,y))$. According to the definition there exist names n_1, n_2 such that $\mathcal{J} \models p(n_1,n_2)$, for all $\mathcal{J} \in \mathcal{M}$. Please observe that both sets of interpretations, defined below, induce epistemic models of $\exists x,y(Bp(x,y))$.

1. $\mathcal{M}_= := \{\mathcal{A} \mid \mathcal{A}$ is an interpretation over D and $(d,d) \in p^{\mathcal{A}}\}$, for some $d \in D$.
2. $\mathcal{M}_{\neq} := \{\mathcal{A} \mid \mathcal{A}$ is an interpretation over D and $(d_1,d_2) \in p^{\mathcal{A}}\}$, for some $d_1,d_2 \in D$ with $d_1 \neq d_2$.

Obviously, neither $\mathcal{M}_= \subseteq \mathcal{M}_{\neq}$ nor $\mathcal{M}_{\neq} \subseteq \mathcal{M}_=$, and $(\mathcal{A},\mathcal{M}_=)$ is an epistemic model of the rule $\forall x(Bp(x,x) \Rightarrow q)$—contrary to the desired soundness result.

The example suggests that the problem is related to an interplay of the modal operator and the quantifiers. There happens something interesting if the scopes of two existential quantifications interact with the scope of an occurrence of the modal operator.

Let $(\mathcal{A},\mathcal{M})$ be a structure. For $B(p \vee q)$ all ways to make $p \vee q$ true may be covered by $\mathcal{M}$. Similarly, $\mathcal{M} := \{\mathcal{A} \mid \mathcal{A}$ is an interpretation over D and $\exists x,y(p(x,y))$ is true in $\mathcal{A}\}$ covers all possible ways to satisfy $\exists x,y(p(x,y))$ with interpretations over a fixed domain D. But consider $\exists x,y(Bp(x,y))$. It is impossible that $\mathcal{M}$ covers all possibilities how $p(x,y)$ can be made true given that there are objects d_1 and d_2 for which it is *just*[2] required that (d_1,d_2) is in the extension of p. The set $\mathcal{M}$ must be incomplete in this respect, because selecting names n_1 and n_2 either with $n_1 = n_2$ or $n_1 \neq n_2$ to substitute for x and y is a commitment to either a set of type $\mathcal{M}_=$ or $\mathcal{M}_{\neq}$, respectively. Note that this problem does not occur with trigger rules.

In [5] epistemic logics are formalized using *partitions with infinite equivalence classes as interpretation domains and the notion of pre-variable assignments.* This conception enables an epistemic model to vary also over all possible variable assignments by assigning the elements of the range of a pre assignment to different equivalence classes in different interpretations. With these modifications it is straight forward to get adequate soundness and completenss results for the rule formalism.

References

[1] F. Baader and Ph. Hanschke. A scheme for integrating concrete domains into concept languages. In Mylopoulos and Reiter [10].

[2] R. Brachman, D. McGuinness, P. Pate-Schneider, and L. Resnick. Living with CLASSIC: When and how to use a KL-ONE-like language. In *Principles of Semantic Networks*. Morgan Kaufmann, 1991.

[3] F.M. Donini, D.N. Lenzerini, A. Schaerf, and W. Nutt. Adding epistemic operators to concept languages. In Nebel et al. [11].

[4] J. Edelmann and B. Owsnicki. Data models in knowledge representation systems: A case study. In *GWAI-86 and 2. Österreichische Artificial-Intelligence Tagung*, pages 69–74. Springer, 1986.

[1]The definitions of the other approaches vary in the treatment of $\mathcal{A}$ and $\mathcal{A}'$.

[2]In particular, there is nothing said about $x = y$ or $x \neq y$.

[5] Ph. Hanschke. *A Declarative Integration of Terminological, Constraint-based, Data-driven, and Goal-directed Reasoning.* PhD thesis, University of Kaiserslautern, Germany, 1989.

[6] Ph. Hanschke. Specifying role interaction in concept languages. In Nebel et al. [11].

[7] J. Levesque, H. Foundations of a functional approach to knowledge representation. *Artificial Intelligence*, 1984.

[8] V. Lifschitz. Minimal belief and negation as failure. In Mylopoulos and Reiter [10].

[9] R. MacGregor. A deductive pattern matcher. In *AAAI*, pages 403–408, 1988.

[10] John Mylopoulos and Ray Reiter, editors. *12th International Joint Conference on Artificial Intelligence*, 1991.

[11] B. Nebel, Ch. Rich, and W. Swartout, editors. *Third International Conference on Principles of Knowledge Representation and Reasoning (KR '92).* Morgan Kaufmann, 1992.

[12] R. Reiter. On asking what a database knows. In *Computational Logic Symposium Proceedings*, November 1990.

Refining the Structure of Terminological Systems: Terminology = Schema + Views*

M. Buchheit[1] and **F. M. Donini**[2] and **W. Nutt**[1] and **A. Schaerf**[2]

1. German Research Center for Artificial Intelligence (DFKI), Saarbrücken, Germany
{buchheit,nutt}@dfki.uni-sb.de

2. Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", Italy
{donini,aschaerf}@assi.dis.uniroma1.it

Abstract

Traditionally, the core of a Terminological Knowledge Representation System (TKRS) consists of a so-called TBox, where concepts are introduced, and an ABox, where facts about individuals are stated in terms of these concepts. This design has a drawback because in most applications the TBox has to meet two functions at a time: on the one hand, similar to a database schema, framelike structures with typing information are introduced through primitive concepts and primitive roles; on the other hand, views on the objects in the knowledge base are provided through defined concepts.

We propose to account for this conceptual separation by partitioning the TBox into two components for primitive and defined concepts, which we call the *schema* and the *view* part. We envision the two parts to differ with respect to the language for concepts, the statements allowed, and the semantics.

We argue that by this separation we achieve more conceptual clarity about the role of primitive and defined concepts and the semantics of terminological cycles. Moreover, three case studies show the computational benefits to be gained from the refined architecture.

1 Introduction

Research on terminological reasoning usually presupposes the following abstract architecture, which reflects quite well the structure of existing systems. There is a logical representation language that allows for two kinds of statements: in the TBox or *terminology*, concept descriptions are introduced, and in the ABox or *world description*, individuals are characterized in terms of concept membership and role relationship. This abstract architecture has been the basis for the design of systems, the development of algorithms, and the investigation of the computational properties of inferences.

Given this setting, there are three parameters that characterize a terminological system: (*i*) the language for concept descriptions, (*ii*) the form of the statements allowed, and (*iii*) the semantics given to concepts and statements. Research tried to improve systems by modifying these three parameters. But in all existing systems and almost all theoretical studies language and semantics have been kept uniform.[1]

The results of these studies were unsatisfactory in at least two respects. First, it seems that tractable inferences are only possible for languages with little expressivity. Second, no consensus has been reached about the semantics of terminological cycles, although in applications the need to model cyclic dependencies between classes of objects arises constantly.

Based on an ongoing study of applications of terminological systems, we suggest to refine the two-layered architecture consisting of TBox and ABox. Our goal is twofold: on the one hand we want to achieve more conceptual clarity about the role of primitive and defined concepts and the semantics of terminological cycles; on the other hand, we want to improve the tradeoff between expressivity and worst case complexity. Since our changes are not primarily motivated by mathematical considerations but by the way systems are used, we expect to come up with a more practical system design.

In the applications studied we found that the TBox has to meet two functions at a time. One is to declare frame-like structures by introducing primitive concepts and roles together with typing information like isa-relationships between concepts, or range restrictions and number restrictions of roles. *E.g.,* suppose we want to model a company environment. Then we may introduce the concept Employee as a specialization of Person, having exactly one name of type Name and at least one affiliation of type Department. This is similar to class declarations in object-oriented systems. For this purpose, a simple language is sufficient. Cycles occur naturally in modeling

*This work was partly supported by the Commission of the European Union under ESPRIT BRA 6810 (Compulog 2), by the German Ministry of Research and Technology under grant ITW 92-01 (TACOS), and by the CNR (Italian Research Council) under Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo, LdR "Ibridi."

[1]In [Lenzerini and Schaerf,1991] a combination of a weak language for ABoxes and a strong language for queries has been investigated.

tasks, *e.g.*, the boss of an Employee is also an Employee. Such declarations have no definitional import, they just restrict the set of possible interpretations.

The second function of a TBox is to define new concepts in terms of primitive ones by specifying necessary *and* sufficient conditions for concept membership. This can be seen as defining *abstractions* or *views* on the objects in the knowledge base. Defined concepts are important for querying the knowledge base and as left-hand sides of trigger rules. For this purpose we need more expressive languages. If cycles occur in this part they must have definitional import.

As a consequence of our analysis we propose to split the TBox into two components: one for declaring frame structures and one for defining views. By analogy to the structure of databases we call the first component the *schema* and the second the *view* part. We envision the two parts to differ with respect to the language, the form of statements, and the semantics of cycles.

The schema consists of a set of primitive concept introductions, formulated in the *schema language*, and the view part by a set of concept definitions, formulated in the *view language*. In general, the schema language will be less expressive than the view language. Since the role of statements in the schema is to restrict the interpretations we want to admit, first order semantics, which is also called descriptive semantics in this context (see Nebel 1991), is adequate for cycles occurring in the schema. For cycles in the view part, we propose to choose a semantics that defines concepts uniquely, *e.g.*, least or greatest fixpoint semantics.

The purpose of this work is not to present the full-fledged design of a new system but to explore the options that arise from the separation of TBoxes into schema and views. Among the benefits to be gained from this refinement are the following three. First, the new architecture has more parameters for improving systems, since language, form of statements, and semantics can be specified differently for schema and views. So we found a combination of schema and view language with polynomial inference procedures whereas merging the two languages into one would have led to intractability. Second, we believe that one of the obstacles to a consensus about the semantics of terminological cycles has been precisely the fact that no distinction has been made between primitive and defined concepts. Moreover, intractability results for cycles mostly refer to inferences with defined concepts. We proved that reasoning with cycles is easier when only primitive concepts are considered. Third, the refined architecture allows for more differentiated complexity measures, as shown later in the paper.

In the following section we outline our refined architecture for a TKRS, which comprises *three* parts: the *schema*, the *view taxonomy*, and the *world description*, which comprise primitive concepts, defined concepts and assertions in traditional systems. In the third section we show by three case studies that adding a simple schema with cycles to existing systems does not increase the complexity of reasoning.

2 The Refined Architecture

We start this section by a short reminder on concept languages. Then we discuss the form of statements and their semantics in the different components of a TKRS. Finally, we specify the reasoning services provided by each component and introduce different complexity measures for analyzing them.

2.1 Concept Languages

In concept languages, complex concepts (ranged over by C, D) and complex roles (ranged over by Q, R) can be built up from simpler ones using concept and role forming constructs (see Tables 1 and 2 a set of common constructs). The basic syntactic symbols are (*i*) *concept names*, which are divided into *schema names* (ranged over by A) and *view names* (ranged over by V), (*ii*) *role names* (ranged over by P), and (*iii*) *individual names* (ranged over by a, b). An *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of the *domain* $\Delta^{\mathcal{I}}$ and the *interpretation function* $\cdot^{\mathcal{I}}$, which maps every concept to a subset of $\Delta^{\mathcal{I}}$, every role to a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, and every individual to an element of $\Delta^{\mathcal{I}}$ such that $a^{\mathcal{I}} \neq b^{\mathcal{I}}$ for different individuals a, b (*Unique Name Assumption*). Complex concepts and roles are interpreted according to the semantics given in Tables 1 and 2, respectively.

In our architecture, there are two different concept languages in a TKRS, a *schema language* for expressing schema statements and a *view language* for formulating views and queries to the system.

2.2 The Three Components

We first focus our attention to the schema. The schema introduces concept and role names and states elementary type constraints. This can be achieved by *inclusion axioms* having one of the forms:

$$A \sqsubseteq D, \qquad P \sqsubseteq A_1 \times A_2,$$

where A, A_1, A_2 are schema names, P is a role name, and D is a concept of the schema language. Intuitively, the first axiom states that all instances of A are also instances of D. The second axiom states that the role P has domain A_1 and range A_2. A *schema* $\mathcal{S}$ consists of a finite set of schema axioms.

Inclusion axioms impose only necessary conditions for being an instance of the schema name on the left-hand side. For example, the axiom "Employee $\sqsubseteq$ Person" declares that every employee is a person, but does not give a sufficient condition for being an employee.

A schema may contain *cycles* through inclusion axioms (see Nebel 1991 for a formal definition). So one may state that the bosses of an employee are themselves employees, writing "Employee $\sqsubseteq$ ∀boss.Employee." In general, existing systems do not allow for terminological cycles, which is a serious restriction, since cycles are ubiquitous in domain models.

There are two questions related to cycles: the first is to fix the semantics and the second, based on this, to come up with a proper inference procedure. As to the

Construct Name	Syntax	Semantics
top	$\top$	$\Delta^{\mathcal{I}}$
singleton set	$\{a\}$	$\{a^{\mathcal{I}}\}$
intersection	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
union	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
negation	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
universal quantification	$\forall R.C$	$\{d_1 \mid \forall d_2 : (d_1, d_2) \in R^{\mathcal{I}} \rightarrow d_2 \in C^{\mathcal{I}}\}$
existential quantification	$\exists R.C$	$\{d_1 \mid \exists d_2 : (d_1, d_2) \in R^{\mathcal{I}} \wedge d_2 \in C^{\mathcal{I}}\}$
existential agreement	$\exists Q \doteq R$	$\{d_1 \mid \exists d_2.(d_1, d_2) \in Q^{\mathcal{I}} \wedge (d_1, d_2) \in R^{\mathcal{I}}\}$
number restrictions	$(\geq n\, R)$	$\{d_1 \mid \sharp\{d_2 \mid (d_1, d_2) \in R^{\mathcal{I}}\} \geq n\}$
	$(\leq n\, R)$	$\{d_1 \mid \sharp\{d_2 \mid (d_1, d_2) \in R^{\mathcal{I}}\} \leq n\}$

Table 1: Syntax and semantics of concept forming constructs.

Construct Name	Syntax	Semantics
inverse role	P^{-1}	$\{(d_1, d_2) \mid (d_2, d_1) \in P^{\mathcal{I}}\}$
role restriction	$(R{:}\, C)$	$\{(d_1, d_2) \mid (d_1, d_2) \in R^{\mathcal{I}} \wedge d_2 \in C^{\mathcal{I}}\}$
role chain	$Q \circ R$	$\{(d_1, d_3) \mid \exists d_2.(d_1, d_2) \in Q^{\mathcal{I}} \wedge (d_2, d_3) \in R^{\mathcal{I}}\}$
self	ϵ	$\{(d_1, d_1) \mid d_1 \in \Delta^{\mathcal{I}}\}$

Table 2: Syntax and semantics of role forming constructs.

semantics, we argue that axioms in the schema have the role of narrowing down the models we consider possible. Therefore, they should be interpreted under descriptive semantics, *i.e.*, like in first order logic: an interpretation $\mathcal{I}$ satisfies an axiom $A \sqsubseteq D$ if $A^{\mathcal{I}} \subseteq D^{\mathcal{I}}$, and it satisfies $P \sqsubseteq A_1 \times A_2$ if $P^{\mathcal{I}} \subseteq A_1^{\mathcal{I}} \times A_2^{\mathcal{I}}$. The interpretation $\mathcal{I}$ is a model of the schema $\mathcal{S}$ if it satisfies all axioms in $\mathcal{S}$. The problem of inferences will be dealt with in the next section.

The *view part* contains *view definitions* of the form

$$V \doteq C,$$

where V is a view name and C is a concept in the view language. Views provide abstractions by defining new classes of objects in terms of the concept and role names introduced in the schema. We refer to "$V \doteq C$" as the *definition* of V. The distinction between schema and view names is crucial for our architecture. It ensures the separation between schema and views.

A *view taxonomy* $\mathcal{V}$ is a finite set of view definitions such that (*i*) for each view name there is at most one definition, and (*ii*) each view name occurring on the right hand side of a definition has a definition in $\mathcal{V}$.

Differently from schema axioms, view definitions give necessary *and* sufficient conditions. As an example of a view, one can describe the bosses of the employee Bill as the instances of "BillsBosses $\doteq$ ∃boss-of.{BILL}."

Whether or not to allow cycles in view definitions is a delicate design decision. Differently from the schema, the role of cycles in the view part is to state recursive definitions. For example, if we want to describe the group of individuals that are above Bill in the hierarchy of bosses we can use the definition "BillsSuperBosses $\doteq$ BillsBosses ⊔ ∃boss-of.BillsSuperBosses." But note that this does not yield a definition if we assume descriptive semantics because for a fixed interpretation of BILL and of the role boss-of there may be several ways to interpret BillsSuperBosses in such a way that the above equality holds. In this example, we only obtain the intended meaning if we assume least fixpoint semantics. This observation holds more generally: if cycles are intended to uniquely define concepts then descriptive semantics is not suitable. However, least or greatest fixpoint semantics or, more generally, a semantics based on the μ-calculus yield unique definitions (see Schild 1994). Unfortunately, algorithms for subsumption of views under such semantics are known only for fragments of the concept language defined in Tables 1 and 2.

In this paper, we only deal with acyclic view taxonomies. In this case, the semantics of view definitions is straightforward. An interpretation $\mathcal{I}$ satisfies the definition $V \doteq C$ if $V^{\mathcal{I}} = C^{\mathcal{I}}$, and it is a model for a view taxonomy $\mathcal{V}$ if $\mathcal{I}$ satisfies all definitions in $\mathcal{V}$.

A state of affairs in the world is described by *assertions* of the form

$$C(a), \qquad R(a, b),$$

where C and R are concept and role descriptions in the view language. Assertions of the form $A(a)$ or $P(a, b)$, where A and P are names in the schema, resemble basic facts in a database. Assertions involving complex concepts are comparable to view updates.

A *world description* $\mathcal{W}$ is a finite set of assertions. The semantics is as usual: an interpretation $\mathcal{I}$ satisfies $C(a)$

if $a^\mathcal{I} \in A^\mathcal{I}$ and it satisfies $R(a,b)$ if $(a^\mathcal{I}, b^\mathcal{I}) \in R^\mathcal{I}$; it is a model of $\mathcal{W}$ if it satisfies every assertion in $\mathcal{W}$.

Summarizing, a knowledge base is a triple $\Sigma = \langle \mathcal{S}, \mathcal{V}, \mathcal{W} \rangle$, where $\mathcal{S}$ is a schema, $\mathcal{V}$ a view taxonomy, and $\mathcal{W}$ a world description. An interpretation $\mathcal{I}$ is a model of a knowledge base if it is a model of all three components.

2.3 Reasoning Services

For each component, there is a prototypical reasoning service to which the other services can be reduced.

Schema Validation: Given a schema $\mathcal{S}$, check whether there exists a model of $\mathcal{S}$ that interprets every schema name as a nonempty set.

View Subsumption: Given a schema $\mathcal{S}$, a view taxonomy $\mathcal{V}$, and view names V_1 and V_2, check whether $V_1^\mathcal{I} \subseteq V_2^\mathcal{I}$ for every model $\mathcal{I}$ of $\mathcal{S}$ and $\mathcal{V}$;

Instance Checking: Given a knowledge base Σ, an individual a, and a view name V, check whether $a^\mathcal{I} \in V^\mathcal{I}$ holds in every model $\mathcal{I}$ of Σ.

Schema validation supports the knowledge engineer by checking whether the skeleton of his domain model is consistent. Instance checking is the basic operation in querying a knowledge base. View subsumption helps in organizing and optimizing queries (see *e.g.* Buchheit *et al.* 1994). Note that the schema $\mathcal{S}$ has to be taken into account in all three services and that the view taxonomy $\mathcal{V}$ is relevant not only for view subsumption, but also for instance checking. In systems that forbid cycles, one can get rid of $\mathcal{S}$ and $\mathcal{V}$ by expanding definitions. This is not possible when $\mathcal{S}$ and $\mathcal{V}$ are cyclic.

2.4 Complexity Measures

The separation of the core of a TKRS into three components allows us to introduce refined complexity measures for analyzing the difficulty of inferences.

The complexity of a problem is generally measured with respect to the size of the whole input. However, with regard to our setting, three different pieces of input are given, namely the schema, the view taxonomy, and the world description. For this reason, different kinds of complexity measures may be defined, similarly to what has been suggested in [Vardi,1982] for queries over relational databases. We consider the following measures (where $|X|$ denotes the size of X):

Schema Complexity: the complexity as a function of $|\mathcal{S}|$;

View Complexity: the complexity as a function of $|\mathcal{V}|$;

World Description Complexity: the complexity as a function of $|\mathcal{W}|$;

Combined Complexity: the complexity as a function of $|\mathcal{S}| + |\mathcal{V}| + |\mathcal{W}|$.

Combined complexity takes into account the whole input. The other three instead consider only a part of the input, so they are meaningful only when it is reasonable to suppose that the size of the other parts is negligible. For instance, it is sensible to analyze the schema complexity of view subsumption because usually the schema is much bigger than the two views which are compared. Similarly, one might be interested in the world description complexity of instance checking whenever one can expect $\mathcal{W}$ to be much larger than the schema and the view part.

It is worth noticing that for every problem combined complexity, taking into account the whole input, is at least as high as the other three. For example, if the complexity of a problem is $O(|\mathcal{S}| \cdot |\mathcal{V}| \cdot |\mathcal{W}|)$, its combined complexity is cubic, whereas the other ones are linear. Similarly, if the complexity of a given problem is $O(|\mathcal{S}|^{|\mathcal{V}|})$, both its combined complexity and its view complexity are exponential, its schema complexity is polynomial, and its world description complexity is constant.

In this paper, we use combined complexity to compare the complexity of reasoning in our architecture with the traditional one. Moreover, we use schema complexity to show how the presence of a large schema affects the complexity of the reasoning services previously defined. View and world description complexity have been investigated (under different names) in [Nebel,1990; Baader,1990] and [Schaerf,1993; Donini *et al.*,1994], respectively.

3 The Case Studies

We studied some illustrative examples that show the advantages of the architecture we propose. We extended three systems by a simple cyclic schema language and analyzed their computational properties.

As argued before, a schema language should at least be expressive enough for declaring subconcept relationships, restricting the range of roles, and specifying roles to be necessary (at least one value) or single valued (at most one value). These requirements are met by the language $\mathcal{SL}$, which was introduced in [Buchheit *et al.*,1994] and that is defined by the following syntax rule:

$$C, D \longrightarrow A \mid \forall P.A \mid (\geq 1\, P) \mid (\leq 1\, P).$$

Obviously, it is impossible to express in $\mathcal{SL}$ that a concept is empty. Therefore, schema validation in $\mathcal{SL}$ is trivial. Also, subsumption of concept names is polynomially decidable.

We proved that inferences become harder for extensions of $\mathcal{SL}$. If we add inverse roles, schema validation remains trivial, but subsumption of schema names becomes NP-hard. If we add any construct by which one can express the empty concept—like disjointness axioms—schema validation becomes NP-hard. However, in our opinion this does not mean that extensions of $\mathcal{SL}$ are not feasible. For some extensions, there are natural restrictions on the form of schemas that decrease the complexity. Also, it is not clear whether realistic schemas will contain structures that require complex computations.

In all the three cases studied, the schema language is $\mathcal{SL}$. For the view language, we propose three different languages derived from three actual systems described in the literature, namely the deductive object-oriented database system ConceptBase [Jarke,1992], and the terminological systems KRIS [Baader and Hollunder,1991] and CLASSIC [Borgida *et al.*,1989]. We investigated the computational properties of the reasoning services with respect to $\mathcal{SL}$-schemas. We aimed at showing two results: (*i*) reasoning w.r.t. schema complexity is always tractable, (*ii*) combined complexity is not increased by the presence of terminological cycles in the schema.

In all three cases, we assume that view names are allowed in membership assertions and that the view taxonomy is acyclic. In this setting, every view name can be substituted with its definition. For this reason, from this point on, we suppose that view concepts are completely expanded. Therefore, when evaluating the complexity, we replace the size of the view part by the size of the concept representing the view.

We have found the following results for the three systems in which $\mathcal{SL}$ is the schema language and the concept language the abstraction of the query language of CONCEPTBASE introduced in [Buchheit *et al.*,1994], or the language offered by KRIS or CLASSIC, respectively.

CONCEPTBASE: instance checking is in PTIME w.r.t. combined complexity (view subsumption has been proved in PTIME in [Buchheit *et al.*,1994]).

KRIS: view subsumption and instance checking are PSPACE-complete problems w.r.t. combined complexity and PTIME problems w.r.t. schema complexity.

CLASSIC: view subsumption and instance checking are problems in PTIME w.r.t. combined complexity.

We conclude that adding (possibly cyclic) schema information does not change the complexity of reasoning within the systems taken into account.

4 Conclusion

We have proposed to replace the traditional TBox in a terminological system by two components: a schema, where primitive concepts describing frame-like structures are introduced, and a view part that contains defined concepts. We feel that this architecture reflects adequately the way terminological systems are used in most applications.

We also think that this distinction can clarify the discussion about the semantics of cycles. Given the different functionalities of the schema and view part, we propose that cycles in the schema are interpreted with descriptive semantics while for cycles in the view part a definitional semantics should be adopted.

In three case studies we have shown that the revised architecture yields a better tradeoff between expressivity and the complexity of reasoning.

The schema language we have introduced might be sufficient in many cases. Sometimes, however, one might want to impose more integrity constraints on primitive concepts than those which can be expressed in it. We see two solutions to this problem: either enrich the language and have to pay by a more costly reasoning process, or treat such constraints in a passive way by only verifying them for the objects in the knowledge base. The second alternative can be given a logical semantics in terms of epistemic operators (see Donini *et al.* 1992).

References

[Baader and Hollunder, 1991] Franz Baader and Bernhard Hollunder. A terminological knowledge representation system with complete inference algorithm. In *Proc. PDK-91*, LNAI, pages 67-86, 1991.

[Baader, 1990] Franz Baader. Terminological cycles in KL-ONE-based knowledge representation languages. In *Proc. AAAI-90*, pages 621–626, 1990.

[Borgida *et al.*, 1989] Alexander Borgida, Ronald J. Brachman, Deborah L. McGuinness, and Lori Alperin Resnick. CLASSIC: A structural data model for objects. In *Proc. ACM SIGMOD*, pages 59-67, 1989.

[Buchheit *et al.*, 1994] Martin Buchheit, Manfred A. Jeusfeld, Werner Nutt, and Martin Staudt. Subsumption between queries to object-oriented databases. *Information Systems*, 19(1):33–54, 1994.

[Donini *et al.*, 1992] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, Werner Nutt, and Andrea Schaerf. Adding epistemic operators to concept languages. In *Proc. KR-92*, pages 342–353, 1992.

[Donini *et al.*, 1994] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Andrea Schaerf. Deduction in concept languages: From subsumption to instance checking. *Journal of Logic and Computation*, 4(92–93):1–30, 1994.

[Jarke, 1992] M. Jarke. ConceptBase V3.1 User Manual. Aachener Informatik-Berichte 92-17, RWTH Aachen, 1992.

[Lenzerini and Schaerf, 1991] Maurizio Lenzerini and Andrea Schaerf. Concept languages as query languages. In *Proc. AAAI-91*, pages 471–476, 1991.

[Nebel, 1990] Bernhard Nebel. Terminological reasoning is inherently intractable. *Artificial Intelligence*, 43:235–249, 1990.

[Nebel, 1991] Bernhard Nebel. Terminological cycles: Semantics and computational properties. In John F. Sowa, editor, *Principles of Semantic Networks*, pages 331–361. Morgan Kaufmann, Los Altos, 1991.

[Schaerf, 1993] Andrea Schaerf. On the complexity of the instance checking problem in concept languages with existential quantification. *Journal of Intelligent Information Systems*, 2:265–278, 1993.

[Schild, 1994] Klaus Schild. Terminological cycles and the propositional μ-calculus. In *Proc. KR-94*, 1994.

[Vardi, 1982] M. Vardi. The complexity of relational query languages. In *Proc. STOC-82*, pages 137–146, 1982.

Flexible Inference Strategies for DL Systems*

J. Joachim Quantz and Guido Dunker
Technische Universität Berlin, Projekt KIT-VM11
FR 5-12, Franklinstr. 27/28, D-10587 Berlin
{jjq,dunker}@cs.tu-berlin.de

Véronique Royer
ONERA DES/SIA
avenue Division Leclerc
B.P. 72, F-92322 Châtillon Cedex
royer@onera.fr

1 Motivation

Based on the work presented in [6] and [7] we are currently developing the DL system FLEX, whose main characteristic is the support of flexible inference strategies. The main motivation for FLEX is based on experiences obtained by using the BACK system [2], which was developed for an Information System application, in the context of Natural Language Processing [5].

Though Description Logics are generally taken to be application-independent representation formalisms, the actual use of a DL system reveals specific requirements wrt *expressiveness*, *performance*, and *completeness*. One way of addressing these specific requirements is to take them explicitly into account when designing the DL system. The obvious disadvantage of this approach is that the resulting DL system might be adequate for the particular application it was designed for, but could be inadequate for a different application.

The BACK system, for example, offers an expressive term description language but is incomplete wrt some of the term-forming operators. When using the system for an NLP application we noticed that many inferences not needed in the Information System application were crucial for the NLP application.

We were thus facing two immediate problems:

1. It is not trivial to add the missing inferential power since it would involve the revision of basic assumptions underlying the implementation of BACK V5.
2. Even worse, integrating additional inferences would slow down the performance of BACK in the information system application, which would not benefit from the additional inferences.

Given this negative experience, we decided on a different approach to the implementation of DL systems which would allow *adaptability* of inference behavior according to the specific requirements of a given application.

There are three main characteristics underlying the development of FLEX:

1. FLEX offers an expressive term description language including negation, disjunction, role composition, role inversion, role value maps, and qualifying number restrictions.
2. For all term-forming operators FLEX provides a set of inference rules which is as complete as possible given the limits of decidability if termination is to be guaranteed.
3. The inference rules are explicitly represented and their application is declaratively controllable. Most inference rules can be switched off completely, or applied in forward or backward mode.[1]

In the following we will briefly sketch how flexible inference strategy is achieved in FLEX. In doing so, we will also point out problems and open questions—it should be noted that FLEX is not yet a full-fledged DL system, but still under development.

Note further that we assume that the adaption of FLEX towards a particular application is not performed by the users of FLEX but by the developers themselves.

2 Normalforms

In order to cope with disjunctions we use the format for normalforms presented in [3]. Thus a normalform 'nf(CON,DIS)' consists of a part 'CON' containing non-disjunctive information and a part 'DIS' containing disjunctive information, which is represented as a list of normalforms. Furthermore 'nf_ctop', 'nf_cbot', 'nf_rtop', and 'nf_rbot' are special normalforms with the obvious meanings. The non-disjunctive information is a list of atoms, where atoms are used to represent the concept and role-forming operators supported by FLEX, e.g. 'all(RNF,CNF)' or 'atleast(N,RNF,CNF)'. Note that atoms thus contain normalforms. Formally, we can define normalforms and atoms by a parallel inductive definition.

The following examples, in which the a_i stand for arbitrary atoms, illustrate this format:

$a_1 \wedge a_2$	nf([a_1, a_2],[ctop])
$a_1 \vee a_2$	nf([ctop],[nf([a_1],[ctop]),nf([a_2],[ctop]])
$a_1 \wedge (a_2 \vee a_3)$	nf([a_1],[nf([a_2],[ctop]),nf([a_3],[ctop]])

*The project KIT-VM11 is supported by the German Federal Minister of Research and Technology (BMFT) under contract 01 IV 101Q8.

[1]Note that such an explicit representation of inference rules might also be a good basis for an explanation component.

The main advantage of this representation format is that inconsistency between two terms can often be detected by unifying the respective non-disjunctive parts of their normalforms. (Checking consistency is one of the most important functionality of FLEX in the NLP application.) Furthermore, terms containing no disjunctive information are processed without any loss of performance.

The first normalization step in FLEX consists in the mapping of a DL term in external notation into this internal format. In doing so we "eliminate" all boolean operators:

1. Conjunction is represented by membership in the conjunctive part of a normal form.
2. Disjunction is represented by filling up the disjunctive part of normalforms.
3. Negation is moved to the inside. We chose our set of atoms in a way that it is closed under negation. Whereas this is trivial for most atoms, e.g. 'not(all(R,C))' gives 'some(R,not(C))' we had to introduce "non-standard" atoms in some cases, e.g. 'not(fillers(R,O))' give 'neg_fillers(R,O)'.

Another important issue involves the interdefinability of term-forming operators, such as 'some(R,C) = atleast(1,R and range(C))'. These equalities were used in BACK V5 to represent "new" operators by using "old" ones. In FLEX, however, we have explicit atoms for most term-forming operators. The main reason is that we want to allow application of certain inference rules to 'some' terms, for example, without being forced to apply them to 'atleast' terms in general.

The inference rules in FLEX are applied to the normalforms in two different ways: *normalization* rules map normalforms into normalforms, whereas *subsumption* rules test whether a normalform subsumes another. FLEX thus follows the normalize-compare paradigm underlying most existing DL systems.

3 Inference Rules

In [6] and [7] DL inference rules are derived systematically by rewriting Sequent Calculus proofs. The resulting inference rules, which have the general format

$$\gamma_1, \ldots, \gamma_n \;\rightharpoonup\; \gamma$$

are the basis for the inference rules implemented in FLEX. In order to use the sequent-style inference rules in FLEX, however, we have to transform them into normalization and subsumption rules.

3.1 Normalization Rules

The basic idea of *normalization rules* is to make information implicitly contained in a normalform explicit. Normalforms are thus expanded by applying normalization rules for logical completion and not only for getting some clean syntactic form, as is usually the cas case in general proof theory. In principle, normalization is achieved by taking atoms contained in a normalform as triggering conditions and then applying inference rules to include additional atoms into the normalform. To make normalization flexible and adaptable, normalization rules are explicitly represented and can be switched on and off.

It should be noted, however, that the sequent-style inference rules are not homogeneously treated in FLEX. We rather distinguish three different types of normalization rules:

1. Some inference rules are already used in the transformation of external terms into internal normalform format, e.g.
$$\rightharpoonup \quad \neg\forall r : c \doteq \exists r : \neg c$$
2. Most inference rules yield a subsumption formula where the subsuming term can be represented as an atom and the subsumed term as a list of atoms, e.g.
$$\begin{array}{l} c_1 \sqcap c_2 \sqsubseteq c_3, \\ r_2 \sqsubseteq r_1, r_2 \sqsubseteq r_3 \quad \rightharpoonup \quad \exists r_1 : c_1 \sqcap \forall r_2 : c_2 \sqsubseteq \exists r_3 : c_3 \end{array}$$
3. Some inference rules yield a subsumption formula containing disjunctions in the subsuming or subsumed term, e.g.
$$\rightharpoonup \quad \forall r.r^- : c \sqsubseteq c \sqcup \leq 0r : \top$$

Thus the first type of normalization rules is "hard-wired" into the parser and cannot be controlled from the outside. This is necessary to guarantee a format of normalforms which makes further processing more efficient (cf. the restriction to Negation Normal Forms in certain variants of the Sequent Calculus).

For the second type of normalization rules, a straightforward normalization strategy is used. The conjunctive part of a normalform, i.e. a list of atoms, is processed one by one, using the currently processed atom as trigger and then looking for the other atoms needed in the triggering conditions. Formally, these normalization rules have the general format

$$\alpha_1, \ldots, \alpha_n \;\rightharpoonup\; \alpha$$

i.e. if the atoms $\alpha_1, \ldots, \alpha_n$ are contained in the conjunctive part of a normalform, then α is added to this conjunctive part.

Simplifying the presentation, we use a predicate 'norm(TriggerAtom,CNF,AddAtom)' to realize this type of normalization rules, i.e. we have predicates like

```
norm(some(R1,C1),Con,some(R1,C1 and C2)) :-
    active_norm_rule(23),
    member(all(R2,C2),Con),
    subsumes(R2,R1).
```

Note that the information whether this rule is applied in the normalization phase is provided by 'active_norm_rule'. Thus a configuration specifying a particular inference strategy for FLEX consists of a list of declarations for active normalization or subsumption rules.

The third type of normalization rules is more difficult to realize, since it involves either checking of disjunctions in the triggering conditions or adding of complex information to a normalform.

An open problem in the normalization process concerns the minimization of rule applications. A naive implementation applies normalization rules until a fixed point is reached, i.e. a complete phase of normalization has been performed without changing the normalform.[2] We are currently investigating alternative techniques to minimize application of normalization rules, e.g. by storing whether a rule has already been applied to avoid redundant applications.

3.2 Subsumption Rules

Subsumption rules are used to determine subsumption between two normalforms. Obviously, the general strategy to prove subsumption between normalforms is to reduce it to subsumption between the conjunctive and the disjunctive parts, and then ultimately to subsumptions between atoms. Atomic subsumption rules are then implemented as follows:

```
subsumes(all(R1,C1),all(R2,C2)) :-
    active_sub_rule(17),
    subsumes(C2,C1),
    subsumes(R1,R2).
```

In [6, Sect. 5] it is shown that restriction to atomic subsumption rules is sound if normalization produces vivid normalforms. For arbitrary partitions of normalization and subsumption rules, however, subsumption rules become more complex. The main problem is that an atom might subsume a normalform without being a member of its conjunctive part. Take for example the normalization rule for 'all' and 'some' given above. If it is not active, we cannot rely on 'some(R1,C1 and C2)' being in the conjunctive part of a normalform when 'all(R,C1)' and 'some(R,C2)' are. To guarantee completeness for these cases we have to check for atoms in general, i.e. we have to chain possible normalization and subsumption rules backward. We are currently investigating different strategies to guide this backward chaining.

It seems reasonable to "collect" information along the role hierarchy and along role value maps, i.e. when the subsuming term contains an atom 'some(R1,C1)', we look for an atom 'some(R2,C2)' such that R2 is subsumed by R1 or by a role R3 which is "role-value mapped" to R4 which is in turn subsumed by R1. Doing subsumption checking this way guarantees completeness even if normalization rules regarding the role hierarchy and role value maps are not active.

In any case, to guarantee a minimal degree of efficiency, rules like

$$r \sqsubseteq c|^{\mathrm{Tr}} \;\rightharpoonup\; \exists r : \top \sqsubseteq c$$

should not be eligible for backward application, i.e. certain rules have to be applied in the normalization phase if completeness is desired.

[2]Of course, the normalization rules have to be written in such a way that termination is guaranteed. Note that this is not trivial if normalization rules which delete atoms are used.

An important trade-off concerning the choice of normalization and subsumption rules should be noted. Applying normalization rules guarantees that inconsistencies are immediately detected, but it also means that rules are applied without yielding any relevant result. On the other hand, applying subsumption rules guarantees that only relevant information is collected, but it means that inconsistencies might not be noticed immediately. Once we have implemented rules both in normalization and subsumption format, we are able to empirically evaluate different strategies for realistic applications.

Besides the flexibility concerning application of inference rules in the normalization or in the subsumption phase, FLEX also allows to switch off inference rules completely. This guarantees efficiency in the sense that only inferences needed in a particular application are actually performed, i.e. though being incomplete from a theoretical point of view, the system is still complete wrt the inferences needed in the particular application.

4 Reasoning about Objects

Reasoning about *objects* can involve complex case reasoning and is therefore harder than pure terminological reasoning. One way to cope with this problem is to consider alternative semantics for queries involving objects, e.g. [1; 8].

FLEX offers two mechanisms to cope with the complexity of object reasoning. First, application of propagation rules like

$$o_1{::}r : o_2, o_1{::}\forall r : c \;\rightharpoonup\; o_2{::}c$$

can be restricted to certain roles. Note that a similar strategy is already used in BACK V5 [2]. In addition to ordinary objects and concepts, BACK V5 supports attribute sets, numbers, and strings, which are treated as *values*, but not as full-fledged objects. Thus having a value restriction '0..20' for a number-valued role, instead of propagating this value restriction to the role filler, it is only checked whether the filler actually satisfies this constraint. The main difference between propagation and constraint-checking is that propagation can trigger additional operations and is thus non-local, whereas consistency checking is local.

The main source of inefficiency in object-level reasoning stems from this non-locality and we therefore investigate strategies for restricting the global effects of object-level inferences.

Furthermore, FLEX uses situated descriptions, independently motivated in [4] to perform case reasoning. Consider the following inference rule:

$$o{::}\forall r : \{o_1, \ldots, o_n\}^{\vee}, o{::}\exists r : c \;\rightharpoonup\; o_1{::}c \vee \ldots \vee o_n{::}c$$

To reason with disjunctive information, FLEX creates various non-disjunctive situations and a situation defined as the disjunction of these situations. A formula is valid in the disjunctive situation iff it is valid in all non-disjunctive situations it consists of.

Situated descriptions are also used to realize the integration of *weighted defaults* into FLEX. Roughly speaking, the possible "extensions" are represented in different situations, which are then scored and thus preferentially ordered [5].

5 Conclusion

The sequent-style inference rules derived in [7] provide an excellent basis for the development of DL systems supporting flexible inference strategies. The main problems we encountered so far involve the integration of disjunctive inference rules, the efficient resolution of subsumption queries while guaranteeing completeness, the normalform representation of role compositions, and the disjunctive nature of object-level reasoning in general, in particular wrt complex roles.

We are still experimenting with the FLEX implementation, which will be used in the project KIT-VM11. KIT-VM11 is part of the German Machine Translation project VERBMOBIL and is concerned with the disambiguation of Natural Language. The release of a first prototype of FLEX is envisaged for Summer '94. Though this first release will be mostly adapted towards the NLP application we also intend to use it to perform an empirical evaluation of different inference strategies for various existing applications. Such an evaluation should also reveal criteria for categorizing applications.

References

[1] F.M. Donini, M. Lenzerini, D. Nardi, A. Schaerf, W. Nutt, "Adding Epistemic Operators to Concept Languages", *KR-92*, 342–353

[2] T. Hoppe, C. Kindermann, J.J. Quantz, A. Schmiedel, M. Fischer, *BACK V5 Tutorial & Manual*, KIT Report 100, Technische Universität Berlin, 1993

[3] B. Kasper, "A Unification Method for Disjunctive Feature Descriptions", *ACL-87*

[4] J.J. Quantz, "An HPSG Parser Based on Description Logics", *to appear in COLING-94*

[5] J.J. Quantz, U. Küssner, B. Schmitz, "Using Description Logics for Disambiguation in Natural Language Processing", *International Workshop on Description Logics*, 1994

[6] V. Royer, J.J. Quantz, "Deriving Inference Rules for Terminological Logics", in , D. Pearce, G. Wagner (eds), *Logics in AI, Proceedings of JELIA'92*, Berlin: Springer, 84–105, 1992

[7] V. Royer, J.J. Quantz, *Deriving Inference Rules for Description Logics: a Rewriting Approach into Sequent Calculi*, KIT Report 111, Technische Universität Berlin, 1993

[8] V. Royer, J.J. Quantz, "On Intuitionistic Query Answering in Description Bases", *to appear in CADE-94*

Intuitionistic Query Answering in Description Bases

Véronique Royer, ONERA DES/SIA,
avenue Division Leclerc, B.P. 72, F-92322 Châtillon Cedex, royer@onera.fr

J. Joachim Quantz, Technical University of Berlin, KIT-VM11,
FR 5–12, Franklinstr. 28/29, D-10587 Berlin, jjq@cs.tu-berlin.de

1 Introduction

In this paper we present two calculi for Query Answering in Description Logics (DL), by expoiting some principles of Deductive Databases and Logic Programming. Given the standard model-theoretical semantics for DL, a complete Query Answering calculus has to perform complex case analyses to cope with implicit disjunctions stemming from some of the concept-forming operators in DL. To avoid this complexity and get linear strategies like in Logic Programming, we propose an intuitionistic approach to Query Answering based on the Sequent-Calculus-style axiomatization of DL we have developed in [11] and [12]. By taking into account only the intuitionistic inference schemata of this axiomatization, we obtain a strong intuitionistic Query Answering calculus. An additional restriction to reasoning about explicit objects allows a further simplification of the proof theory and yields a weak intuitionistic calculus. We prove completeness of these calculi wrt axiomatic semantics based on the Intuitionistic Sequent Calculus. For the weak calculus we also give a least fixed point semantics as known in Deductive Databases.

2 Proof-theoretical complexity of Query Answering

The distinction between TBox and ABox in Description Logics (DL) corresponds to a similar distinction in Deductive Databases (DDB) between the *intensional database*, rule-based knowledge usually in the form of Horn clauses, and the *extensional database*, which contains contingent knowledge modeled as ground facts. To make this analogy explicit we will in the following speak of DL knowledge bases as *description bases* (DB). We want here to consider this analogy, beyond syntax, with regards to computation techniques for answering queries about assertions. There are however two major sources of complexity making a significant difference with DDB.

First, the *procedural interpretation* of terminological knowledge in a DB, playing the role of *deductive rules* for assertions like in the intensional databases, makes sense only if the original TBox is closed under terminological entailment. This can be illustrated by considering a description base containing the terminological knowledge $c_1 \sqsubseteq c_2$ and the assertion $\forall r{:}c_1(o)$. If $c_1 \sqsubseteq c_2$ is supposed to behave as a deduction rule $c_1 \rightarrow c_2$, the deduction process being ordinary pattern matching, then the assertion $\forall r{:}c_2(o)$ cannot be derived (the assertion does not match the rule antecedent). To derive the assertion deductively, one should add the rule $\forall r{:}c_1 \rightarrow \forall r{:}c_2$ which corresponds to the formula $\forall r{:}c_1 \sqsubseteq \forall r{:}c_2$ logically entailed by $c_1 \sqsubseteq c_2$. Therefore, any rule processing in Query Answering must somehow take into account the *logical closure of the original TBox*. This presupposes some constructive characterization of it. Here we will rely on the finite Sequent style axiomatization proposed in [12] for a fragment of DL with conjunction, atomic negation, universal, existential and numerical restrictions.

friend(*john*,*susan*)	friend(*john*,*peter*)
¬ married(*mary*)	married(*susan*)
loves(*susan*,*peter*)	loves(*peter*,*mary*)

Figure 1: The "married friends" example.

A second source of complexity comes for assertional reasoning itself, as illustrated by the "married friends" example of Figure 1 taken from [3]. Model-theoretically, one can show that this ABox entails the fact

$\exists$friend:(married $\sqcap$ $\exists$loves:¬ married)(*john*).

However, the reasoning is not intuitionistic because it uses an Excluded Middle Law, namely that either *peter* is married or *peter* is not. Proof-theoretically, in Sequent Calculus [4], Excluded Middle corresponds to a more general form of reasoning with right[1] disjunctions and Right Contraction rules. It is well-known that Right Contraction is particularly expensive because its leads to arbitrary duplications[2] of the proof contexts, which means that no *linear control* of the proof process is possible.

[1] "Right" means intuitively that the property is about the consequent, as opposed to "left" which is about the antecedents, in any entailment relation $\Gamma \models \phi$.

[2] The Right Contraction rule means that proving some ϕ may be done by proving $\phi \vee \phi$, thus allowing different proof strategies for each copy of ϕ. This is exactly what happens when proving the "married friends" example in Sequent Calculus.

team $\doteq$
$\forall$member:human $\sqcap \leq 4$ member $\sqcap \geq 1$ leader
modern-team $\doteq$ team $\sqcap$ $\forall$leader:woman
woman :< human $\sqcap \neg$ man
man :< human
leader :< member

modern-team(*mt*)	≤ 3 member(*mt*)
member(*mt*,*dick*)	man(*dick*)
member(*mt*,*tom*)	man(*tom*)
member(*mt*,*mary*)	

Figure 2: The "modern team" example

Elimination of Right Contraction gives the Intuitionistic Sequent Calculus, which can be also considered as the proof-theoretical foundation of Logic Programming and Definite Deductive Databases [5]. The conclusion is that efficient Query Answering procedures as known from DDB or LP [1; 8], i.e. based on *linear* goal-subgoal strategies, cannot be performed in a DL whose underlying proof-theory allows for Right Contraction. Eventually, this motivates an *intuitionistic approach* to DL.

3 Towards Query Answering calculi: the Modern Team example

In this section we sketch the basic ideas underlying our Query Answering calculi by considering the "modern team" example in Figure 2 taken from [9]. It can be proved model-theoretically, but also intuitionistically, that *mary* is the leader of *mt* [3]. The Query Answering procedure will be formalized very much like in the **Alexander Method** [7] used in Deductive Databases[4]. It proceeds by introducing explicitly *query assertions*, c?(x) or r?(x,y), meaning that c or r are queried about, and *answer assertions*, c!(x) or r!(x,y), meaning that x or (x,y) has been obtained as a solution to the query c?(x) or r?(x,y)[5]. The intuition behind the Alexander Method is that any rule $c_1 \rightarrow c_2$ in a DDB provides with one query-rule $c_2?(x) \Rightarrow c_1?(x)$ and one answer-rule $c_2?(x) \wedge c_1!(x) \Rightarrow c_2!(x)$. This idea will be adapted to the context of DB.

[3] *tom*, *dick*, and *mary* are all members of team *mt* which has at most three members. Hence, due to the *Unique Name Assumption*, these are all members of *mt*. As there must be at least one leader for *mt*, necessarily a female one, this cannot be *tom* or *dick*. Note that this is a "moderate" form of Case Reasoning, which can be performed in Intuitionistic Sequent Calculus because it involves only reasoning with left disjunctions.

[4] This is also very close to the Magic Sets method [2] in DDB and Meta Logic Programming.

[5] Formally, c?(x) and c!(x) are assertions in an extended language. The arguments x and y are either variables or object names. The notations !/? not only stress the difference in interpretation between answer-rules and query-rules, but also the difference between specific Query-Answering calculi and ordinary hybrid entailment.

Let us show how it works for answering the query "who is the leader of *mt*?", written leader?(*mt*,x), in the Modern Team example. Note that this query is *open* and contains the variable x. As an answer we expect to retrieve all the objects o in $\mathcal{O}$ (the set of ABox object names) such that DB entails leader(*mt*,o) intuitionistically.

In order to stress the procedural meaning, the definitions in the TBox are first rewritten into *terminological clauses* looking like definite Horn clauses, the atoms being possibly role restriction terms[6]. Any terminological definition $c \doteq c_1 \sqcap \ldots \sqcap c_n$ gives:

1. one clause stating a sufficient condition for c, namely $c_1 \sqcap \ldots \sqcap c_n \rightarrow c$.
2. *n* clauses stating necessary conditions for c, namely $c \rightarrow c_i$, for i=1...n .

Processing the necessary conditions team $\rightarrow \geq 1$ leader and modern-team $\rightarrow$ team in a backward chaining manner, leads to new queries team?(*mt*) and modern-team?(*mt*). The computation of the queries and corresponding answers is formalized by specific *query-rules* and *answer-rules*[7], as shown in the following tables. We will write alternatively the query- or answer-rule (in brackets) and the derived query or answer.

{r?(x,y) $\Rightarrow \geq n$ leader?(x)}
≥ 1 r?(*mt*)
{TBox $\vdash c_1 \rightarrow c_2$, $c_2?(x) \Rightarrow c_1?(x)$, *Triggering*}
team?(*mt*)
{$c_2?(x)$, $c_1 \rightarrow c_2 \Rightarrow c_1?(x)$}
modern-team?(*mt*)
{c?(x) , c(o) $\Rightarrow$ c!(o), *Lookup in ABox*}
modern-team!(*mt*)
{TBox $\vdash c_1 \rightarrow c_2$, $c_1!(x) \Rightarrow c_2!(x)$, *Triggering*}
team!(*mt*) , ≥ 1 leader!(*mt*) ,
$\forall$leader:woman!(*mt*)

The third rule formalizes the basic answering process of just looking up in the ABox. Therefore it can only derive answers with respect to *known* objects, hence could not answer the open query leader?(*mt*,x). Answer-rules performing some kind of *Skolemisation* are necessary for that, therefore introducing *new variables names* to denote implicit objects.

{r?(o,x) , $\geq n$ r!(o) $\Rightarrow \wedge_{i=1}^{i=n}$ r!(o,s_i), for new s_i}
leader!(*mt*,s)

The query-answering process goes on collecting more evidence or *constraints* about the still undefined answer "s", in particular querying about cardinality upper bounds:

[6] There is no explicit disjunction.

[7] They are "hybrid" rules operating both on query-answer assertions, ABox assertions and TBox clauses.

{r?(x,y) $\Rightarrow$ $\leq n$ r?(x)}
$\leq n$ leader?(*mt*)
{$\leq n$ r$_1$?(*mt*) , r$_1 \rightarrow$ r$_2 \Rightarrow$ $\leq n$ r$_2$?(*mt*)}
$\leq n$ member?(*mt*)
{$\leq n$ r?(x) $\Rightarrow$ r?(x,y)}
member?(*mt*,y)
{*Lookup in ABox*}
member!(*mt*,s) , member!(*mt*,*tom*)
member!(*mt*,*dick*) , member!(*mt*,*mary*)
≤ 3 member!(*mt*)
{$\wedge_{i=1}^{i=n+1}$ r!(x,yi) , $\leq n$ r!(x) $\Rightarrow \vee_{i \neq j}$(yi = yj)
and Unique Name Assumption}
s=*tom* $\vee$ s= *mary* $\vee$ s=*dick*
{$\forall$r:c!(x), r!(x,y) $\Rightarrow$ c!(y)}
woman!(s)
{*Triggering*}
$\neg$man!(s)

Note that the second rule takes into account TBox entailment, namely the antimonotonicity of the **atmost** constructor. Now, one has obtained equality constraints which can be solved by exploiting the information about the male/female status of objects, according to the classical equality (substitution) axioms:

{$\neg$man!(x) $\Rightarrow$ man?(y) , if x$\notin\mathcal{O}$}
man?(y)
{*Lookup in ABox*}
man!(tom) , man!(dick)
{$\neg$c!(x) $\wedge$ c!(y) $\Rightarrow$ x $\neq$ y}
s $\neq$ tom , s $\neq$ dick
{$x \neq x_1$, $x = x_1 \vee \ldots \vee x = x_n$
$\Rightarrow x = x_2 \vee \ldots \vee x = x_n$}
s = mary
{y = z , r!(x,y) $\Rightarrow$ r!(x,z)}
leader!(*mt*,mary)

Finally, it should be noted that quite complex inferences have been performed, involving both the elimination of an existential quantification ($\geq$ 1 leader(*mt*)) and case reasoning wrt the derived disjunctive constraint: $x = tom \vee x = mary \vee x = dick$. As opposed to the "married friends" example this is still *intuitionistic reasoning* because only *left* disjunctions are necessary for the corresponding sequent proof [13]. Note also that when only man(*tom*) is known in the ABox, there is no way to determine the identity of "s" without ambiguity and therefore no determinate answer for leader?(*mt*,y). In this case, only equality constraints for leader!(*mt*,s) are computed.

4 Technical results

The following technical results are presented in more details in [13].

We define intuitionistic semantics for DL in two different ways. A first approach consists in exploiting the standard translation of DL into FOL and specifying an *axiomatic semantics based on derivability wrt the Intuitionistic Sequent Calculus* [5]. We actually present two different axiomatic semantics: a *strong intuitionistic semantics* based on the classical Intuitionistic Sequent Calculus and a *weak intuitionistic semantics* based on a fragment of Intuitionistic Sequent Calculus without elimination of right existentials. Thus the second semantics does not capture the reasoning about implicit objects. Following the techniques from DDB, we also specify a *least fixed point semantics based on intuitionistic inference schemata* for DL. In this third semantics, the intuitionistic inference schemata are obtained by restricting to the intuitionistic part of the complete set of inference rules proposed in our previous work about the axiomatization of DL [12].

Then, we present a *weak* and a *strong intuitionistic query-answering calculi*, by means of finite sets of query- and answer-rules as shown in the Modern Team example. These calculi work for DL languages having the usual boolean constructors, including disjunction and negation, as well as the universal, existential and numerical role restrictions. We assume no extensional constructor like **oneof** and no explicit equality or inequality assertion, except for the usual Unique Name Assumption between object names of the ABox. The weak and strong calculi are proved complete wrt to the weak and strong axiomatic semantics, respectively. The weak calculus is also shown to be complete wrt the least fixed point semantics.

The query-answering calculi represent basically *assertional components* for hybrid reasoning. The bridge betwen assertional and terminological reasoning is realized by *triggering rules*, which formalize the notion of triggering terminological clauses *entailed* from the original TBox. Hybrid reasoning is thus addressed by combining two calculi, relatively independent of each other: one assertional query-answering calculus and one terminological inference system.

The essential distinction between the weak and the strong calculus is that the strong calculus allows reasoning about objects implicitly defined by existential assertions like $\geq n$ rc. The strong calculus basically proceeds as shown in the Modern Team example. It has a *Skolemisation* answer-rule and therefore deals with answer-assertions not only about the original object names of the ABox but also about the new parameters introduced by Skolemisation. Some Equality reasoning is thus performed in the strong calculus in order to determine (as completely as possible) the identity of the skolem parameters.

The weak calculus is much simpler than the strong calculus because it lacks reasoning about implicit objects. It has no skolemisation answer-rule. Consequently, it deals only with answer-assertions about object names of the ABox and needs no Equality reasoning. It computes only "epistemic" answers in the sense of [3] and therefore it fails on the Modern Team example.

5 Conclusion

This contribution is a continuation of our previous works about the axiomatization of DL. In [11] we have presented a systematic method towards axiomatizing DL, namely by deriving inference schemata via rewriting Sequent Calculus proofs of terminological formulae. In [12] we have applied this method to the expressive terminological fragment underlying BACKV5 [6]. We did not address the axiomatization of assertional inferences, however, which is dealt now here.

Our approach shows how to *axiomatize assertional reasoning*, by means of explicit query- and answer-rules. The link with terminological reasoning is performed by *triggering rules* of the terminological clauses. This is in some analogy with the least-fixed point semantics of Deductive Databases.

On the other hand, the restriction to intuitionistic semantics relies on some analogy with Logic Programming. Indeed, the main motivation and interest for intuitionistic calculi was to obtain some *linear control* of the query derivation process as in Logic Programming. This cannot be obtained for classical semantics, even for languages without disjunction as shown by the "married friends" example.

The Query-Answering calculi presented here allow an efficient answering of open queries due to the *explicit control of the proof strategy* given by the existence of query- and answer-rules. This contrasts to the tableaux-based technique used for example in [3], which is mainly intended for answering closed queries and can only be inefficiently applied to open queries[8].

Since we have followed the standard strategy of Deductive Databases, an extension of our approach to recursive description bases seems possible by integrating methods similar to the ones developed in these fields (e.g. [2; 7]).

Let us finally conclude by some comments about complexity. The Strong Calculus is intrinsically complex because of Skolemisation and Equality reasoning. In both calculi, some unavoidable source of complexity lies in the need to perform terminological inferences, in order to trigger appropriate terminological clauses. On one hand, as intuitionistic semantics only partly solve the problems of disjunctions ("right-disjunctions" are eliminated), it is wise to restrict to *languages without disjunction* and with primitive negation, as is usually done in implemented systems like BACKV5. (Otherwise triggering rules more complex than the one used here are necessary: arbitrarily long disjunctive queries are derived and the search space grows exponentially.) On the other hand, performing terminological inferences pre-supposes some specific automated system. To increase efficiency, at the expense of completeness, one should take the approach of systems based on *flexible inference strategies*, where some inferences only are performed [10].

[8]The query c?(x) has to be mapped to queries c?(o_i) for all $o_i \in \mathcal{O}$.

References

[1] K. Apt, H. Blair, A. Walker, "Towards a Theory of Declarative Knowledge", in *Workshop on Foundations of Deductive Databases and Logic Programming*, Washington, 1986

[2] F. Bancilhon, D. Maier, Y. Sagic, J. Ullman, "Magic Sets and other Strange Ways of Implementing Logic Programs", in int. conf. *Principles of Database Systems (PODS-86)*, 1–15, 1986

[3] F.M. Donini, M. Lenzerini, D. Nardi, A. Schaerf, W. Nutt, "Adding Epistemic Operators to Concept Languages", in KR'92, 342–353, 1992.

[4] J. Gallier, *Logic for Computer Science; Foundations of Automatic Theorem Proving*, New York: Harper and Row, 1986

[5] J.Y. Girard, Y. Lafont, P. Taylor, *Proofs and Types*, Cambridge: Cambridge University Press, 1989

[6] T. Hoppe, C. Kindermann, J. J. Quantz, A. Schmiedel, M. Fischer, BACK V5 *Tutorial & Manual*, KIT Report 100, Technische Universität Berlin, 1993

[7] J.M. Kerisit, R. Lescoeur, J. Rohmer, "The Alexander Method: a Technique for the Processing of Recursive Axioms in Deductive Databases", *New Generation Computing* 3(4), 1986

[8] J.W. Lloyd, *Foundations of Logic Programming*, Berlin: Springer, 1987

[9] B.Nebel, *Reasoning and Revision in Hybrid Representation Systems*, Lecture Notes in Artificial Intelligence 422, Springer Verlag 1990

[10] J. J. Quantz, G. Dunker, V. Royer, "Flexible Inference Strategies for DL Systems", International Workshop on Description Logics, 1994

[11] V. Royer, J. J. Quantz, "Deriving Inference Rules for Terminological Logics", in , D. Pearce, G. Wagner (eds), *Logics in AI, Proceedings of JELIA'92*, Berlin: Springer, 84–105, 1992

[12] V. Royer, J. J. Quantz, "Deriving Inference Rules for Description Logics: a Rewriting Approach into Sequent Calculi", KIT Report 112, Technische Universität Berlin, 1993

[13] V. Royer, J. J. Quantz, "On Intuitionistic Query Answering in Description Bases", *12th conf. on Automated Deduction (CADE'94)*, Nancy 1994.

6 Appendix

We include in Figure 3 the query-answer rules of the strong calculus for languages without disjunction and with primitive negation (for concept assertions only).

The rules of the weak calculus are obtained by discarding the rules corresponding to *Skolemisation* and *Equality reasoning*.

Triggering of TBox clauses:

$\mathrm{TBox} \vdash c_1 \rightarrow c_2 \text{ , } c_2?(x) \Rightarrow c_1?(x)$

$\mathrm{TB} \vdash c_1 \rightarrow c_2 \text{ , } c_1!(u) \Rightarrow c_2!(u)$

Lookup in the ABox:

$c?(x) \text{ , } c(o) \text{ , } o \in \mathcal{O} \Rightarrow c!(o)$

$r?(x,y) \text{ , } r(o,o') \text{ , } o, o' \in \mathcal{O} \Rightarrow r!(o,o')$

Conjunction elimination:

$(c_1 \sqcap c_2)?(x) \Rightarrow c_1?(x) \wedge c_2?(x)$

$c_1!(u) \text{ , } c_1!(u) \Rightarrow (c_1 \sqcap c_2)!(u)$

Existential introduction:

$\geq nr{:}c?(x) \Rightarrow r?(x,y) \wedge c?(y)$

$\wedge_{i=1}^{i=n} r!(u,v_i) \wedge c!(v_i) \wedge \mathit{dif}(v_1, \ldots, v_n) \Rightarrow \geq nr{:}c!(u)$

Propagation:

$c?(x) \Rightarrow \forall r{:}c?(y) \wedge r?(y,x)$

$\forall r{:}c!(u) \text{ , } r!(u,v) \Rightarrow c!(v)$

Skolemization: introduction of new parameters v_i

$c?(y) \Rightarrow \geq nr{:}c?(x)$

$r?(x,y) \Rightarrow \geq nr{:}c?(x)$

$\geq nr{:}c!(u) \Rightarrow \wedge_{i=1}^{i=n} (r!(u,v_i) \wedge c!(v_i)) \wedge \mathit{dif}(v_1, \ldots, v_n)$

Negation reasoning:

$\neg r?(x,y) \Rightarrow \forall r{:}c?(x) \wedge \neg c?(y)$

$\neg r?(x,y) \Rightarrow \leq nr{:}c?(y) \wedge c?(y)$

$\neg c?(x) \Rightarrow \leq nr{:}c?(y) \wedge r?(y,x)$

$\forall r{:}c!(u) \text{ , } \neg c!(v) \Rightarrow \neg r!(u,v)$

$\leq nr{:}c!(u) \text{ , } c!(v_i) \text{ , } r!(u,v_i) \text{ , } 1 \leq i \leq n \text{ , } r!(u,v_{n+1}) \text{ , } \mathit{dif}(v_1, \ldots, v_{n+1}) \Rightarrow \neg c!(v_{n+1})$

$\leq nr{:}c!(u) \text{ , } c!(v_i) \text{ , } r!(u,v_i) \text{ , } 1 \leq i \leq n \text{ , } c!(v_{n+1}) \text{ , } \mathit{dif}(v_1, \ldots, v_{n+1}) \Rightarrow \neg r!(u,v_{n+1})$

Equality reasoning:

Unique Name Axiom and the Identity Axioms

$c!(b) \text{ , } b \notin \mathcal{O} \Rightarrow \neg c?(x)$

$b \notin \mathcal{O} \text{ , } c!(b) \text{ , } r!(u,b) \Rightarrow \leq nr{:}c?(u)$

$c!(u) \text{ , } u = v \Rightarrow c!(v)$

$c!(v) \text{ , } \neg c!(u) \Rightarrow u \neq v$

$\leq nr{:}c!(u) \text{ , } c!(v_i) \text{ , } r!(u,v_i) \text{ , } 1 \leq i \leq n+1 \Rightarrow \mathit{equ}(v_1, \ldots, v_{n+1})$

$\mathit{dif}(v_1, \ldots, v_n) \overset{def}{=} \neg \mathit{equ}(v_1, \ldots, v_n) \overset{def}{=} \wedge_{i \neq j}(v_i \neq v_j)$

Figure 3: Strong query-answering calculus (rules for concept assertions)

Homogeneous Concepts in a Temporal Description Logic

Alessandro Artale[*] and **Enrico Franconi**
Knowledge Representation and Reasoning Lab.
IRST, 38050 Povo TN, Italy
{artale, franconi}@irst.it

Claudio Bettini
University of Milano
via Comelico 39, 20135 Milano, Italy
bettini@imiucca.csi.unimi.it

1 Introduction

In the temporal literature *homogeneity* is a property of predicates in relation to time. Homogeneity characterizes the temporal behavior of world states: when a state holds over an interval of time t, it also holds over subintervals of t. Thus, if a parking is free on Sunday, one can conclude that it is also free on Sunday morning.

On the other hand, actions are not necessarily homogeneous. In the linguistic literature a difference is made between *activity* and *performance* verbs, the distinction comes out in the fact that performance verbs do not, whereas activity verbs do, have subevents that are denoted by the same verbs. Generally, activity verbs represent ongoing events, for example *to eat* and *to run*, and can be described as homogeneous predicates; whereas performance verbs represent events with a well defined granularity in time, like *to prepare spaghetti*. Performance verbs are example of anti-homogeneous events: if they occur over an interval of time t, then they do not occur over a subinterval of t, as they would not yet be completed.

2 The Temporal Language

We describe a simple description logic to represent classes of individuals and their temporal relations – related work can be found in [Schmiedel,1990; Weida and Litman,1992; Bettini,1992]. This extends with the homogeneity operator the language presented in [Artale and Franconi,1994a; Artale and Franconi,1994b; Artale and Franconi,1993; Artale,1994]. For sake of simplicity, the language presented here does not include complex concept forming operators, like disjunction and temporal substitutive qualifier.

Basic types of the language are *concepts*, *individuals*, *temporal variables* and *intervals*. A concept is a description gathering the common properties among a collection of individuals. Concepts can describe entities of the world, states, events and processes. Temporal variables denote intervals bound by temporal constraints, by means of which abstract temporal patterns in the form of constraint networks are expressed. Concepts (resp. individuals) can be specified to hold at a certain interval variable (resp. value) defined by the constraint network. In this way, *action types* (resp. *individual actions*) can be represented in a uniform way by temporally related concepts (resp. individuals).

C, D	$\rightarrow$	$A \mid$	(atomic concept)
		$\top \mid$	(top)
		$C \sqcap D \mid$	(conjunction)
		$p \downarrow q \mid$	(agreement)
		$p : C \mid$	(selection)
		$C@X \mid$	(temporal qualifier)
		$\diamond(X^{+})\ \mathcal{T}c^{+}.\ C \mid$	(temporal existential quantifier)
		∇C	(homogeneous concept)
p, q	$\rightarrow$	$f \mid$	(atomic feature)
		$\star g \mid$	(atomic parametric feature)
		$p \circ q$	(feature chain)
$\mathcal{T}c$	$\rightarrow$	$(X\ (R)\ Y)$	(temporal constraint)
R, S	$\rightarrow$	$R\ ,\ S \mid$	(disjunction of relations)
		$\mathsf{s} \mid \mathsf{mi} \mid \mathsf{f} \mid \ldots$	(Allen basic temporal relations)
X, Y	$\rightarrow$	$\natural \mid \mathsf{x} \mid \mathsf{y} \mid \ldots$	(temporal variables)

Figure 1: Syntax rules for the temporal concept language

Concept expressions (denoted by C, D) are syntactically built out of *atomic concepts* (denoted by A), *atomic features* (denoted by f), *atomic parametric features* (denoted by $\star g$) and constrained *interval variables* (denoted by X, Y) according to the abstract syntax rules of figure 1. For the basic interval relations we use the same notation as in [Allen,1991]: before (b), meets (m), during (d), overlaps (o), starts (s), finishes (f), equal (=), after (a), met-by (mi), contains (di), overlapped-by (oi), started-by (si), finished-by (fi). Temporal variables are introduced by the temporal existential quantifier "$\diamond$". Variables appearing in temporal constraints should be declared within the same temporal quantifier, with the exception of the special variable $\natural$.

Concept expressions are interpreted in our logic over pairs of *temporal intervals* and *individuals* $\langle i, a \rangle$, meaning that the individual a is in the extension of the concept at the interval i. If a concept is intended to denote an action, then its interpretation can be seen as the set

[*]Current address: Ladseb-CNR, I-35020 Padova PD, Italy

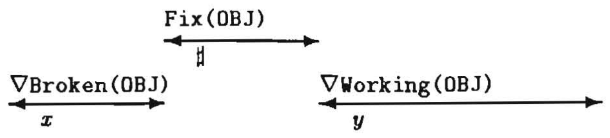

Figure 2: Temporal dependencies in the definition of the Fix action.

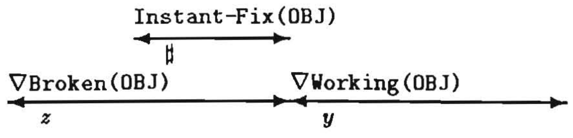

Figure 3: Temporal dependencies in the definition of the Instant-Fix action.

of individual actions of that type occurring at some interval. Homogeneous concepts "∇C" are also temporal concepts. Their semantics will show that they are obtained by a special case of universal temporal quantifier "$\Box$" [Bettini,1993a]. For a formal definition of the semantics of the homogeneity-free language see [Artale and Franconi,1994a].

Within a concept, the special "$\sharp$" variable denotes the generic interval at which the concept itself *holds*; in the case of actions, it refers to the temporal interval at which the action itself *occurs*. A concept holds at an interval X if it is temporally qualified at X – written $C@X$; in this way, every occurrence of $\sharp$ embedded within the concept expression C is interpreted as the X variable. Since any concept is implicitly temporally qualified at the special $\sharp$ variable – $C \equiv C@\sharp$ – it is not necessary to explicitly qualify concepts at $\sharp$. The temporal existential quantifier introduces interval variables, related each other and possibly to the $\sharp$ variable in a way defined by the set of *temporal constraints*.

The semantics of homogeneous concepts is easily given in terms of the semantics of the universal temporal quantifier, the counterpart of the existential temporal quantifier operator available in this language. In fact, the following equivalence holds:

$$\nabla C \equiv \Box x\ (x\ (=,\mathsf{s},\mathsf{d},\mathsf{f})\ \sharp).\ C@x$$

meaning that, C is an homogeneous concept (∇C) if and only if when it holds at an interval it remains true at each subinterval. In particular, $\Box x$ universally qualifies the temporal variable x, while the temporal constraint $(x\ (=,\mathsf{s},\mathsf{d},\mathsf{f})\ \sharp)$ has to be read as the disjunction $(x = \sharp) \vee (x\ \mathsf{s}\ \sharp) \vee (x\ \mathsf{d}\ \sharp) \vee (x\ \mathsf{f}\ \sharp)$, imposing that x is a generic interval contained in $\sharp$. Moreover, it is always true that $\nabla C \sqsubseteq C$, i.e. ∇C is a more specific concept than C.

Let us consider as an example the definition of the Fix action:

$$\begin{array}{l}\texttt{Fix} \doteq \\ \quad \Diamond(x\ y)\ (x\ \mathsf{m}\ \sharp)(\sharp\ \mathsf{m}\ y). \\ \quad (\star\texttt{OBJECT} : (\nabla\texttt{Broken}@x \sqcap \nabla\texttt{Working}@y))\end{array}$$

Fix denotes, according to the definition, any action occurring at some interval involving an ⋆OBJECT that was once Broken and then Working. The $\sharp$ interval could be understood as the occurring time of the action type being defined: referring to it within the definition is an explicit way to temporally relate states and actions occurring in the world with respect to the occurrence of the action itself. The variables x, y are existentially introduced by the operator $\Diamond$. The temporal constraints $(x\ \mathsf{m}\ \sharp)$ and $(\sharp\ \mathsf{m}\ y)$ state that the interval denoted by x should meet the interval denoted by $\sharp$ – the occurrence interval of the action type Fix – and that $\sharp$ should meet y. Figure 2 shows the temporal dependencies of the intervals in which the concept Fix holds. The parametric feature ⋆OBJECT plays the role of *formal* parameter of the action, mapping any individual action of type Fix to the object to be Fixed, independently from time. Please note that, whereas the existence and identity of the ⋆OBJECT of the action is time invariant, it can be qualified differently in different intervals of time, e.g., the ⋆OBJECT is necessarily Broken only during the interval denoted by x and its subintervals.

The assertion $\texttt{Fix}(i, a)$ says that a is an individual action of type Fix occurred at the interval i. Moreover, the same assertion implies that a is related to a ⋆OBJECT, say b, which is of type Broken at some interval j – meeting i – and at all intervals included in j, and of type Working at another interval l – met by i – and at all intervals included in l:

$$\begin{array}{l}\texttt{Fix}(i,a) \Longrightarrow \\ \quad \exists b.\ \star\texttt{OBJECT}(a,b)\ \wedge \\ \qquad \exists\ j,l.\ \mathsf{m}(j,i) \wedge \mathsf{m}(i,l)\ \wedge \\ \qquad\qquad \forall\ \hat{j},\hat{l}.\ (=,\mathsf{s},\mathsf{d},\mathsf{f})(\hat{j},j) \wedge (=,\mathsf{s},\mathsf{d},\mathsf{f})(\hat{l},l) \rightarrow \\ \qquad\qquad\qquad \texttt{Broken}(\hat{j},b) \wedge \texttt{Working}(\hat{l},b).\end{array}$$

It is interesting to notice that the Fix action subsumes the following Instant-Fix action, whose temporal dependencies are depicted in figure 3:

$$\begin{array}{l}\texttt{Instant-Fix} \doteq \\ \quad \Diamond(z\ y)\ (\sharp\ \mathsf{f}\ z)(\sharp\ \mathsf{m}\ y). \\ \quad (\star\texttt{OBJECT} : (\nabla\texttt{Broken}@z \sqcap \nabla\texttt{Working}@y))\end{array}$$

Subsumption holds because the class of intervals – obtained by homogeneity of the state Broken in the definition of Fix – including x and all its subintervals, is a subset of the class of intervals over which the object is known to be broken, according to the definition of Instant-Fix; this class includes all the subintervals of z.

Please note that, if the Instant-Fix action had been defined without the ∇ operator, then it would not specialize any more the Fix action. In fact, according to the weaker definition of Instant-Fix, specifying that the object is broken at z does not imply that the object is broken at subintervals of z; in particular, we can not deduce any more that the object is broken at x and its subintervals, as specified in the definition of Fix action. For the same reason, the *weak* Instant-Fix action type would not specialize the *weak* Fix action type, too. Thus, homogeneity helps us to define states and actions in a more accurate way, such that important inferences are captured.

3 Calculus for Subsumption

In [Artale and Franconi,1994a] it is shown that subsumption between temporal concepts in this logic, without homogeneous concepts, is decidable. We remind that subsumption in a propositionally complete description logic with both existential and universal temporal quantification is undecidable and it is still not clear if it becomes decidable in absence of negation [Bettini,1993a]. The homogeneity operator is a restricted form of universal quantification and we are particularly interested in an even more restricted form, where the concept C in ∇C does not contain any other temporal operator (we call it *simple homogeneous concept*). The expressiveness of the resulting logic is enough, for example, to correctly represent the homogeneous nature of states. We have developed an algorithm to compute subsumption in this logic (see also [Bettini,1993c]). Even if a formal proof is still not available, there are good arguments to conjecture its completeness with respect to the usual definition of subsumption (see [Artale and Franconi,1994a] for its temporal formulation). This would also prove decidability of this logic and of corresponding modal logics.

We first compute a *normal form* for concepts: each temporal concept is normalized and transformed into an equivalent minimal temporal constraint network of the form:

$$\diamond(\overline{X})\ \overline{\mathcal{T}c}.\ (Q^1@X^1 \sqcap \ldots \sqcap Q^n@X^n)$$

where Q^i are conjunctions of non-temporal and simple homogeneous concepts ($Q^i = L^i \sqcap \nabla M^i$, with L^i, M^i non-temporal concepts), while arcs are labeled with temporal relations. Each normalized conceptual temporal constraint network is computed in such a way that it has some interesting properties: temporal constraints are always explicit, computing the transitive closure of the Allen temporal relations obtaining the minimal temporal network (see e.g. [van Beek and Cohen,1990]); the information in each node is independent from the information in the other nodes; the network is minimal wrt time-invariant information, maintained only in the $\sharp$ node of the graph.

In absence of the homogeneity operator, concept subsumption in the temporal language is reduced to concept subsumption between non-temporal concepts and to subsumption between temporal constraint networks. A similar general procedure was first presented in [Weida and Litman,1992], where the language is less expressive. Algorithms to compute subsumption between non-temporal concepts are well known, see e.g. [Hollunder *et al.*,1990]. An exhaustive search is done to find a mapping between each node in the potential subsumer and some node in the subsumee, such that subsumption between concepts labeling corresponding nodes holds, while temporal relations labeling arcs between corresponding nodes have to be less specific in the potential subsumer (with $R_1 \subseteq R_2$, we indicate that R_1 is a more restricted set of temporal constraints than R_2).

Definition (S-mapping)
A *s-mapping* from a conceptual temporal constraint network $\langle \overline{X}, \overline{\mathcal{T}c}_1, \overline{Q@X}\rangle$ to a conceptual temporal constraint network $\langle \overline{Y}, \overline{\mathcal{T}c}_2, \overline{P@Y}\rangle$ is a total function $\mathcal{S} : \overline{X} \mapsto \overline{Y}$ such that $\mathcal{S}(\sharp) = \sharp$, and the concept labeling each node x in $\overline{X}$ subsumes the concept labeling the corresponding node $\mathcal{S}(x)$, and for each temporal constraint $(X_1\ R_1\ X_2) \in \overline{\mathcal{T}c}_1$ there exists a temporal constraint $(Y_1\ R_2\ Y_2) \in \overline{\mathcal{T}c}_2$ which satisfies $\mathcal{S}(X_1) = Y_1$, $\mathcal{S}(X_2) = Y_2$ and $R_1 \supseteq R_2$. □

The following proposition introduces a procedure for checking subsumption between conceptual temporal constraint networks.

Proposition 1 *A conceptual temporal constraint network $C = \langle \overline{X}, \overline{\mathcal{T}c}_1, \overline{Q@X}\rangle$ subsumes a conceptual temporal constraint network $D = \langle \overline{Y}, \overline{\mathcal{T}c}_2, \overline{P@Y}\rangle$, written $C \sqsupseteq D$, if and only if there exists a s-mapping from C to D.*

Proof. See [Artale and Franconi,1994a]. □

In order to compute subsumption in presence of homogeneity, we must first define a simpler form of temporal subsumption, to test when a concept associated to a node subsumes a concept associated to another node. Since each Q^i, in the normal form, can be expressed by the conjunction $L^i \sqcap \nabla M^i$, we extend the non-temporal subsumption algorithm to test whether $(L \sqcap \nabla M) \sqsupseteq (N \sqcap \nabla O)$.

Proposition 2 *Let L, M, N, O be non-temporal concepts:*
$(L \sqcap \nabla M) \sqsupseteq (N \sqcap \nabla O)$ iff $L \sqsupseteq (N \sqcap O)$ and $M \sqsupseteq O$.

Proof. See [Bettini,1993c]. □

Proposition 2 shows how this simple form of temporal subsumption can be reduced to non-temporal subsumption, for which algorithms are known.

In order to take into account homogeneity, new variables in the subsumee D – representing intervals of time *contained* into intervals explicitly present in D – should be considered. These are actually new nodes in the conceptual temporal constraint network, and their associated concepts are inherited from the nodes that temporally contain them. Since we have a dense temporal domain we can have an infinite number of these nodes. However, many of them will have the same qualitative temporal relations with the intervals present in D and identical associated concepts. As a matter of fact, there is only a finite number of significative groups of these intervals. We can consider one new node for each group. Each new node must be an image of a node in the subsumer, C, for which does not exist an exact mapping. Its temporal constraints with other nodes in D, that are images of nodes in C, must be tighter or equal with respect to the corresponding constraints in C. Moreover, the node will be useful to detect subsumption only if the interval associated to it is contained in one or more intervals associated to nodes in D. In fact, only from these nodes it could inherit some homogeneous concept.

The algorithm, illustrated in figure 4, implements a procedure that find a *S-mapping*, modified as discussed

```
HOM-SUBS($C$,$D$)
/*
 * $C = \langle \overline{X}, \overline{Tc_1}, \overline{Q@X} \rangle$
 * where $\overline{Q@X} = Q^0@\natural \sqcap Q^1@x_1 \sqcap \ldots \sqcap Q^n@x_n$
 * $D = \langle \overline{Y}, \overline{Tc_2}, \overline{P@Y} \rangle$
 * where $\overline{P@Y} = P^0@\natural \sqcap P^1@y_1 \sqcap \ldots \sqcap P^m@y_m$
 */
  • IF $Q^0 \sqsupseteq P^0$
        THEN subs ← FIND-MAP(($\natural \rightarrow \natural$),$D$,$x_1$)
    ELSE subs ← False
RETURN(subs).

FIND-MAP(Map,Net2,$x_i$)
  • success ← False ; $j \leftarrow 1$
  • WHILE ($y_j$ in $\overline{Y}$) AND NOT(success) DO
      – Newnet2 ← TEMP-SUBS(Map,Net2,$x_i$,$y_j$)
      – IF Newnet2 AND ($Q^i \sqsupseteq P^j$) THEN
          * NewMap ← Map ∪ $\{x_i \rightarrow y_j\}$
          * IF $i = n$ THEN success ← NewMap
            ELSE success ←
                FIND-MAP(NewMap,Newnet2,$x_{i+1}$)
      – $j \leftarrow j + 1$
  • IF NOT(success) THEN
    /* There is no direct mapping with variables in $D$
     * Try to exploit homogeneity properties */
    success ← TRY-HOM(Map,Net2,$x_i$)
RETURN(success).

TEMP-SUBS(Map,Net2,$x_i$,$y_j$)
  • Temp-net ← Net2
  • WHILE (($xRx_i$) ∈ $C.\overline{Tc_1}$) AND (Temp-net) DO
        IF ($x \rightarrow y$) ∈ Map THEN
            IF ($y \in D.\overline{Y}$ AND ($yR'y_j$) ∈ $D.\overline{Tc_2}$
                AND NOT ($R \supseteq R'$))
                /* y is one of the original nodes in $D$*/
            THEN Temp-net ← 0 /* infeasible mapping */
            ELSE IF ($y \notin D.\overline{Y}$ AND ($yR'y_j$) ∈ Net2.$\overline{TC}$)
                THEN /* y is one of the added nodes */
                    – $R' \leftarrow R \cap R'$
                    – IF ($R' = \emptyset$) THEN Temp-net ← 0
                      ELSE Temp-net ← Propagate(Net2)
RETURN(Temp-net).
```

Figure 4: An algorithm to compute subsumption in the temporal language extended with the homogeneity operator.

above in order to take into account homogeneous concepts. The first time, it tries to find a mapping without introducing new nodes in the subsumee – it can be used for computing subsumption in the absence of homogeneity, too. The function FIND-MAP, after trying a *direct* mapping without success, calls TRY-HOM to check if the subsumption can be proved considering homogeneous properties of concepts. TRY-HOM is illustrated in figure 5; it considers the fact that each node in the subsumee, labeled with a homogeneous concept, can be expanded generating new nodes. While, in principle, their number is infinite, the number of *useful* ones is finite and a mapping is found if and only if the subsumption holds. The TRY-HOM function succeeds if:

- the added node does not modify the temporal constraints among the original variables of D – as computed in HOM-TEST;
- the new node inherits homogeneous concepts from other nodes, in such a way that the concept associated to the unmatched node of the subsumer network could subsume the concept associated with the new node.

The function HOM-TEST, used by TRY-HOM, takes as arguments, respectively, a subset of nodes, a network and the new node. It restricts each relation between the new node and nodes in the subset to include only relations in {s,d,f}. For example, {o,s,d} is restricted to {s,d}. This restriction is necessary to inherit homogeneous concept: the node must be necessarily *internal*. Finally the constraints are propagated. The test succeeds if the restricted relations are *compatible* with the original network, and the relations involving nodes in the original network are not restricted – otherwise we would be checking subsumption of a different network, not necessarily subsumed by the original one.

Particular attention must be paid to the temporal relations that we enforce on the nodes that we add to D. This fact is reflected in the TEMP-SUBS function. This function is called when a mapping between a node of the subsumer and an existing node of the subsumee is tried. This node can be an original node of D or an internal node added in a previous step of the algorithm. TEMP-SUBS returns a network. If the node is an original node of D, it must return a nonzero value iff temporal constraints in C subsume the corresponding constraints in D, under the partial mapping, *Map*, augmented with the new mapping – ($x_i \rightarrow y_j$). If the node is an added node, we just check that the *intersection* of corresponding temporal constraints is not empty. If this intersection restricts the constraints in the subsumee network, this restriction must be considered in further steps of the algorithm.

References

[Allen, 1991] Allen, James F. 1991. Temporal reasoning and planning. In Allen, James F.; Kautz, Henry A.; Pelavin, Richard N.; and Tenenberg, Josh D., editors 1991, *Reasoning about Plans*. Morgan Kaufmann. chapter 1, 2–68.

```
TRY-HOM(Map,Net2,$x_i$)
  • success ← False
  • Newnet2 ← ADDNODE(Net2,$\overline{y}$,$\overline{x}_i$,Map)
  • Newnet2 ← Propagate(Newnet2)
  • IF Newnet2 THEN
      – Let $Y = \{y \in D.\overline{Y} : \overline{y}Ry \in$ Newnet2.$\overline{TC}$
                  AND (s,d,f)$\cap R \neq \emptyset)\}$
      – IF $Y \neq \emptyset$ THEN FOR each subset $SY$ of $Y$ DO
        * Let $P@\overline{y} \in$ Newnet2.$\overline{P@Y}$
        * IF $Q^i \sqsupseteq P$ THEN
          · Tempnet ← HOM-TEST($SY$,Newnet2,$\overline{y}$)
          · IF Tempnet THEN
                + NewMap ← Map $\cup \{x_i \rightarrow \overline{y}\}$
                + Newnet2 ← Tempnet
                + IF $i = n$ THEN success ← NewMap
                  ELSE success ←
                    FIND-MAP(NewMap,Newnet2,$x_{i+1}$)
RETURN(success).

ADDNODE(Net,$\overline{y}$,$\overline{x}_i$,Map)
  • Newnet.$\overline{Y}$ ← Net.$\overline{Y} \cup \{\overline{y}\}$
  • Newnet.$\overline{TC}$ ← Net.$\overline{TC} \cup \{(yR\overline{y}) : \exists x_k\ (x_k \rightarrow y) \in$ Map
          AND $(x_k R x_i) \in C.\overline{\mathcal{T}c}_1\}$
  • Newnet.$\overline{P@Y}$ ← Net.$\overline{P@Y} \cup \{\top@\overline{y}\}$
RETURN(Newnet).

HOM-TEST($Z$,Net,$\overline{y}$)
  • FOR each $\overline{y}Rz_i \in$ Net.$\overline{TC}$ with $z_i \in Z$ DO
       $R \leftarrow R \cap \{s,d,f\}$
  • success ← Propagate(Net)
  • IF success THEN /**The network is consistent**/
       FOR each $y_i R' y_j \in D.\overline{\mathcal{T}c}_2$ with $y_i, y_j \in D.\overline{Y}$ DO
          IF $R \subset R'$ where $y_i R y_j \in$ Net.$\overline{TC}$ THEN
             success ← Fail
RETURN(success).
```

Figure 5: A sketch of the TRY-HOM algorithm.

[Artale and Franconi, 1993] Artale, Alessandro and Franconi, Enrico 1993. A unified framework for representing time, actions and plans. In Anger, F. D.; Guesgen, H. W.; and van Benthem, J., editors 1993, *Workshop Notes of the IJCAI Workshop on Temporal and Spatial Reasoning*, Chambery, France. 193–217. Also in the Workshop Notes of the IJCAI-93 Workshop on Object-Based Representation System; a shorter version appears in the Workshop Notes of the Italian 1993 Workshop on Automatic Planning, Roma Italy, September 1993.

[Artale and Franconi, 1994a] Artale, Alessandro and Franconi, Enrico 1994a. A computational account for a description logic of time and action. In *Proc. of the 4th International Conference on Principles of Knowledge Representation and Reasoning*, Bonn, Germany.

[Artale and Franconi, 1994b] Artale, Alessandro and Franconi, Enrico 1994b. Time, actions and plans representation in a description logic. *International Journal of Intelligent Systems.* To appear.

[Artale, 1994] Artale, Alessandro 1994. *Rappresentazione di Tempo ed Azioni e Ragionamento Tassonomico per Object-Oriented Databases nel Contesto dei Linguaggi Terminologici.* Ph.D. Dissertation, University of Florence, Italy. (in Italian).

[Bettini, 1992] Bettini, Claudio 1992. A formalization of interval-based temporal subsumption in first order logic. In *Workshop Notes of the ECAI Workshop on Theoretical Foundations of Knowledge Representation and Reasoning*, Vienna, Austria.

[Bettini, 1993a] Bettini, Claudio 1993a. *Temporal Extensions of Terminological Languages.* Ph.D. Dissertation, Computer Science Department, University of Milano, Italy. (in Italian).

[Bettini, 1993b] Bettini, Claudio 1993b. A family of temporal terminological logics. In *Proc. of the 3rd Congress of the Italian Association for Artificial Intelligence, Torino, Italy*, Lecture Notes in Artificial Intelligence, N.728, pages 120–131. Springer Verlag, 1993.

[Bettini, 1993c] Bettini, Claudio 1993c. Computing subsumption between action descriptions. Technical Report 102-93, University of Milano, Computer Science Dept.,Italy.

[Hollunder *et al.*, 1990] Hollunder, B. and Nutt, W. and Schmidt-Schauß, M. 1990. Subsumption algorithms for concept description languages. In *Proc. of ECAI-90*, Stockholm, Sweden. 348-353.

[Schmiedel, 1990] Schmiedel, A. 1990. A temporal terminological logic. In *Proc. of AAAI-90*, Boston, MA. 640-645.

[van Beek and Cohen, 1990] van Beek, P. and Cohen, R. 1990. Exact and Approximate Reasoning about Temporal Relations. In *Computational Intelligence*, 6:132-144.

[Weida and Litman, 1992] Weida, Robert and Litman, Diane 1992. Terminological reasoning with constraint networks and an application to plan recognition. In *Proc. of the 3 rd International Conference on Principles of Knowledge Representation and Reasoning*, Cambridge, MA. 282–293.

Extending Standard Description Logics

Patrick Lambrix
Department of Computer and Information Science
Linköping University, S-581 83 Linköping, Sweden
email: patla@ida.liu.se

INTRODUCTION

A lot of work is being done on extending description logics with for instance time [1; 2; 4; 13; 14] and part-of [5; 6; 8]. In our group we have been working on extending description logics with defaults [11; 10; 16], time [7] and part-whole reasoning [9]. In this paper we describe briefly our work on the latter two. (The work on part-whole reasoning is together with Lin Padgham and the work on time is together with Ralph Rönnquist).

WITH PART-WHOLE REASONING

The sort of reasoning we may hope to do in a KR system supporting part-of relations includes such things as deducing the existence of composite objects based on the existence of their parts, answering questions such as whether a particular object is a part of some other object, and determining whether one class is a possible building block of another class. In [9] we propose a framework, based on description logics, for representing and reasoning about part-of relations[1].

As a first step we took a relatively simple description logic with only unstructured roles and a limited number of constructs. We considered the case where we know for each kind of parts[2] how many of them have to occur in an individual belonging to a certain composite concept.

To facilitate part-whole reasoning we defined key relations which enable us to maintain a part-of hierarchy. The intuition behind the relation ***builds*** is that if a concept A ***builds*** a concept B, then an individual belonging to concept A is a possible building block for composing an individual belonging to concept B. Possible building blocks can be *parts* or *modules*. The relation of *compositional inclusion* is defined as the transitive closure of the inverse of ***builds***. If B *compositionally includes* A, then any individual belonging to concept A may be useful in building a composite belonging to B. However if A ***builds*** B is not also true, then the A individual must first be used to build intermediate compositions before being used in the B individual. *Compositional inclusion* is a partial order and can be maintained efficiently by a *part-of hierarchy*. In this part-of hierarchy we defined, similar to the case of a is-a hierarchy, the notion of *most composite includee.*

At the individual level we extended a standard *Abox* language to be able to assert also such things as individual x is an n-part or a module of individual y. We defined a new kind of inferencing, *compositional inferencing* which lets the system infer new compositions on the basis of the existence of available parts. A *compositional extension* is an extended *Abox* where the information inferred is acceptable in that it does not violate such restrictions as non-sharing of parts. There may however be large numbers of such extensions. On the basis of these extensions we defined *credulous compositional extensions* which are preferred extensions. In the case where only one result is wanted we can consider the *skeptical compositional conclusion* which is a unique *Abox* containing the inferences of the credulous compositional extensions, up until the point where there is an ambiguous choice to be made in the compositional inferencing process.

WITH TEMPORAL INFORMATION

When approaching the task of mixing description logic with time, we observe that the idea of 'temporal concept' requires some analysis. On the one hand, a concept may be time-dependent in its extension, i.e. the set of objects satisfying its definition is different at different times. This is the case usually dealt with, and it seems to be in some way a first or more immediate case.

A concept may also be time-dependent in its definition, whereby we mean that an object is included in or excluded from the concept extension depending on which development the object shows. For instance, the concept 'traffic-light' defined as a light cycling over being green, yellow, and red is such a concept. This kind of concepts seems to potentially extend beyond the idea of concept languages, because the extension of a concept no longer

[1]We assume the following properties for the part-of relation: (i) a part cannot be a part of itself and (ii) there are no cycles in the part hierarchy. We incorporated these properties in the definition of the *Tbox* and *Abox*.

[2]It is important to allow for a mechanism to distinguish different kinds of parts. We use a part names.

is a set of single objects but more like a set of history fragments involving several objects in some certain development combinations. For instance, the concept of 'chess-game' denotes scenarios involving (at least) two players, a chess board, and chess pieces, where the latter move over the chess board admitting to a small set of specific rules. To cover these kinds of concepts in terms of description logics, one has to reify; to introduce abstract objects that denote scenarios.

Finally, a concept may also change in itself, so that its extension changes because its definition changes. For instance the concept of (legally) 'adult' may change by decreasing the required age. This kind of development is probably even more difficult to deal with within description logics, since for one thing, it means that subsumption relationships change.

In [7] we extended the same base language as [13]. In contrast to most of the other proposals we did not use Allen's interval calculus as the base temporal framework. Instead we studied the combination of LITE (Logic Involving Time and Evolution, e.g. [12]) semantics and the base description logic. LITE is a variation to first-order predicate logic where in particular the notion of *object* is revised from being an indivisible entity into being a temporal structure of *versions*. Each object version is then indivisible and unchanged in time and represents an appearance of the object at a certain time. As a first step we concentrated on extending the syntax and semantics of a basic description logic to include temporal information about changing extensions of an object and concepts defined in terms of development of objects, but we did not give a subsumption algorithm.

In our attempt to combine the LITE semantics with description logic, we defined a term valuation function relative to a structure so that the extension of a concept is a set of pairs $\langle d, t \rangle$ of objects d and time slices t. We call such a pair a *pointer* which points to the version $d@t$. The extension of a role is a set of pointer pairs.

The constructs in our language can be divided into three classes. The *standard terms* have counterparts in the base language and their definition follows as much as possible the standard definitions for these terms. The *primary temporal terms* are used for temporal qualifications based on inter-object synchronization. For instance, the concept term (**when** r c) is in principle the same as (**all** r c) but with the addition of the constraint that the role filling pointer is synchronized with the time at which the role is filled. The *secondary temporal terms* allow synchronization based on object development. The outcome of a (**seq** $r1$ $r2$) term, for example, is a new role which denotes the transition from the first to the last object versions between which the indicated development (described by $r1$ and $r2$) appears. As an example a 'safe intersection' in a traffic situation is described in [7].

An advantage of using LITE is that we can express mixed temporal relations. Therefore roles are allowed to relate objects in different temporal contexts. Further, as properties are associated to versions we do not need Shoham's hereditary properties [15] to conclude information about sub-intervals. Our secondary temporal terms have some resemblance with the use of the plan description forming operators in CLASP [3]. A plan could be seen as a transition function between times and therefore as a role where a pointer at one time is associated with a pointer at another time involving the same object.

References

[1] Artale, A., Franconi, E., 'A Computational Account for a Description Logic of Time and Action', *Principles of Knowledge Representation and Reasoning: Proceedings of the Fourth International Conference - KR 94*, 1994.

[2] Bettini, C., 'A Formalization of Interval-Based Temporal Subsumption in First Order Logic', *Workshop Notes of the ECAI Workshop on Theoretical Foundations of Knowledge Representation and Reasoning*, 1992.

[3] Devanbu,P., Litman, D., 'Plan-Based Terminological Reasoning', *Principles of Knowledge Representation and Reasoning: Proceedings of the Second International Conference - KR 91*, pp 128-138, 1991.

[4] Fisher, M., 'The Integration of Temporal Operators into a Terminological Representation System', KIT Report 99, Technische Universität Berlin, 1992.

[5] Franconi, E., 'A Treatment of Plurals and Plural Qualifications based on a Theory of Collections', *Minds and Machines*, Vol 3(4), pp 453-474, 1993.

[6] Jang, Y., Patil, R., 'KOLA: A knowledge organization language', *Proceedings of the 13th Symposium on Computer Applications in Medical Care*, 1989.

[7] Lambrix, P., Rönnquist, R., 'Terminological Logic Involving Time and Evolution: A Preliminary Report', *Proceedings of the Seventh International Symposium on Methodologies for Intelligent Systems - ISMIS 93*, pp 162-171, 1993.

[8] Napoli, A., 'Subsumption and Classification-Based Reasoning in Object-Based Representations', *Proceedings of the 10th European Conference on Artificial Intelligence*, pp 425-429, 1992.

[9] Padgham, L., Lambrix, P., 'A Framework for Part-Of Hierarchies in Terminological Logics', *Principles of Knowledge Representation and Reasoning: Proceedings of the Fourth International Conference - KR 94*, 1994.

[10] Padgham, L., Nebel, B., 'Combining Classification and Nonmonotonic Inheritance Reasoning: A First Step', *Proceedings of the Seventh International Symposium on Methodologies for Intelligent Systems - ISMIS 93*, pp 132-141, 1993.

[11] Padgham, L., Zhang, T., 'A Terminological Logic with Defaults: A Definition and an Application', *Proceedings of the International Conference on Artificial Intelligence - IJCAI 93*, pp 662-668, 1993.

[12] Rönnquist, R., *Theory and Practice of Tense-bound Object References*, Ph.D. Thesis, 270, Dpt. of Computer Science, Linköping University, 1992.

[13] Schmiedel, A., 'A Temporal Terminological Logic', *Proceedings of the National Conference on Artificial Intelligence - AAAI 90*, pp 640-645, 1990.

[14] Schild, K., 'A Tense-Logical Extension of Terminological Logics', KIT Report 92, Technische Universität Berlin, 1991.

[15] Shoham, Y., 'Temporal Logics in AI : Semantical and Ontological Considerations', *Artificial Intelligence*, Vol 33(1), pp 89-104, 1987.

[16] Wahllöf, N., forthcoming Lic. Thesis, Dpt. of Computer Science, Linköping University, 1994.

CLASSIC extended with whole-part relations*

Piet-Hein Speel
University of Twente

Peter F. Patel-Schneider
AT&T Bell Laboratories

Abstract

Physically composed objects play a central role in most (commercial) applications. At the moment, the internal structure of these objects cannot be transparently represented in description logics (DLs), nor can the appropriate inferences be drawn. We introduce a small, computationally attractive extension of CLASSIC which handles many aspects of the physical whole-part relation.

1 Introduction

In the literature, considerable attention has been paid to whole-part relations. We distinguish three perspectives. First, the existence of various types of whole-part relations in natural language have been studied, from both a linguistic and a psychological point of view [11; 21; 6]. These relation types have different properties and require different treatment. Second, in knowledge representation systems (KRSs), attempts have been made for representing and reasoning with a general whole-part relation (in contrast to the first perspective) as employed in natural language [20; 5] or common sense knowledge [14; 10], or with a particular whole-part relation to be used in particular applications [13; 9]. Third, a formal, primitive whole-part relation is used in mereology, which is becoming a rather popular research field lately [7].

In this paper, we focus on representation and reasoning aspects of a particular, frequently occurring whole-part relation, called the *physical whole-part relation*. This direction is inspired by three applications.

We propose a restricted whole-part extension for CLASSIC that can be used in many physical whole-part applications, and has attractive computational properties. This fits in with the CLASSIC philosophy, which favors a small but useful, i.e., predictable and efficient, DL.

*The research concerning the physical whole-part relation presented in this paper is mainly established during a three-month visit of the first author at AT&T Bell Labs. We want to thank Paul van der Vet, Rolf Kleef, Jeroen Nijhuis, Deb McGuinness, Jon Wright, Wilco ter Stal, Ron Brachman, Nicolaas Mars, and everyone else who contributed, for their valuable help.

2 An application-based approach

The approach in this paper is inspired by three applications, namely (i) a stereo system application, to be used in a new demo of the CLASSIC system, (ii) telephone switching and transmission products of PROSE, a commercial knowledge-based configurator [22], and (iii) the Plinius project.

The first two applications serve as model for a large number of (commercial) applications in which physically composed products play a central role. In stereo systems and telecommunication products, as in other configurations, whole-part relations are in the application, therefore we want to capture them, if only for representational fidelity reasons. In addition, several types of inferences through whole-part relations are important, as for example, computations of capacity or cost of a whole, based on capacities or costs of its parts.

Title: Dual-Phase Magnesia–Zirconia Ceramics With Strength Retention at Elevated Temperatures.
Abstract: Two-phase polycrystalline ceramics containing MgO and ZrO_2 were fabricated by pressureless sintering powder compacts in air to near theoretical density.
[...]
Some samples were also fabricated in which 8 mol.% CaO was added to stabilize the high-temperature cubic polymorph of zirconia to room temperature.
[...]
Fracture toughness of samples $\underline{\text{containing CaO}}$ was 3.0 MPa$\cdot\sqrt{m}$ while that of the samples $\underline{\text{without CaO}}$ was 5.2 MPa$\cdot\sqrt{m}$.
[...]

Table 1: A Plinius input document. This document roughly consists of two parts. In the first part, the process of producing "two-phase magnesia-zirconia" is described where the internal structure of the material is tried to be manipulated due to, among others, the addition of CaO. In the second part, properties of the resulting samples are studied.

For the Plinius project, we show the need for a physical whole-part relation with corresponding inferences in some detail. The Plinius project is a research project currently undertaken by the Knowledge-Based Systems Group at the University of Twente. The aim of the Plinius project is to build a system which is able to semi-automatically extract domain knowledge from title and abstract of scientific publications in the field of ceramic science [12; 19]. An example of an input document is presented in Table 1.

$\mathcal{E}$ is the class of Chemical elements: Ac, Ag, Al, ..., Zn, Zr.
$\mathcal{N}$ is the class of natural numbers.
A concept Group g stands for a set of tuples g_i where

$$g_i = \langle e, n \rangle \text{ with } e \in \mathcal{E} \text{ and } n \in \mathcal{N}.$$

E.g., $g1 \overset{\text{def}}{=} \{\langle \text{Mg}, 1\rangle\}$; $g2 \overset{\text{def}}{=} \{\langle \text{O}, 1\rangle\}$; $g3 \overset{\text{def}}{=} \{\langle \text{Zr}, 1\rangle\}$; $g4 \overset{\text{def}}{=} \{\langle \text{Ca}, 1\rangle\}$
A concept Pure substance ps stands for a set of tuples ps_i where

$$ps_i = \langle g, n \rangle \text{ with } g \text{ a Group and } n \in \mathcal{N}.$$

E.g., $ps1 \overset{\text{def}}{=} \{\langle g1, 1\rangle, \langle g2, 1\rangle\}$; $ps2 \overset{\text{def}}{=} \{\langle g3, 1\rangle, \langle g2, 2\rangle\}$; $ps3 \overset{\text{def}}{=} \{\langle g4, 1\rangle, \langle g2, 1\rangle\}$.
$\mathcal{A}$ is the class of aggregation states.
$\mathcal{W}$ is the class of real numbers between 0 and 1.
A concept Phase ph stands for a tuple

$$ph = \langle c, a \rangle \text{ with } a \in \mathcal{A}$$

where c specifies the concept Chemical composition which stands for a set of tuples c_i:

$$c_i = \langle ps_i, w_i \rangle \text{ with } w_i \in \mathcal{W}$$

where ps_i stands for a Pure substance, and $\sum_i w_i = 1$.
E.g., $ph1 \overset{\text{def}}{=} \langle\{\langle ps1, 0.8\rangle, \langle ps2, 0.2\rangle\}, \text{cubic}\rangle$;
$ph2 \overset{\text{def}}{=} \langle\{\langle ps1, 0.76\rangle, \langle ps2, 0.16\rangle, \langle ps3, 0.08\rangle\}, \text{cubic}\rangle$.

Table 2: A simplified part of the Plinius ontology. Whole-part relations exist between the concepts Chemical elements, Groups, Pure substances, and Phase. For example, *ps*3 *is part of ph*1 and *ps*3 *is not part of ph*2.

Natural language processing (NLP) is used to extract domain knowledge from the texts. This process makes use of both language-dependent knowledge bases (a grammar and the language-dependent part of a lexicon) and domain-specific knowledge bases (an ontology, the domain-specific part of the lexicon, and background knowledge). The Plinius ontology [18] specifies the relevant concepts and relations in the ceramics domain (roughly the domain consists of samples, materials, processes and properties). A simplified part of the ontology is presented in Table 2. For a CLASSIC representation of this ontology, see [17]. NLP can be divided into syntactic, semantic and discourse analysis, where the discourse analysis includes reference resolution.

When processing the document given in Table 1, derived whole-part information is needed during the reference resolution process. For example, the two underlined phrases in Table 1 refer to samples of a material with and without CaO (corresponding to *ph*2 and *ph*1 in Table 2), respectively. If it is possible to check whether CaO (corresponding to *ps*3 in Table 2) is part of these materials, it is possible to add the property results to the appropriate samples.

This means that in the process of reference resolution, an operation is needed in order to determine whether some object/concept is part of another object/concept. Computations of transitive closures of whole-part relations play an important role in this operation.

3 Physical whole-part properties

Based on the intuitions formed by studying the three applications mentioned above (see [18] for the Plinius intuitions), we specify the properties of physically composed objects and physical whole-part relations.[1]

If an object is *physically composed* then: (i) it consists of a nonempty set of part-objects; (ii) it cannot be a physical part of itself; (iii) each part-object is connected[2] to at least one other part-object (in case two or more parts are available); and (iv) its properties are constrained by properties of its part-objects. Important properties of physical objects are behavior and form, including geometry (such as size, location and shape) and material.

The following rule can always be applied to physically composed objects: *"When a physically composed object exists, its parts exist as well."*

Given these properties, we introduce the following "contains" relation between objects:[3]

1. w *directly contains* p, with w and p objects, iff w is a physically composed object and p is an element of the set of part-objects of w;
2. w *contains* p, with w and p objects, iff w *directly contains* p, or $\exists$ object x, such that w *directly contains* x and x *contains* p;
3. the *contains* relation between objects is transitive and asymmetric.

The inverse relation is called *"is part of"*.

In addition, we introduce the "contains" and "is part of" relations between concepts:

1. W *contains* P, with W and P concepts, iff $\forall w \in W\ \exists p \in P\ w$ *contains* p;
2. P *is part of* W, with P and W concepts, iff $\forall p \in P\ \exists w \in W\ p$ *is part of* w;
3. both relations are transitive, but (in contrast to the contains relation between objects) not asymmetric.

[1]See [16], for additional specifications of physical whole-part properties.

[2]Within the three applications mentioned above, the connection relation is considered to be symmetric and intransitive.

[3]Within the three applications mentioned above, we did not find the need to represent and reason with exhaustive decompositions and disjointness and overlap relations. In our extension, we have left a door open to additional, appropriate constructions and inferences in case these are needed in other applications.

Various *whole-part queries* need to be answered. For example, we distinguish *connection queries*: "Is part x connected to part y in whole z?" from *contains queries*: "Does x *contain* y?" and "*Is* y *part of* x?".

4 The CLASSIC whole-part extension

Constructions for physical whole-part extensions for DLs in general are proposed in [16]. In this paper, we focus on a computationally attractive subset of these constructions.

4.1 Whole-part roles

When defining a CLASSIC description for a physically composed individual/concept, the parts need to be declared. For this purpose, a new kind of role, called *whole-part role*, is introduced, together with a whole-part hierarchy and the inverse *part-whole role*. This new role is to be treated separately from existing roles. This leads to the following extension, where $\mathtt{W}$ is a whole-part role, $\mathtt{WN}$ a whole-part role name, and $\mathtt{WN}^{\mathcal{I}} : \Delta^{\mathcal{I}} \to 2^{\Delta^{\mathcal{I}}}$:[4]

Axioms

Syntax	Semantics
$\mathtt{WN} \doteq_w \mathtt{W}$	$\mathtt{WN}^{\mathcal{I}} = \mathtt{W}^{\mathcal{I}}$
$\mathtt{WN} \sqsubseteq_w \mathtt{W}$	$\mathtt{WN}^{\mathcal{I}} \subseteq \mathtt{W}^{\mathcal{I}}$

Whole-part role descriptions

Syntax	Semantics
$\mathtt{W}$	$\mathtt{W}^{\mathcal{I}}$
$\mathtt{W}^{-1}$	$\{(d,d') \mid (d',d) \in \mathtt{W}^{\mathcal{I}}\}$

Concept descriptions

Syntax	Semantics
$\forall\mathtt{W} : \mathtt{C}$	$\{d \in \Delta^{\mathcal{I}} \mid \mathtt{W}^{\mathcal{I}}(d) \subseteq \mathtt{C}^{\mathcal{I}}\}$
$\geq n\ \mathtt{W}$	$\{d \in \Delta^{\mathcal{I}} \mid \lvert\mathtt{W}^{\mathcal{I}}(d)\rvert \geq n\}$
$\leq n\ \mathtt{W}$	$\{d \in \Delta^{\mathcal{I}} \mid \lvert\mathtt{W}^{\mathcal{I}}(d)\rvert \leq n\}$
$\mathtt{W} : \mathtt{IN}$	$\{d \in \Delta^{\mathcal{I}} \mid \mathtt{IN}^{\mathcal{I}} \in \mathtt{W}^{\mathcal{I}}(d)\}$

For example,[5] the concept description $\forall\mathtt{ampl} : \mathtt{AMPLIFIER} \sqcap \geq 1\ \mathtt{ampl}$, extracted from Figure 1, denotes the following set of objects:

$$\{d \in \Delta^{\mathcal{I}} \mid \mathtt{ampl}^{\mathcal{I}}(d) \subseteq \mathtt{AMPLIFIER}^{\mathcal{I}} \wedge \lvert\mathtt{ampl}^{\mathcal{I}}(d)\rvert \geq 1\}.$$

This means that all objects denoted by the concept description contain at least one amplifier object. This corresponds to the *contains* relation[6] between concepts given in Section 3.

In order to represent number-restrictions between various parts, a whole-part role hierarchy is used. This hierarchy is independent from other role hierarchies, because it does not make sense to combine whole-part roles with other roles in a hierarchy. See Figure 1 for example, where TWO_HOUSED_STEREO_SYSTEM is defined as a stereo system

$$\begin{aligned}
&\mathtt{sh\_comp} \sqsubseteq_w \top \times \top\\
&\mathtt{ampl, cass, cd} \sqsubseteq_w \mathtt{sh\_comp}\\
&\mathtt{SHELF\_COMPONENT} \sqsubseteq_c \top\\
&\mathtt{AMPLIFIER, CASSETTE\_DECK, CD\_PLAYER} \sqsubseteq_c\\
&\quad \mathtt{SHELF\_COMPONENT}\\
&\mathtt{TWO\_HOUSED\_STEREO\_SYSTEM} \sqsubseteq_c\\
&\quad \forall\mathtt{sh\_comp} : \mathtt{SHELF\_COMPONENT} \sqcap\ \leq 2\ \mathtt{sh\_comp} \sqcap\\
&\quad \forall\mathtt{ampl} : \mathtt{AMPLIFIER} \sqcap\ \geq 1\ \mathtt{ampl} \sqcap\ \leq 1\ \mathtt{ampl} \sqcap\\
&\quad \forall\mathtt{cass} : \mathtt{CASSETTE\_DECK} \sqcap\ \leq 2\ \mathtt{cass} \sqcap\\
&\quad \forall\mathtt{cd} : \mathtt{CD\_PLAYER} \sqcap\ \leq 1\ \mathtt{cd}
\end{aligned}$$

Figure 1: A whole-part role hierarchy

that needs to be placed in a rack with two shelves (without speakers). Although, various configurations may occur, a stereo system needs to have exactly one amplifier. Using whole-part role **sh_comp** (shelf component) as parent of whole-part roles **ampl**, **cass**, and **cd**, we are able to represent this number-restriction.

$$\begin{aligned}
&\mathtt{pow\_ampl} \sqsubseteq_w \top \times \top\\
&\mathtt{int\_ampl} \doteq_w \mathtt{pow\_ampl}^{-1}\\
&\mathtt{POWER\_AMPLIFIER} \sqsubseteq_c \top\\
&\mathtt{INTEGRATED\_AMPLIFIER} \sqsubseteq_c\\
&\quad \geq 1\ \mathtt{pow\_ampl}\\
&\mathtt{UN\_HOUSED\_POWER\_AMPLIFIER} \doteq_c\\
&\quad \mathtt{POWER\_AMPLIFIER} \sqcap\\
&\quad \forall\mathtt{int\_ampl} : \mathtt{INTEGRATED\_AMPLIFIER} \sqcap\\
&\quad \geq 1\ \mathtt{int\_ampl} \sqcap\ \leq 1\ \mathtt{int\_ampl}
\end{aligned}$$

Figure 2: A part-whole role

When defining a concept description for a part, it may be useful to declare the whole it belongs to. Using the $\forall\mathtt{W}^{-1} : \mathtt{C} \sqcap \geq 1\ \mathtt{W}^{-1}$ construction, one is able to represent the *is part of* relation between concepts[7] given in Section 3. For example, a power amplifier which does not occur separately, but only as part of an integrated amplifier, can be represented by UN_HOUSED_POWER_AMPLIFIER in Figure 2.

The choice to use a role to represent physical whole-part relations has consequences in our view. [3] states the following about roles:

> *Any generalized attribute* [...] *has two important pieces: (1) the particular entity that becomes the value for the attribute in an instance of the Concept, and (2) the functional role which that entity fills in the conceptual complex.*

In DLs, roles are treated this way which means that a "hate" role between two persons is treated differently from a "love" role between two persons. Therefore,

[4]See [1] or [15] for details of DLs syntax and semantics.

[5]The examples here will be drawn from the stereo system application, because no specialized knowledge is required.

[6]Without the ≥ 1 **ampl** construction, the intuitive "*may contain*" relation between concepts is represented.

[7]Omitting $\geq 1\ \mathtt{W}^{-1}$ in this construction, results in the intuitive "*may be part of*" relation between concepts.

users of DLs expect roles to have functions. In order to be consistent with this use of roles, we think that whole-part roles implicitly have functionality too. The advantage of this approach is that it is possible to define a stereo system, for example, which contains a cassette deck in a "record" function, which will be treated differently from a stereo system which contains a cassette deck in a "play" function. However, a disadvantage is that functional whole-part relations have different properties. Therefore, a whole-part role is transitive from a "physical" point of view and intransitive from a "functional" point of view (as is concluded in the linguistic approach), which has consequences for inferences as will be shown in Subsection 4.5.

4.2 Acyclicity of whole-part roles

Since the possibility to construct cycles formed by whole-part roles between individuals needs to be avoided, as is illustrated in Figure 3, we have modified the semantics of axiom $\mathtt{IN} \in \mathtt{C}$, where $\mathtt{W}_i$ is a whole-part role:[8]

Syntax: $\mathtt{IN} \in \mathtt{C}$
Semantics: $\mathtt{IN}^{\mathcal{I}} \in \mathtt{C}^{\mathcal{I}} \sqcap \mathtt{IN}^{\mathcal{I}} \notin (\mathtt{W}_1^{\mathcal{I}} \cup \ldots \cup \mathtt{W}_n^{\mathcal{I}})^{\mathtt{trans}}(\mathtt{IN}^{\mathcal{I}})$

where $\mathtt{W}_1, \ldots, \mathtt{W}_n$ are all the primitive, atomic whole-part roles and $\cdot^{\mathtt{trans}}$ is the transitive closure. This means that an object cannot be an argument of a "whole-part" object-tuple which is part of a cycle of "whole-part" object-tuples.

$\mathtt{sh\_comp} \sqsubseteq_w \top \times \top$
$\mathtt{ampl}, \mathtt{cd} \sqsubseteq_w \mathtt{sh\_comp}$
$\mathtt{AMPLIFIER}, \mathtt{CD\_PLAYER} \sqsubseteq_c \top$
$\mathtt{STEREO\_SYSTEM} \sqsubseteq_c$
$\quad \forall \mathtt{ampl} : \mathtt{AMPLIFIER} \sqcap \geq 1\ \mathtt{ampl} \sqcap \leq 1\ \mathtt{ampl} \sqcap$
$\quad \forall \mathtt{cd} : \mathtt{CD\_PLAYER} \sqcap \leq 1\ \mathtt{cd}$
$\mathtt{DISCMAN} \sqsubseteq_c \mathtt{CD\_PLAYER} \sqcap \mathtt{STEREO\_SYSTEM}$
$\mathtt{Discman\_D34} \in \mathtt{DISCMAN}$
$\mathtt{Discman\_D34} \in \mathtt{STEREO\_SYSTEM} \sqcap \mathtt{cd} : \mathtt{Discman\_D34}$

Figure 3: A cycle of individual whole-part roles: A discman is represented as an integrated stereo system configuration. In addition, it can play the role of cd player in a stereo system. In this example, a discman individual is created which is part of itself

4.3 Connections between parts

In order to represent connections between parts, we introduce the following concept description:[9]

Syntax: $\mathtt{W}_1 \xrightarrow{\mathtt{T}} \mathtt{W}_2$
Semantics:
$\{d \in \Delta^{\mathcal{I}} \mid \forall i \in \mathtt{W}_1^{\mathcal{I}}(d)\, \forall j \in \mathtt{W}_2^{\mathcal{I}}(d)\, (i \neq j \Rightarrow \langle i,j \rangle \in \mathtt{T}^{\mathcal{I}})\}$

This means that an object/concept satisfying this construction has the property that all parts corresponding to $\mathtt{W}_1$ are connected to all parts corresponding to $\mathtt{W}_2$ (as long as they are not the same) via a connection of type $\mathtt{T}$.

When looking at stereo system configurations again, we are now able to represent connections between the components of stereo systems. For example, concept $\mathtt{STEREO\_SYSTEM}$ in Figure 4 contains exactly one amplifier, which is connected to the (possible) cassette decks and the (possible) cd player via $\mathtt{play\_c}$ connections.

$\mathtt{ampl}, \mathtt{cass}, \mathtt{cd} \sqsubseteq_w \top \times \top$
$\mathtt{CASSETTE\_DECK} \sqsubseteq_c \top$
$\mathtt{CD\_PLAYER} \sqsubseteq_c \top$
$\mathtt{AMPLIFIER} \sqsubseteq_c \top$
$\mathtt{STEREO\_SYSTEM} \sqsubseteq_c$
$\quad \forall \mathtt{ampl} : \mathtt{AMPLIFIER} \sqcap \geq 1\ \mathtt{ampl} \sqcap \leq 1\ \mathtt{ampl} \sqcap$
$\quad \forall \mathtt{cass} : \mathtt{CASSETTE\_DECK} \sqcap \leq 2\ \mathtt{cass} \sqcap$
$\quad \forall \mathtt{cd} : \mathtt{CD\_PLAYER} \sqcap \leq 1\ \mathtt{cd} \sqcap$
$\quad \mathtt{ampl} \xrightarrow{\mathtt{play\_c}} \mathtt{cass} \sqcap \mathtt{ampl} \xrightarrow{\mathtt{play\_c}} \mathtt{cd}$

Figure 4: Connections between parts

4.4 Whole-part constraints

We temporarily use the **test-c** construction and CLASSIC rules to cover pieces of the whole-part constraints, pending a declarative constraint extension[10] to come in the (near) future.

4.5 Whole-part inferences

Finally, we distinguish three types of inferences (according to the division of [4]): (i) *completion inferences*, including inheritance of whole-part knowledge down the taxonomy, computations of transitive closures of physical whole-part roles,[11] propagations of consequences due to whole-part constraints, and creations of part individuals when a physically composed individual is created (only parts of the individual creation inferences are implemented); (ii) *contradiction detection inferences*, including contradictions due to whole-part role number/value restrictions, detections of cycles of individual whole-part roles, and contradictions due to whole-part constraints; and (iii) *classification and subsumption inferences*, where whole-part knowledge needs to be taken into consideration.

[8]This means that an object cannot be an argument of a "whole-part" object-tuple which is part of a cycle of "whole-part" object-tuples.

[9]This means that an object/concept satisfying this construction has the property that all parts corresponding to $\mathtt{W}_1$ are connected to all parts corresponding to $\mathtt{W}_2$ (as long as they are not the same) via a connection of type $\mathtt{T}$.

[10]See [8] for a general approach in which representing and reasoning with constraints in a declarative manner is integrated with representation and reasoning in DLs.

[11]See [14], for an extended approach of these transitive closure inferences .

Following the CLASSIC philosophy, we want the types of inferences for the CLASSIC extension to have attractive computational properties. Most types of inferences belonging to this extension, satisfy this property. In particular, we expect the computations of transitive closures of whole-part relations to have the same complexity results as inferences in CLASSIC role-hierarchies (namely, of exponential complexity in worst case).

In addition, we focus on subsumption computations. Since the functional whole-part role is intransitive (Subsection 4.1), transitive closures of whole-part roles do not influence subsumption inferences. This means that in some cases subsumption computations hold from a "functional" point of view but do not hold from a "physical" point of view. Users interested in physical whole-part relations are restricted to special purpose queries which support the "physical" point of view (including the transitivity of physical whole-part roles). In addition, in the whole-part extension proposed here, constraints, represented by **test-c** constructions and CLASSIC rules, neither influence subsumption inferences. Given these restrictions and given the complexity results of the subsumption algorithm in standard CLASSIC [2], the subsumption algorithm for extended CLASSIC belongs to the same complexity class, since (1) whole-part roles are treated separately from other types of roles, and (2) the connection constructions only add simple computations, when appropriately represented internally.

4.6 Implementation

The whole-part constructions of the CLASSIC extension, introduced in Subsections 4.1–4.4, have been implemented on top of the current version of CLASSIC. Besides this whole-part representation extension, we have added most of the corresponding inferences mentioned in Subsection 4.5. In addition, the whole-part extension is able to answer contains queries. For creation of part individuals when a physically composed individual is created, a special operation will be added which follows user instructions.

5 Conclusion

From a user point of view, we have focused on the particular, frequently occurring physical whole-part relation. For this relation, we have specified its properties in detail and we have motivated the applicability. We think the extension described in this paper may be useful for both research applications, such as projects including NLP, and practical applications, such as projects focused on configurations.

Adding full support for the physical whole-part relationship (as described in [16]) has severe computational and implementation consequences. The computationally attractive subset presented in this paper captures only a portion of whole-part constructions, but may be sufficient to be useful in many applications. In this way the utility of DLs like CLASSIC can be extended while still retaining their desirable computational properties.

The work presented in this paper is an initial step in defining a whole-part module for CLASSIC. This module need to be further refined and developed, both from a theoretical and a practical point of view.

References

[1] Franz Baader, Hans-Jürgen Bürckert, Jochen Heinsohn, Bernhard Hollunder, Jürgen Müller, Bernhard Nebel, Werner Nutt, and Hans-Jürgen Profitlich. Terminological knowledge representation: A proposal for a terminological logic. German Research Center for Artificial Intelligence (DFKI), June 1991.

[2] Alex Borgida and Peter F. Patel-Schneider. A semantics and complete algorithm for subsumption in the classic description logic. *Journal of Artificial Intelligence Research*, 1:pages 277–308, 1994.

[3] Ronald J. Brachman. On the epistemological status of semantic networks. In Nicholas V. Findler, editor, *Associative Networks: Representation and Use of Knowledge by Computers*, pages 3–50. Academic Press, New York, 1979.

[4] Ronald J Brachman. "reducing" CLASSIC to practice: Knowledge representation theory meets reality. In *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning*, pages 247–258, San Mateo, California, October 1992. Morgan Kaufmann.

[5] Enrico Franconi and Vania Rabito. A relation-based description logic. In Franz Baader, Maurizio Lenzerini, Werner Nutt and Peter F. Patel-Schneider, editors, *Working Notes of the 1994 Description Logic Workshop*, DFKI Report, May 28–29, 1994.

[6] Peter Gerstl. Midwinters, end games, and body parts: A classification of part-whole relations. In Nicola Guarino and Roberto Poli, editors, *International Workshop on Formal Ontology in Conceptual Analysis and Knowledge Representation*, pages 251–260, March 17–19, 1993.

[7] Nicola Guarino and Roberto Poli, editors. *Proceedings of the International Workshop on Formal Ontology in Conceptual Analysis and Knowledge Representation*, March 17–19, 1993.

[8] Philipp Hanschke. *A Declarative Integration of Terminological, Constraint-Based, Data-Driven, and Goal-Directed Reasoning.* PhD thesis, University of Kaiserslautern, Kaiserslautern, Germany, 1993.

[9] Yeona Jang and Ramesh Patil. KOLA: Knowledge organization language. Technical Report TR-396, Massachusetts Institute of Technology, 1988.

[10] Douglas B. Lenat and R. V. Guha. *Building Large Knowledge-Based Systems.* Addison-Wesley, Reading, Massachusetts, 1990.

[11] Judith A. Markowitz, J. Terry Nutter, and Martha W. Evans. Beyond is-a and part-whole: More semantic network links. In Fritz Lehmann,

editor, *Semantic networks in artificial intelligence*, pages 377–390. Pergamon Press, Oxford, England, 1992.

[12] Nicolaas J.I. Mars, Wilco ter Stal, Hidde de Jong, Paul E. van der Vet, and Piet-Hein Speel. Semi-automatic knowledge acquisition in Plinius: An engineering approach. Memorandum UT-KBS-93-37, University of Twente, Enschede, the Netherlands, 1993.

[13] Amedeo Napoli. Subsumption and classification-based reasoning in object-based representations. In *Proceedings of the Tenth European Conference on Artificial Intelligence*, pages 425–429, Chichester, United Kingdom, August 1992. European Coordinating Committee for Artificial Intelligence, John Wiley.

[14] Lin Padgham and Patrick Lambrix. A framework for part-of hierarchies in terminological logics. In *Proceedings of the Fourth International Conference on Principles of Knowledge Representation and Reasoning*, San Mateo, California, May 1994. Morgan Kaufmann.

[15] Peter F. Patel-Schneider and Bill Swartout. Description-logic knowledge representation system specification. Available from AI Principles Research Department, AT&T Bell Labs., 1993.

[16] Piet-Hein Speel and Peter F. Patel-Schneider. A whole-part extension for description logics. In *Notes of the ECAI'94 Workshop Parts and Wholes: Conceptual part-whole relations and formal mereology*, Amsterdam, The Netherlands, August 8, 1994.

[17] Piet-Hein Speel, Paul E. van der Vet, Wilco G. ter Stal, and Nicolaas J.I. Mars. Formalization of an ontology of ceramic science in CLASSIC. In Karen S. Harber, editor, *Proceedings of the Seventh International Symposium on Methodologies for Intelligent Systems, Poster Session*, pages 110–124, Oak Ridge, Tennessee, June 15–18, 1993. Oak Ridge National Laboratory.

[18] Paul E. van der Vet and Nicolaas J. I. Mars. Structured system of concepts for storing, retrieving, and manipulating chemical information. *Journal of Chemical Information and Computer Sciences*, 33:pages 564–568, 1993.

[19] Paul E. van der Vet, Hidde de Jong, Nicolaas J. I. Mars, Piet-Hein Speel, and Wilco G. ter Stal. Plinius intermediate report. Memorandum UT-KBS-94-10, University of Twente, Enschede, the Netherlands, 1994.

[20] Robert Wilensky. Some problems and proposals for knowledge representation. Technical Report UCB/CSD 86/294, Computer Science Division, University of California at Berkeley, May 1986.

[21] Morton E. Winston, Roger Chaffin, and Douglas Herrmann. A taxonomy of part-whole relations. *Cognitive Science*, 11(4):417–444, 1987.

[22] Jon R. Wright, Elia S. Weixelbaum, Karen Brown, Gregg T. Vesonder, Stephen R. Palmer, Jay I. Berman, and Harry H. Moore. A knowledge-based configurator that supports sales, engineering, and manufacturing at AT&T network systems. In *Proceedings of the Innovative Applications of Artificial Intelligence Conferences*, pages 183–193, Menlo Park, California, July 1993. American Association for Artificial Intelligence.

Using Description Logics for Software Reuse and Case-based Reasoning

John Yen, Swee Hor Teh and Xiaoqing Liu
Department of Computer Science, HR Bright Building
Texas A&M University College Station, TX 77843-3112
(409) 845-5481 shteh@cs.tamu.edu

Abstract

One of the major difficulties in measuring similarity between concepts is to find their commonality and differences based on their meanings. Given a target definition t, the goal is to find a degree of similarity between the target instance t and a source definition S for retrieval and reuse purposes. We present a principled and consistent measure to find the concept similarity using a classification taxonomy as a framework for our semantic-based analysis. Our approach offers several important benefits. First, it reasons and analyzes the meanings of concepts. Second, concepts are not defined in terms of a predefined set of properties. This means that our approach allows for more meaningful definitions. Future work includes using the measure to retrieve reusable software components from a library. Using the classification taxonomy and this measure, most similar reusable software component can be found and retrieved efficiently.

1 Introduction

A principal purpose of software reuse is to allow users to retrieve software components, to assess their applicability, and finally reuse the desired component in a new application. Like software reuse, the interests in case-based reasoning are in the analysis and retrieval of similar cases. software reuse and case-based reasoning involve several issues: (1) the classification and storage of concepts (cases) in the library; (2) semantic-based analysis of concepts; and (3) intelligent retrieval of similar cases. In previous work, we addressed the first issue using a principled modeling method [10] that allows developers to have a high-level view of the system requirements and functionality of an expert system [12; 13]. This paper focuses on the second issue. The last issue is ongoing research.

Similarity measurements are applied in the areas of software reuse [6], case-based reasoning [5], analogical reasoning [7] and retrieval [9]. Even though similarity between concepts has long been recognized as an important part of selecting concepts, few efforts have been made to develop a measure based on the meanings of definitions. Instead most measures (1) depend on strict and definite structures [5; 4; 3] that ignore the semantics in the problem domain or (2) use heuristics [2]. For example, Domeshek's system, Abby [1], uses a frame representation with strict use of atomic slot-fillers and a fixed structure. It only handles a fixed relationship possibility. Heuristics for structuring and discriminating between similar cases are often domain dependent. To address these problems, we have developed a principled and consistent measure of similarity between concepts. The *measure of degree of similarity* represents how prominent and different the target class is and what are the differences when compared to other classes in a library. Our approach reasons and analyzes the semantics of concepts to determine their commonalities and differences. The CaPER system [4] differs from our approach because it measures cases that have shared features but does not determine the differences in negative cases.

Our approach provides a precise measure to determine the commonalities between concepts in a library of expert systems for *reuse purpose*. The primary purpose of this measure is to determine how similar a target instance is to an existing class in the repository. This measure is used to retrieve the most similar software solution available in the repository for reuse. We use LOOM [8] to express the semantics of concepts and the taxonomic structure of a terminological knowledge base to infer this measure. Wall et al. [11] also used the taxonomic structure as a framework for retrieval and explanation in case-based reasoning but it still requires a "man in the loop" to determine the most similar case.

Retrieval Procedure The goal of retrieval is to find reusable instances in the terminological knowledge base given a target instance, t. If there is a local instance i in the classified class C in which t is classified, then i is a candidate for reuse. However if there is no local instance associated with C then we search for reusable instances in its superclasses P_j. If there is more than one immediate superclass then we determine the set of instances in each class using the heuristic function. The heuristic function measures the similarity between concepts and it is used to determine the order in which the superclasses are explored. Hence, the heuristic function guides the search for reusable instances based on how similar the instances in a class are to t. If we fail to locate a reusable instance in P_j, then we search for reusable instances in the subclasses of C, C_j. The distance measure is used to determine the most similar instance in a class. This measure is only used after the candidate class is found. Our focus is to determine how different the instances are compared to t and rank these instances in an increasing order. A smaller distance value implies that the similarity of an instance to t is higher. This process continues until the most similar instance is found.

2 Semantic-based Measure for Similarity

Our goal is to determine the relationships between a target instance t and a source concept S given t. Hence, the primary purpose of having a measure is to determine how similar a target instance is to an existing class in the repository. This measure is used to retrieve the most similar software component available in the repository for reuse purposes. This measure has two main objectives. First, it is used to determine which candidate concept has the highest likelihood for the target instance to occur. The candidate concept is the one with the highest valued ratio. Second, it is used to find how different the instances (in the candidate class) are from the target instance. This measure, in turn, provides us with information on how similar the instance target is to the chosen instance.

A heuristic function is used to determine the candidate concept in which the instance target is most likely to occur. The heuristic function measures the chances in which the target instance t occurs in another class S assuming that S subsumes t and S is a concept existing in the repository. If there is more than one candidate concept then this function provides a means to determine in which each superclass should be explored. A distance measure is used to determine how different the instances in a candidate concept is from the target instance. This measure provides information on how far the target instance is to the instances in the candidate concept.

We use the taxonomic classification structure of a terminological knowledge base $\mathcal{T}$ as a framework for calculating the similarity measure. $\mathcal{T}$ consists of concepts and roles (relations) defined using a term subsumption language. An interpretation $\mathcal{I}_\mathcal{T}$ of $\mathcal{T}$ is a pair $(\mathcal{D}, \mathcal{E})$ where $\mathcal{D}$ is a set of individuals described by terms in $\mathcal{T}$ and $\mathcal{E}$ is an extension function that maps concepts in $\mathcal{T}$ to subsets of $\mathcal{D}$, and roles in $\mathcal{T}$ to subsets of the Cartesian product $\mathcal{D} \times \mathcal{D}$.

Heuristic Function The first factor in calculating the measure of similarity is to determine the likelihood of the target instance t occurring in a source concept, S. Intuitively, since we are interested in the possibility of t being in a superclass then the instances in the superclass and t are related in some way. We assumed that there is a uniform distribution of instances in the instance space. That is, we assume that the likelihood of t occurring in S as conditional independence in probability theory. Given two classes, S_1, S_2, and a target instance t where $S_1 \succ S_2 \succ t$, then we calculate the probability of t occurring in S_1 and in S_2. Assume that t is a target instance in the class T. From the calculated values we determine the instances in S_1 or S_2 that are more similar to t. If $S_1 \succ T$, $S_2 \succ T$ and $P(t|S_1) \succ P(t|S_2)$ then we prefer an instance in S_1 to that in S_2. The rationale is that the number of instances in S_1 and not in T is less than the number in S_2 and not in T. That is $||S_1 \setminus T|| < ||S_2 \setminus T||$ where $\setminus$ means not. Hence, an instance in S_1 is likely to be more similar to T than an instance in S_2.

Definition 1 *(Heuristic Function)*
Let t be the target instance and S be the source concept. Assume that t is an instance in the concept T. The probability of t given S is defined as

$$P(t|S) = \frac{||\varepsilon(t \wedge S)||}{||\varepsilon(S)||}$$

Definition 1 assumes that (1) t is an instance of of some classes T and T contains no other instances besides t, (2) S is in the taxonomy, (3) S and T are related and (3) the library contains a sufficiently large number of instances and classes. It finds a quantifiable value to relate the concepts S and T and compares the probability that t is an instance of other classes. For example, if we define T in terms of C and e_1 and S in terms of C and e_2, then the probability of T given S is the probability of differences in T given S. e_1 and e_2 are different concept forming expressions in T and S respectively. More formally, we have Theorem 1:

Theorem 1 *Let $T = (C \wedge e_1)$ and $S = (C \wedge e_2)$. The probability of T given S is $P(T|S) = P(e_1|e_2)$ assuming that*

$$\frac{||\varepsilon(C \wedge e_1 \wedge e_2)||}{||\varepsilon(C \wedge e_2)||} = \frac{||\varepsilon(e_1 \wedge e_2)||}{||\varepsilon(e_2)||} \tag{1}$$

Proof Following definition 1, we have

$$P(T|S) = \frac{||\varepsilon(C \wedge e_1) \bigcap \varepsilon(C \wedge e_2))||}{||\varepsilon(C \wedge e_2)||}$$

$$= \frac{||\varepsilon(C \wedge e_1 \wedge e_2)||}{||\varepsilon(C \wedge e_2)||}$$

Based on the assumption (1) we have

$$\begin{aligned} P(T|S) &= \frac{||\varepsilon(e_1 \wedge e_2))||}{||\varepsilon(e_2)||} \\ &= P(e_1|e_2) \end{aligned}$$

□

We interpret assumption 1 as:

$$\frac{||\varepsilon(C \wedge e_1 \wedge e_2)||}{||\varepsilon(C \wedge e_2)||} = \frac{||\varepsilon(e_1 \wedge e_2)||}{||\varepsilon(e_2)||}$$

so that it can be interpreted according to probabilitiy theory as:

$$\frac{P(C \wedge e_1 \wedge e_2)}{P(C \wedge e_2)} = \frac{P(e_1 \wedge e_2)}{P(e_2)}$$

$$P(e_1|C, e_2) = P(e_1|e_2)$$

where C and e_1 are conditionally independent on e_2.

If the class S is empty (due to an incomplete library) and there are no experiences described in the library relevant to calculating the desired conditional probability then this theorem fails. This theorem handles situations where T and S share some commonalities and differences.

Because we are interested in the detailed differences in the concepts, we now briefly explain the factors that contribute to making the distinctions. There are two types of concepts: primitive and defined. The differences between primitive concepts are denoted by values provided by the developer. A defined concept is described using other concept forming expressions. Value restriction and number restrictions are examples of two concept forming expressions. A value restriction is expressed as (:all R C_1) and means that all values of the role R are of type C. A number restriction can either be (:at-least l_1 R) which means that the role R has at least l_1 values or (:at-most u_1 R) which states that the role R has at most u_1 values.

T and S can be different in several ways: different value restrictions, different number restrictions, and different value restrictions and number restrictions. Theorem 2 formally states the impact of different value restriction on the heuristic function. e_1 and e_2 are concept-forming expressions that share some superclasses, C, but have different value restrictions, C_1 and C_2 respectively.

Theorem 2 *Let* $e_1 = C \wedge$ *(:all* R C_1*) and* $e_2 = C \wedge$ *(:all* R C_2*). The probability of* T *given* S *is* $P(e_1|e_2) = P(C_1|C_2)$ *assuming that*

$$\frac{\|\{x|\langle x,y\rangle \in \varepsilon(R) \rightarrow y \in \varepsilon(C_1) \wedge \varepsilon(C_2)\}\|}{\|\{x|\langle x,y\rangle \in \varepsilon(R) \rightarrow y \in \varepsilon(C_2)\}\|} = \frac{\|\varepsilon(C_1) \wedge \varepsilon(C_2)\|}{\|\varepsilon(C_2)\|} \quad (2)$$

Proof Using

$$\varepsilon(e_1) = \{x|\langle x,y\rangle \in \varepsilon(R) \rightarrow y \in \varepsilon(C_1)\}$$

and

$$\varepsilon(e_2) = \{x|\langle x,y\rangle \in \varepsilon(R) \rightarrow y \in \varepsilon(C_2)\}$$

and following definition 1 we have

$$P(e_1|e_2) = \frac{\|\varepsilon(e_1 \wedge e_2)\|}{\|\varepsilon(e_2)\|}$$

Based on assumption 2, we have

$$\begin{aligned} P(e_1|e_2) &= \frac{\|\varepsilon(C_1 \wedge C_2)\|}{\|\varepsilon(C_2)\|} \\ &= P(C_1|C_2) \end{aligned}$$

□

Here, we assume that there is a uniform distribution of instances in the instance space. Concepts can also differ in terms of their number restrictions. Theorem 3 formally states the probability of an expression e_1 given another expression, e_2 when their value restrictions and number restrictions are different.

Theorem 3 *Let* e_1 = *(:at-least* l_1 R*)* $\wedge$ *(:at-most* u_1 R*)* $\wedge$ *(:all* R C_1*) and* e_2 = *(:at-least* l_2 R*)* $\wedge$ *(:at-most* u_2 R*)* $\wedge$ *(:all* R C_2*).*
The probability of e_1 *given* e_2 *is:*

$$P(e_1|e_2) = \frac{(u_{12} - l_{12} + 1)}{(u_2 - l_2 + 1)} P(C_1|C_2)$$

where $u_{12} = min[u_1, u_2]$ $l_{12} = max[l_1, l_2]$ *and* $(u_{12} - l_{12} + 1)$ *is the intersection of* e_1 *and* e_2 *assuming that*

$$\|\{x|\langle x,y_1\rangle, \langle x,y_2\rangle, ..., \langle x,y_m\rangle\}\| = constant \times (u - l + 1) \quad (3)$$

where $m \in [l, u]$

This theorem assumes that the cardinality of the set of instances having a fixed number of fillers for role R is a constant. The underlying assumption requires a uniform distribution of instances in the instance space.

Proof Using

$$\varepsilon(E_2) = \{x|\langle x,y_1\rangle, \langle x,y_2\rangle, ..., \langle x,y_k\rangle \in \varepsilon(R) \rightarrow y_i \in \varepsilon(C_2)\}$$

and

$$\varepsilon(E_1 \wedge E_2) = \{x|\langle x,y_1\rangle, \langle x,y_2\rangle, ..., \langle x,y_j\rangle \in \varepsilon(R) \rightarrow y_i \in \varepsilon(C_1)\}$$

where $j \in [l_1, u_1] \cap [l_2, u_2] = [l_{12}, u_{12}]$ and $k \in [l_2, u_2]$ we have

$$P(T|S) = \frac{\{x|\langle x,y_1\rangle, ..., \langle x,y_j\rangle \in \varepsilon(R) \rightarrow y_i \in \varepsilon(C_1 \wedge C_2)\}}{\{x|\langle x,y_1\rangle, ..., \langle x,y_k\rangle \in \varepsilon(R) \rightarrow y_i \in \varepsilon(C_2)\}}$$

Based on assumptions 2 and 3, we have

$$P(e_1|e_2) = \frac{(u_{12} - l_{12} + 1)}{(u_2 - l_2 + 1)} P(C_1|C_2)$$

where $(u_{12} - l_{12} + 1)$ is the overlapping interval. □

Distance Measure A distance measure determines how different the instances s (in a candidate concept S) are from the target instance t. Intuitively, the distance measure provides information on how far apart the two instances are based on the meanings of their descriptions and syntax. We examine the differences between the target instance t and the source instance s by focusing on: (1) the number of different non-predefined-features, (2) the distances (differences) between values of features and (3) the differences between cardinality of a feature's values.
Given that e_1 = (:at-least l_1 R) $\wedge$ (:at-most u_1 R) $\wedge$ (:all R C_1) and e_2 = (:at-least l_2 R) $\wedge$ (:at-most u_2 R) $\wedge$ (:all R C_2).

The DM, in terms of the difference in cardinality, is

$$DM([l_1, u_1], [l_2, u_2]) = \frac{||(l_2 - l_1) + (u_2 - u_1)||}{2}$$

assuming that the number restrictions of e_1 and e_2 are disjoint so there is no overlapping between the intervals.

$$DM(e_1, e_2) = DM([l_1, u_1], [l_2, u_2]) + DM(C_1, C_2)$$

Here, we find (1) the difference between the cardinality of e_1 and e_2 and (2) the difference between value restrictions of e_1 and e_2 assuming that both e_1 and e_2 have different values for R. Developers allocate suitable values to indicate differences (1) if C_1 and C_2 are primitives and (2) if C_1 and C_2 are completely different.

Definition 2 *(Distance measure (DM))*
Let e_1 = (:at-least l_1 R_1) $\wedge$ (:at-most u_1 R_1) $\wedge$ (:all R_1 C_1) and e_2 = (:at-least l_2 R_2) $\wedge$ (:at-most u_2 R_2) $\wedge$ (:all R_2 C_2). Then the distance measure between e_1 and e_2 is

$$DM(e_1, e_2) = \alpha_{R_N} DM([l_1, u_1], [l_2, u_2]) + \alpha_{R_V} DM(C_1, C_2)$$

where $DM([l_1, u_1], [l_2, u_2]) = ||\frac{(l_2-l_1)+(u_2-u_1)}{2}||$.

The values for α_{R_N} and α_{R_V} reflect the importance of the number restriction and value restriction respectively. Software developers assign a representative value for the $DM(C_1, C_2)$ if C_1 and C_2 are primitive concepts. This definition also compares the distance apart of pairs between instances. It provides the ratio that one instance is further from t compared to another instance.

Corollary 1 *Suppose e_1 = (:at-least k R) $\wedge$ (:at-most k R) $\wedge$ (:all R C), e_2 = (:at-least l R) $\wedge$ (:at-most l R) $\wedge$ (:all R C) and e_3 = (:at-least m R) $\wedge$ (:at-most m R) $\wedge$ (:all R C) then*

$$DM(e_3, e_1) = \frac{||(m-k)||}{||(l-k)||} DM(e_2, e_1)$$

Axiom 1 *DM is symmetric:*

$$DM(T, S) = DM(S, T)$$

If $DM(T, S) = 0$ then T is equivalent to S and if $DM(T, S) = \infty$ then T and S are very dissimilar.

References

[1] E. A. Domeshek. Indexing stories as social advice. In *Proceedings of The National Conference on Artificial Intelligence AAAI-91*, pages 16 – 21. AAAI, 1991.

[2] D. C. Edelson. When should a cheetah remind you of a bat? Reminding in case-based teaching. In *Proceedings of The National Conference on Artificial Intelligence AAAI-92*, pages 667 – 672. AAAI, 1992.

[3] M. Iwayama, T. Tokunaga, and H. Tanaka. A method of calculating the measure of salience in understanding metaphors. In *Proceedings of The National Conference on Artificial Intelligence AAAI-90*, pages 298 – 303, 1990.

[4] B. P. Kettler, J. A. Hendler, W. A. Andersen, and M. P. Evett. Massively parallel support for case-based planning. In *Proceedings of the Ninth Conference on Artificial Intelligence Applications CAIA-93*, pages 3 – 9, 1994.

[5] L. Lewis. A case-based reasoning approach to the management of faults in communications and networks. In *Proceedings of the Ninth Conference on Artificial Intelligence Applications CAIA-93*, pages 114 – 120, 1994.

[6] N. A. M. Maiden. Saving reuse from the noose: Reuse of analogous specifications through human involvement in reuse process. *Information and Software Technology*, 33(10):780 –7 90, 1991.

[7] N. A. M Maiden and A. G. Sutcliffe. Analogical matching for specification reuse. In *Proceedings of Knowledge-Based Software Engineering Conference 1991*, pages 108 – 116, 1991.

[8] ISX Corporation. LOOM users guide, August 1991.

[9] E. Ostertag, J. Hendler, R. P. Diaz, and C. Braun. Computing similarity in a reuse library system: An AI-based approach. In *ACM Transactions of Software Engineering and Methodology*, volume 1, pages 205 – 228, March 1992.

[10] Swee Hor Teh, J. Yen, and William M. Lively. Heuristic modeling guidelines for expert system reuse. In *Proceedings of Florida AI Research Symposium FLAIRS*, 1994.

[11] R. S. Wall, D. Donahue, and S. Hill. The use of domain semantics for retrieval and explanation in case-based reasoning. In Janet Kolodner, editor, *Proceedings of a Workshop on Case-Based Reasoning*, pages 447 – 462, 1988.

[12] J. Yen, Swee Hor Teh, and William M. Lively. Principled modeling of the requirements and functional specifications of knowledge-based systems. In *The Seventh International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems IEA-AIE 1994 Austin*, 1993.

[13] J. Yen, Swee Hor Teh, and William M. Lively. Principled modeling and automatic classification of functional requirements for improved reusability of the design of knowledge-based systems. In *Proceedings of Conference of Artificial Intelligence Applications CAIA*, pages 599 – 600, 1994.

Useful Defaults in Description Logics

Werner Nutt
DFKI
Saarbrücken
GERMANY
nutt@dfki.uni-sb.de

Peter F. Patel-Schneider
AT&T Bell Laboratories
Murray Hill, New Jersey
U. S. A.
pfps@research.att.com

Non-monotonic constructs have recently attracted considerable attention from researchers in description logics (DLs). (Papers on the subject include [Baader and Hollunder,1992a], [Baader and Hollunder,1992b], [Padgham and Nebel,1993], [Patel-Schneider,1992], and [Quantz and Royer,1992].) This attention has been in response to a generally-felt need for simple nonmonotonic or default reasoning in many applications in which DLs can be useful. However, the proposed schemes have suffered from either lack of expressive power, or undesirable computational complexity.

We undertook to develop a DL that incorporates nonmonotonic reasoning with the following features that we feel make for a practical default extension to DLs.[1]

- The nonmonotonic features mesh well with the DL and are easy to understand.
- The entire logic makes sense from a syntactic and semantic point of view.
- The nonmonotonic features generalize those in commercial representation and programming systems, such as STROBE [Smith,1983; Smith and Carando,1986] or CLOS [Bobrow *et al.*,1988].
- The logic, and particularly its nonmonotonic features, is sufficiently expressively powerful to meet the needs of a large class of applications.
- The logic is computationally attractive, i. e., the nonmonotonic features should not unduely increase the computational complexity of reasoning in the logic. In particular, if reasoning in the underlying logic is worst-case polynomial, reasoning in the full logic should also be worst-case polynomial.

There are several alternatives for the form of the nonmonotonic constructs. We chose to use default logic rules with one free variable, as in Baader and Hollunder [1992b], because they satisfy our first two requirements. Unfortunately, reasoning in the resulting logic is computationally intractable.

One possibility is to restrict the default rules to be normal defaults, This restriction still generalizes the nonmonotonic constructs in commercial systems, which rely on nonmonotonic inheritance or input completion, both (roughly) kinds of normal defaults with specificity. Normal defaults also appear to satisfy many applications, such as the nonmonotonic constructs in the non-DL portion of PROSE [Wright *et al.*,1993], although only experience can determine if normal defaults are indeed expressive enough.

Unfortunately, the restriction to normal defaults still is computationally unattractive. The problem is that even with normal defaults there are potentially very many extensions. This suggests that restricting the number of extensions would be a profitable avenue to explore. If there is more order on the default rule applications then the number of extensions is reduced. Our idea was to extend this order to guarantee a single default extension.[2] One can argue that requiring a single extension is a good idea from the modelling point of view. Defaults are potential sources of modelling errors, and requiring a single extension makes the choices implicit in interacting defaults explicit.

Our nonmontonic extension to DLs is thus to require that the default rules be totally ordered. This results in the following logic.

Definition 1 *An* assertion *is a statement of the form* $\mathsf{IN} \in \mathsf{D}$, *where* IN *is the name of an individual and* D *is a description. A* DL knowledge base *is a finite set of assertions such that all individual names appearing anywhere in the knowledge base appear as the left-hand side of exactly one assertion.*

Definition 2 *A* DL normal default rule *is a pair of descriptions, written* $\mathsf{D}_1 \rightsquigarrow \mathsf{D}_2$. *A* DL normal default rule application, *written* $(\mathsf{D}_1 \rightsquigarrow \mathsf{D}_2, \mathsf{IN})$, *is a pair consisting of a DL normal default rule and an individual name.*

The rough intuitive meaning of a DL normal default rule is that whenever an individual is known to belong to the left-hand side of the rule (D_1) it is made an instance of the right-hand side (D_2), provided that this does not cause a contradiction.

[1]For more details see [Nutt and Patel-Schneider,1994].

[2]Brewka [1989] uses a somewhat similar technique in reverse. He defines the extensions of his preferred default theories in terms of total completions of a partial order over the defaults.

Definition 3 *A totally ordered DL normal default theory is a triple $\mathcal{T} = (\mathcal{W}, \mathcal{D}, <)$, where $\mathcal{W}$ is a DL knowledge base; $\mathcal{D}$ is a finite set of DL normal default rules; and $<$ is a total order over the default rule applications that consist of defaults and individual names in $\mathcal{W}$.*

Definition 4 *An assertion, $\mathsf{IN} \in \mathsf{D}$, follows from a knowledge base, $\mathcal{W}$, written $\mathcal{W} \models \mathsf{IN} \in \mathsf{D}$, if every interpretation that satisfies each assertion in $\mathcal{W}$ also satisfies $\mathsf{IN} \in \mathsf{D}$. The theorems of a knowledge base, $Th(\mathcal{W})$, are the assertions that follow from the knowledge base.*

Definition 5 *A default rule application, $(\mathsf{D}_1 \leadsto \mathsf{D}_2, \mathsf{IN})$ is active in a knowledge base, $\mathcal{W}$, if its prerequisite follows from $\mathcal{W}$ ($\mathcal{W} \models \mathsf{IN} \in \mathsf{D}_1$), and its consequent is both consistent with $\mathcal{W}$ ($\mathcal{W} \not\models \neg\mathsf{IN} \in \mathsf{D}_2$) and does not follow from $\mathcal{W}$ ($\mathcal{W} \not\models \mathsf{IN} \in \mathsf{D}_2$).*

Definition 6 *Let $\mathcal{T} = (\mathcal{W}, \mathcal{D}, <)$, be a totally ordered DL normal default theory. Define $\mathcal{E}_0 := \mathcal{W}$, and, for all $i \geq 0$,*

$$\mathcal{E}_{i+1} := \mathcal{E}_i \cup \{\mathsf{IN} \in \mathsf{D}_2\},$$

where $(\mathsf{D}_1 \leadsto \mathsf{D}_2, \mathsf{IN})$ is the least default rule application from $\mathcal{T}$ active in $\mathcal{E}_i$, provided there is one; otherwise $\mathcal{E}_{i+1} := \mathcal{E}_i$. Then $\mathcal{E}$ is the extension *of $\mathcal{T}$ iff $\mathcal{E} = \lim_{i \to \infty} Th(\mathcal{E}_i)$.*

Since there is exactly one extension for each totally ordered DL normal default theory, the problem of too many extensions is obviously solved. However, it might still be difficult to determine the extension of a totally ordered DL normal default theory.

Theorem 1 *Let $\mathcal{T} = (\mathcal{W}, \mathcal{D}, <)$ be a totally ordered DL normal default theory. Let $\mathcal{A}$ be the set of right-hand and left-hand sides of the default rule applications of $\mathcal{T}$. Then determining the extension of $\mathcal{T}$ takes at most*

$$2(|\mathcal{W}|.|\mathcal{D}|)^2 OF(\text{size}(\mathcal{W} \cup \mathcal{A}))$$

time, where size is a reasonable size function for knowledge bases and facts, and $OF(n)$ is the longest time it takes to determine if a fact follows from a knowledge base for facts and knowledge bases of total size $\leq n$.

Therefore, if determining whether an assertion follows from a knowledge base has polynomial complexity in the base DL, then determining the extension of a totally ordered DL normal default theory is also polynomial.

The above theorem describes a naive way of computing the extension of a totally ordered DL normal default theory, which can be considerably improved if there is a completion process for the DL, i.e., an algorithm for taking a knowledge base and computing a normal form from which determining whether an assertion holds is quick. Then the extension of a theory can be done by at most $|\mathcal{W}|.|\mathcal{D}|$ completions and $(|\mathcal{W}|.|\mathcal{D}|)^2$ queries (of knowledge bases of various sizes). If this completion process is incremental, as is roughly the case in CLASSIC [Resnick *et al.*,1992], so that the completions of $\mathcal{E}_i$ for $i \leq j$ can be done in the same amount of time as the completion of $\mathcal{E}_j$, then the time to compute the extension of a theory is the roughly time to compute the completion of the final extension, because the time for the completion process dominates the time to answer all the queries.

The definitions for totally ordered DL default theories vary considerably from the definitions of default theories with specificity given by Baader and Hollunder. There is a very close correspondence, nonetheless, in that given a DL normal default theory with specificity, for every extension of the theory, there is a totally ordered DL normal default theory with the same set of facts and default rules that has the same extension. However the total order may not be compatible with the specificity order.

How can the total order on rule applications be determined? Certainly having to specify a total order is a bother, but perhaps the total order can be derived from the knowledge base.

Nonmonotonic inheritance networks have many different strategies for determining the priorities of nonmonotonic inferences, making distinctions between on-path and off-path preemption, etc. The situation can only get worse as the expressive power of the formalism increases. Only one source of priorities is uncontroversial, that of strict rule specificity.

Strict rule specificity says that rules with more-specific prerequisites (with respect to subsumption) are to be preferred over rules with more-general prerequisites. So, if D is subsumed by D', then $(\mathsf{D} \leadsto \mathsf{R}, I)$ must be less than $(\mathsf{D} \leadsto \mathsf{R}', I)$ in the ordering of default rule applications. There appears to be no other source of ordering of default rule applications that can be derived from the knowledge base.

So where does the extra order come from? There are many possible sources for the rest of the order, including:

- a user-specified order between rules,
- a user-specified order between rule applications,
- the order of rule definition,[3] and
- the order of first definition of individuals.

Our first proposal is to use the order of definition of rules and individuals to complete the specificity order to a total order on the default rule applications. As rules and individuals can be defined in any order, this will allow many, but not all, orderings to be specified. This proposal is similar to the method of using a user-specified ordering of the parents of classes to determine which value to inherit.

Another proposal is to define a "bold" default logic. Here the system picks some total order on the default rule applications—consistent with the specificity ordering—and reasons in that extension. This may seem to be somewhat "arbitrary", but there may be many cases where the ordering doesn't matter, either because the same extension is reached or because the extensions are equivalent as far as the application is concerned. One way of reducing the arbitrariness of this method is to analyze the

[3]The order in which rules and individuals are defined is, strictly speaking, not a part of the knowledge base.

theory, either beforehand or as the system performs inferences, so as to determine whether the choices made could affect the final extension. If this condition is detected, some other agent could be queried to determine the order to be used. Doing the analysis beforehand is difficult, but an implementation using an assumption-based truth-maintenance system, where the default rule applications are the assumptions, has all the data needed to perform the run-time analysis.

We plan on implementing totally ordered defaults in CLASSIC, probably incorporating at least two of the above mechanisms for completing the order, to gather experience on the utility of totally ordered defaults in DLs. Only experience with applications can determine which, if any, method for specifying the total order over default rule applications is palatable in practice.

References

[Baader and Hollunder, 1992a] Franz Baader and Bernhard Hollunder. Embedding defaults into terminological knowledge representation formalisms. In KR-92 [1992].

[Baader and Hollunder, 1992b] Franz Baader and Bernhard Hollunder. How to prefer more specific defaults in terminological default logic. Research Report RR-92-58, German Research Center for Artificial Intelligence (DFKI), December 1992.

[Bobrow *et al.*, 1988] Daniel G. Bobrow, Linda G. DeMichiel, Richard P. Gabriel, Sonya Keene, Gregor Kiczales, and David A. Moon. Common Lisp object system specification. Xerox Palo Alto Research Centers, 1988.

[Brewka, 1989] Gerhard Brewka. Preferred subtheories: An extended logical framework for default reasoning. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 1043–1048. International Joint Committee on Artificial Intelligence, August 1989.

[KR-92, 1992] *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning*, San Mateo, California, October 1992. Morgan Kaufmann.

[Nutt and Patel-Schneider, 1994] Werner Nutt and Peter F. Patel-Schneider. Totally ordered normal defaults in description logics. Unpublished note, 1994.

[Padgham and Nebel, 1993] Lin Padgham and Bernhard Nebel. Combining classification and nonmonotonic inheritance reasoning: A first step. Unpublished note., 1993.

[Patel-Schneider, 1992] Peter F. Patel-Schneider. Defaults and descriptions. In *Working Notes, AAAI Fall Symposium Series Symposium: Issues in Description Logics: Users Meet Developers*, pages 72–73, October 1992.

[Quantz and Royer, 1992] J. Joachim Quantz and Véronique Royer. A preference semantics for defaults in terminological logics. In KR-92 [1992].

[Resnick *et al.*, 1992] Lori Alperin Resnick, Alex Borgida, Ronald J. Brachman, Deborah L. McGuinness, and Peter F. Patel-Schneider. CLASSIC description and reference manual for the COMMON LISP implementation. AI Principles Research Department, AT&T Bell Laboratories, December 1992.

[Smith and Carando, 1986] Reid G. Smith and Patricia Carando. Structured object programming in Strobe. Research Note SYS-86-26, Schlumberger-Doll Research, December 1986.

[Smith, 1983] Reid G. Smith. STROBE: Support for structured object knowledge representation. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pages 855–858. International Joint Committee on Artificial Intelligence, August 1983.

[Wright *et al.*, 1993] Jon R. Wright, Elia S. Weixelbaum, Karen Brown, Gregg T. Vesonder, Stephen R. Palmer, Jay I. Berman, and Harry H. Moore. A knowledge-based configurator that supports sales, engineering, and manufacturing at AT&T network systems. In *Proceedings of the Innovative Applications of Artificial Intelligence Conferences*, pages 183–193, Menlo Park, California, July 1993. American Association for Artificial Intelligence.

Beyond Description Logics

Robert M. MacGregor

USC/Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292-6695
macgregor@isi.edu

Since its inception, the Loom project [MacGregor 91, MacGregor&Burstein 91] has continually sought to increase the expressive power of its description logic. As the language has expanded, both the Loom syntax and the representation structures used within the Loom classifier have become increasingly resistent to future expansion of the language. In other words, there seems to be an inherent boundary in description logics as a class of languages that has little if anything to do with computational complexity.

During 1992 we began construction of a new classifier that abandons description logic in favor of a predicate calculus syntax, extended to include sets, equality, cardinality, inequalities, and predicate variables. While the predicate calculus (PC) classifier is not yet finished, it can compute most of the inferences made by the Loom classifier, as well as many that Loom cannot. We have benchmarked the performance of the PC classifier against Loom, and it is about half as fast (we expect to do better as the classifier matures). Thus, the PC classifier demonstrates the feasibility of applying description classifier technology to ordinary logic.

The PC classifier converts an arbitrary predicate calculus expression into a graph. A graph contains two kinds of nodes—a "set node" represents a concept/relation or role set or enumerated set; an "individual node" represents a variable or an individual constant. The graph edges represent logical connections and constraints that apply to the nodes. A concept C subsumes a concept D if the graph for C is embeddable in (is a subgraph of) the graph for D. The power of the subsumption test is enhanced by application of canonicalization transformations (that substitute one form of graph structure for another) and elaboration transformations (that add edges and nodes to a graph). We expect eventually to prioritize the transformations, so that definitions will be rapidly classified using "cheaper" transformations, and later on (possibly as a background task) they will be reclassified using more "expensive" transformations. There is in theory no limit to the number of normalization rules that could be incorporated into the classifier; having achieved full-expressivity, the classifier's classification capabilities can be extended in whatever direction seems appropriate. Thus far we have concentrated mainly on emulating the capabilities of description logic classifiers.

Since most description logic systems place strict limits on the kinds of definitions that users are allowed to specify, they provide little feedback on what kinds of definitions users would construct if they had more freedom of expression. The Loom description language, on the other hand, contains a constructor called "satisfies" that enables definitions to be phrased in a KIF-like version of the predicate calculus. We have observed that Loom users frequently resort to the use of "satisfies" expressions to specify definitions that cannot be phrased in Loom's description logic. This provides solid evidence that Loom's description logic is not sufficiently expressive. Unfortunately, while the Loom query processor knows how to reason with "satisfies" constructors, the classifier does not—it treats them as primitive descriptions. This creates a cognitive dissonance in the language that would not be present if Loom limited itself to a pure description logic (DL) syntax. The abandoment (by the PC classifier) of DL syntax eliminates this problem.

Unlike most other DL systems, Loom's implements a number of reasoners besides the classifier, including a query processor, a production rule system, and a temporal reasoner. All of these other reasoners adopt a predicate calculus syntax. The use of an expressive syntax for these capabilities is a necessity rather than a luxury: The Loom query processor is regarded by most of our users as being more essential than the classifier—most of this capability would be sacrificed if queries were restricted to a DL syntax. An early version of the production rule system that was limited to just the inferences computable by the classifier generated numerous complaints from users.

The substitution of the PC classifier in place of the Loom classifier will eliminate the problem of syntactic mismatch between two different sublanguages, and it will enable Loom (actually, its successor) to compute subsumption relationships between descriptions phrased in the full predicate calculus. The absence of an upper bound on expressivity will enable us to experiment with new strategies of deductive reasoning. For example, the new classifier architecture will be much more compatible than the old one with a constraint propagation subsystem (modelled after McAllester's SCREAMER system [McAllester&Siskind 93]) that we are beginning to implement.

While we remain convinced of the utility of the classifier as a deductive engine, we have come to view description logic syntax as an impediment rather than an asset (see [Doyle&Patil 91]). In our view, the important representational idea within DL is its focus on "intensional sets" as key components within a definition. DL systems are more fluent that most other KR technologies at reasoning with sets. However, this capability is not tied to DLs—any system that includes intensional sets as a fundamental part of its representation can duplicate the inferences made by a DL classifier.

Below, we illustrate the syntax for the definitions recognized by the PC classifier. We briefly discuss how the PC classifier organizes descriptions internally, and how it computes subsumption relationships ([MacGregor 94] contains more details). Finally, we return to the question of DLs versus ordinary logic.

The first example in Table 1 defines the concept "Fielded-Soccer-Team" using Loom's DL syntax. The second example shows the same definition phrased in the syntax recognized by the PC classifier. The classifier parses this syntax into internal structures represent a subset of the predicate calculus (but which are representationally complete). A "description" in the PC classifier is represented as a set expression "(setof (<variables>) <definition>)", where <definition> is a sentence that defines a membership test for that set. A description that has a name associated with it is called a "relation".

The third example in Table 1 repeats the "Fielded-Soccer-Team" definition, rephrased to correspond to the graph structures that the PC classifier builds to represent "Fielded-Soccer-Team" internally. Our graph defines three descriptions, two of which are nested within the third. Each of the two nested descriptions is represented by a sentence containing a free variable. We call the sets represented by these descriptions "indexical", since their meaning is dependent on the context in which they are syntactically situated. An indexical set corresponds to the KL-ONE [Brachman&Schmolze 85] notion of a "roleset" (except that our indexical sets are not limited to a single free variable). In our graph structures, all variables are implicity existentially quantified—expressions containing universal quantifiers are converted to set-containment relationships. For example, the "implies" constraint in our example has been translated into a "contained-in" relationship between an indexical set and and the relation "In-the-Game".

Loom Definition:

```
(defconcept Fielded-Soccer-Team
   :is-primitive (and Soccer-Team
                      (atleast 7 player) (atmost 11 player)
                      (exactly 1 goalie) (all player In-the-Game)))
```

PC Classifier Definition:

```
(defconcept Fielded-Soccer-Team (?t)
  :def (and (Soccer-Team ?t)
            (exists (?pnum)
               (and (= ?pnum (cardinality (setof (?p) (player ?t ?p))))
                    (>= ?pnum 7) (<= ?pnum 11)
                    (forall (?p) (implies (player ?t ?p) (In-the-Game ?p)))
                    (= (cardinality (setof (?g) (goalie ?t ?p))) 1)))))
```

PC Classifier Internal Representation:

```
(setof (?t) (and (Soccer-Team ?t)
                 (exists (?set1 ?set2 ?pnum)
                    (and (= ?set1 (setof (?p) (player ?t ?p)))
                         (= ?set2 (setof (?g) (goalie ?t ?p)))
                         (cardinality ?set1 ?pnum)
                         (>= ?pnum 7) (<= ?pnum 11)
                         (contained-in ?set1 (the-relation In-the-Game 1))
                         (cardinality ?set2 2)))))
```

Definition using Sugar-coated Syntax:

```
(defconcept Fielded-Soccer-Team (?t)
  :def (and (Soccer-Team ?t)
            (role-min ?t player 7)
            (role-max ?t player 11)
            (role-type ?t player In-the-Game)
            (role-card ?t goalie 1)))
```

Table 1

The PC classifier uses a structural subsumption test. The test succeeds when a mapping can be found from variables, constants and descriptions in one description onto corresponding components in the other description such that all constraints are satisfied. Within the test, the indexical sets are treated analogously to the way role expressions are manipulated by most DL classifiers. Given relations A and B, to determine if A subsumes B, a part of the subsumption test checks that each indexical set in A maps to an equivalent one in B, and that constraints such as cardinality bounds and containment relationships on a nested set in A are also present on the corresponding nested set in B.

```
(defrelation <name> (<domain variables>)
    [:def | :iff-def] <definition>)
```

Singleton indexical sets that are non-empty are replaced during the canonicalization process by individual variables. For example, if "head-coach" is a functional (single-valued) relation, then the description "Team with a head-coach":

```
(setof (?t) (and
   (Team ?t)
   (exists (?s1) (and
      (= ?s1 (setof (?c) (head-coach ?t ?c)))
      (>= (cardinality ?s1) 1)))))
```

is converted to

```
(setof (?t) (and
   (Team ?t)
   (exists (?c) (head-coach ?t ?c))))
```

Both of these expressions correspond to the DL expression

```
(and Team (atleast 1 head-coach))
```

To obtain more complete reasoning, it is sometimes necessary to represent both a singleton indexical set and the individual variable. In this case, the variable gets classified under the nested set. An equivalent but more elaborate description of "Team with a head-coach" is

```
(setof (?t) (and
   (Team ?t)
   (exists (?s1 ?c) (and
      (= ?s1 (setof (?c2)
                (head-coach ?t ?c2)))
      (>= (cardinality ?s1) 1)
      (head-coach ?t ?c)
      (member-of ?c ?s1)))))
```

The PC classifier recognizes descriptions phrased using second-order syntax. For example, here is the definition for the relation "role-min":

```
(defrelation role-min (?vbl ?rln ?min)
  :iff-def
    (>= (cardinality
           (setof (?filler)
               (?rln ?vbl ?filler)))
         ?min))
```

Allowing for second order relations makes it very easy to extend the language syntax. The last example in Table 1 shows how the use of the second-order relations "role-min", "role-max", "role-type", and "role-card" can be used to regain much of the conciseness that is otherwise lost when converting DL expressions into PC expressions. It turns out to be relatively easy to introduce second order expressions into the classifier's internal representation—the representation must allow variables to denote sets/relations as well individuals.

Many users have the impression that the boundary delineated by a DL language includes descriptions that are easy to classify and excludes those that aren't. In fact, its easy to find examples of descriptions that can be classified quite easily, but don't fit into DL syntax. For example, consider the definitions in Table 2. The PC classifier finds the subsumption relationship between One-of-the-Five-Highest-Scoring-Teams and Third-Highest-Scoring-Team without even breathing hard. Only a slight bit more work is needed to determine the subsumption relation between At-Least-One-Forward and More-Forwards-Than-Halfbacks.

None of the DLs that we know of can express either of the "Highest-Scoring-Team" relations, nor can they define the relation "More-Forwards-Than-Halfbacks". We have come to regard description logic syntax as a straight-jacket that places unnecessary limitations on a user's ability to model a domain and that excludes many useful inferences that fall naturally within the scope of a description classifier.

We expect that the practice of constructing ontologies and domain models around a core of definitional knowledge will become increasingly wide-spread. We also expect that description classifier technology will be incorporated into an increasing percentage of KR systems, as users learn to appreciate the benefits, such as organizing definitions and uncovering inconsistencies, that classifiers make possible [MacGregor 90]. However, the PC classifier demonstrates that these benefits can achieved without the use of a description logic.

```
(defrelation One-of-the-Five-Highest-Scoring-Teams (?t)
    :iff-def (and (Soccer-Team ?t)
                  (>= 5 (cardinality (setof (?hst)
                                             (and (Soccer-Team ?hst)
                                                  (>= (goals-scored ?hst)
                                                      (goals-scored ?t))))))))

(defrelation Third-Highest-Scoring-Team (?t)
    :iff-def (and (Soccer-Team ?t)
                  (= 3 (cardinality (setof (?hst)
                                            (and (Soccer-Team ?hst)
                                                 (>= (goals-scored ?hst)
                                                     (goals-scored ?t))))))))

(defrelation At-Least-One-Forward (?t)
    :iff-def (and (Soccer-Team ?t)
                  (>= (cardinality (setof (?f) (forward ?t ?f))) 1))))

(defrelation More-Forwards-Than-Halfbacks (?t)
    :iff-def (and (Soccer-Team ?t)
                  (> (cardinality (setof (?f) (forward ?t ?f)))
                     (cardinality (setof (?h) (halfback ?t ?h))))))
```

Table 2

References

[Brachman&Schmolze 85] Brachman, R.J. and Schmolze, J.G.", "An Overview of the KL-ONE Knowledge Representation System", Cognitive Science, August, 1985, pp. 171-216.

[Doyle&Patil 91] Jon Doyle and Ramesh Patil, "Two Theses of Knowledge Representation: Language Restrictions, Taxonomic Classification, and the Utility of Representation Services", Artificial Intelligence, 48, 1991, pp.261-297.

[MacGregor 90] Robert MacGregor, "The Evolving Technology of Classification-based Knowledge Representation Systems", Principles of Semantic Networks: Explorations in the Representation of Knowledge, Chapter 13, John Sowa, Ed., Morgan-Kaufman, 1990.

[MacGregor 91] Robert MacGregor, "Using a Description Classifier to Enhance Deductive Inference", Proc. Seventh IEEE Conference on AI Applications, Miami, Florida, February, 1991, pp 141-147.

[MacGregor&Burstein 91] Robert MacGregor and Mark H. Burstein, "Using a Description Classifier to Enhance Knowledge Representation", IEEE Expert, 6:3, June, 1991, pp.41-46.

[MacGregor 94] Robert MacGregor, "A Description Classifier for the Predicate Calculus", submitted to AAAI-94, The National Conference on Artificial Intelligence, 1994.

[McAllester&Siskind 93] Jeffrey M. Siskind and David A. McAllester, "Nondeterministic Lisp as a Substrate for Constraint Logic Programming", AAAI-93 Proc. of the Eleventh Nat'l Conf. on Artificial Intelligence, Washington, DC, pp.133-138.

A Relation-Based Description Logic

Enrico Franconi and **Vania Rabito**
Knowledge Representation and Reasoning Lab.
IRST, I-38050 Povo TN, Italy
{franconi, rabito}@irst.it

1 Introduction

In this paper, we will show the basic ideas for a possible formalization[1] of KODIAK, a relation-based description logic, first informally introduced by Robert Wilensky in [Wilensky,1987]. His work was motivated by particular requirements of natural language applications, specially regarding lexical semantic discrimination: whenever the syntactical parser tries to build a constituent, the consistency of the semantic part of such a constituent is checked, and a minimal semantic representation is computed [Norvig,1987; Jacobs,1992; Bayer and Vilain,1991]. In [Wilensky,1987] several drawbacks of KL-ONE-based languages were highlighted. The main comment is about the limited expressive power of KL-ONE-based languages to represent *relational entities*. They are represented as *roles*, which are second-class citizens:

- Role restrictions do not get defined. E.g., PAINTER is defined via a relation paints restricted to PAINTING; however, the concept of PAINTING is left primitive. Inverse roles solve this problem; however, cyclic definitions are needed.
- Same relational entities need to be defined twice, as roles – in order to define concepts they are related to –and as concepts – in order to better define the relational entities themselves. However, no connection exists between the relation as role and the relation as concept.

The paper is organized as follows. After an informal introduction to KODIAK, we describe $\mathcal{RDL}$, a formalization of KODIAK. Tractable reasoning procedures for $\mathcal{RDL}$ are briefly revealed in section 4. In the last part of the paper, first a solution to the *Reified Relation* and *QUA-link* problems in the context of KL-ONE-based description logics is presented. Then, we show how this solution can be used to map (a subset) of $\mathcal{RDL}$ in KL-ONE.

2 KODIAK

The observations of the previous section lead us to consider a formalization of KODIAK, to be eventually compared with the expressivity of a KL-ONE-based description logic. KODIAK is a description language not belonging to the KL-ONE family. It is an attempt to reconstruct a description language from a different point of view.

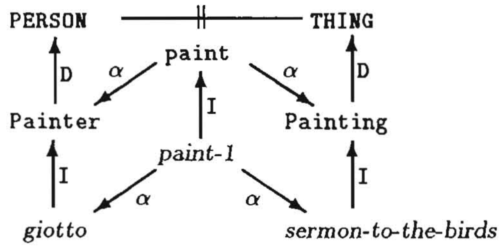

Figure 1: A sample KODIAK knowledge base.

KODIAK is a relation-based representation language, as opposed to KL-ONE, which is a concept-based representation language. We consider here only a simplified version of the language. The syntactic types of the language are *Relations*, *Aspectuals*, *Absolutes*; the basic operators of the language are *Manifest*, *Dominate*, *Instantiate*, *Disjointness*. Figure 1 shows an example. The relation paint manifests the two aspectuals Painter and Painting which are dominated by the mutually disjoint absolutes PERSON and THING; *paint-1* is a relation instantiating paint and manifesting the instances *giotto* and *sermon-to-the-birds*. It is important to notice that this schema defines completely all the mentioned entities. The aspectual Painter *is defined* as being the first participant of the relation paint, whereas the relation Paint is defined as having two participants, namely Painter and Painting. On the other hand, the absolute PERSON is defined only as being disjoint from the absolute THING.

3 The Formal Language $\mathcal{RDL}$

We introduce in this section a very preliminary formalization of KODIAK, called $\mathcal{RDL}$. A $\mathcal{RDL}$ knowledge base is a collection of statements of the following form:

[1]This work started from a discussion with Bernhard Nebel.

DOMINATE (C_1, C_2)	C_1, C_2 with same arity
MANIFEST $(R, C_1 \ldots C_n)$	$n \geq 1$
DISJOINT $(C_1 \ldots C_m)$	$m \geq 2$; C_i with same arity
C $(a_1 \ldots a_l)$	$l \geq 1$

where 'C' is a syntactic object of the sort *concept*, 'R' is a syntactic object of the sort *relation* and 'a' is a syntactic object of sort *individual*; the sort *relation* is a sub-sort of the sort *concept*, i.e. any relation is a concept. A relation is associated to its arity, which is defined by the number of the other concepts appearing in the MANIFEST statement in which they appear as the first argument. Non relational concepts can be also seen as relations of zero arity.

To give a semantics to $\mathcal{RDL}$, let us associate to each relation of arity n a predicate symbol of arity $n+1$, and to each individual a constant symbol in a classical logic. For sake of simplicity, syntactic objects in $\mathcal{RDL}$ and the corresponding semantic symbols in the logic will have the same representation, being clear from the context their meaning. Relations should satisfy the following axioms:

$$\forall R.\, \forall i.\, \exists \bar{x}.\, R(i, \bar{x})$$
$$\forall R_1, R_2. \forall i. \forall \bar{x}, \bar{y}.\, (R_1(i, \bar{x}) \wedge R_2(i, \bar{y})) \rightarrow (\bar{x} = \bar{y})$$

These axioms state a kind of correspondence between the first argument of the predicate and the tuple formed by the rest of the arguments. The first argument – called *index* – represents in some sense the particular instance of the relation, whereas the rest of the arguments represent the instances of the fillers of the relation. Given an index for a relation, the corresponding fillers of the relation are uniquely determined; however, a tuple of fillers of a relation may correspond to more than one index. Our knowledge base is so rewritten in $\mathcal{RDL}$:

DOMINATE (PERSON, Painter)
DOMINATE (THING, Painting)
DISJOINT (PERSON, THING)
MANIFEST (paint, Painter, Painting)
paint (*paint-1*, *giotto*, *sermon-to-the-birds*)
Painter (*giotto*)
Painting (*sermon-to-the-birds*).

Please note that the last two statements are redundant, i.e. they can be deduced from the former statements.

The semantics of the operators is given as a mapping in the classical logic and it as follows[2]:

- DOMINATE (C_1, C_2) iff $\forall i, \bar{x}.\ C_2(i, \bar{x}) \rightarrow C_1(i, \bar{x})$
- MANIFEST $(R, C_1, \ldots, C_n)$ iff
  $\forall x_1 \ldots x_n \exists i.\ R(i, x_1 \ldots x_n) \rightarrow C_1(x_1) \wedge \ldots \wedge C_n(x_n)$
  $\forall x_1.\, C_1(x_1) \rightarrow \exists i, x_2 \ldots x_n.\, R(i, x_1 \ldots x_n)$
  $\vdots$
  $\forall x_n.\, C_n(x_n) \rightarrow \exists i, x_1 \ldots x_{n-1}.\, R(i, x_1 \ldots x_n)$
- DISJOINT $(C_1 \ldots C_n)$ iff
  DISJOINT (C_i, C_j) for all $i, j = 1 \ldots n$ and $i \neq j$
- DISJOINT (C_1, C_2) iff $\forall i, \bar{x}. \neg(C_1(i, \bar{x}) \wedge C_2(i, \bar{x}))$
- C $(a_1 \ldots a_l)$ iff $C(a_1 \ldots a_l)$

[2]In the following, for sake of simplicity it is intended that in the case of non relational concepts, all the arguments of the corresponding predicate – but the first – should be omitted, i.e. $C(i, a_1, \ldots, a_n)$ becomes $C(i)$.

In $\mathcal{RDL}$ relations can be the arguments of other relations; for example, the following is a knowledge base expressing "Vania is watching at Giotto painting the *Sermon to the birds*":

MANIFEST (see, Experiencer, experience)
DOMINATE (experience, paint)
see (*see-1*, *vania*, *paint-1*)
paint (*paint-1*, *giotto*, *sermon-to-the-birds*)

4 Reasoning in $\mathcal{RDL}$

We have defined in $\mathcal{RDL}$ as usual three reasoning services: satisfiability, subsumption between concepts of the same arity and instance checking. We have found sound and complete algorithms for these services, running in polynomial time [Rabito,1994]. Figure 2 (a) shows a sound and complete set of inference rules for the derivation of disjointness and subsumption. This set of rules is inspired by the work of [Atzeni and Parker Jr.,1986]. The inference rules presented in the whole figure 2 (a) + (b) are a sound and complete set of inference rules also for the derivation of ground instances.

A $\mathcal{RDL}$ knowledge base Σ is satisfiable (or consistent) if and only if none of the following conditions holds:

- $\Sigma \vdash$ DISJOINT(C, C)
- $\Sigma \vdash C(i, a_1, \ldots, a_n),\ D(i, a_1, \ldots, a_n)$, DISJOINT$(C, D)$
- $\Sigma \vdash$ DISJOINT(C_i, D_j), MANIFEST$(R, C_1 \ldots C_n)$, MANIFEST$(S, D_1 \ldots D_m)$, $R(i_1, a_1, \ldots, a_n)$, $S(i_2, b_1, \ldots, b_m)$, $a_i = b_j$ for some i, j, $1 \leq i \leq n$, $1 \leq j \leq m$

A concept C subsumes a concept D with respect to a $\mathcal{RDL}$ knowledge base if and only if a statement of the form $C \sqsupseteq D$ can be derived by applying the rules to the knowledge base augmented with the statements defining C and D.

Given a $\mathcal{RDL}$ knowledge base Σ, an ordered set of individuals $i, a_1, \ldots, a_n$ and a concept C, the *instance checking problem* is to test whether $\Sigma \models C(i, a_1, \ldots, a_n)$. Informally, it is the inference task checking whether the individual i is an instance of the concept C and, if C is a relational concept, whether $a_1, \ldots, a_n$ are instances of the fillers of the relation itself. $\Sigma \models C(i, a_1, \ldots, a_n)$ holds if and only if $C(i, a_1, \ldots, a_n)$ can be derived by applying the rules.

In [Rabito,1994] an extension of the language with other operators (e.g. equality, disjointness in context)

(a)

- $\vdash C \sqsupseteq C$
- $\textsc{dominate}(C_1, C_2) \vdash C_1 \sqsupseteq C_2$
- $C_1 \sqsupseteq C_2,\ C_2 \sqsupseteq C_3 \vdash C_1 \sqsupseteq C_3$
- $R \sqsupseteq S,\ \textsc{manifest}(R, C_1 \ldots C_n),$
  $\textsc{manifest}(S, D_1 \ldots D_n) \vdash C_i \sqsupseteq D_i$
  for all $i = 1, \ldots, n$
- $\textsc{disjoint}(C_1 \ldots C_n) \vdash \textsc{disjoint}(C_i, C_j)$
  for all $i, j = 1, \ldots, n$ and $i \neq j$
- $\textsc{disjoint}(C_1, C_1) \vdash \textsc{disjoint}(C_1, C_2)$
- $\textsc{disjoint}(C_1, C_2),\ C_2 \sqsupseteq C_3 \vdash \textsc{disjoint}(C_1, C_3)$
- $\textsc{disjoint}(C_1, C_1) \vdash C_2 \sqsupseteq C_1$
- $\textsc{manifest}(R, C_1 \ldots C_n),\ \textsc{manifest}(S, D_1 \ldots D_n),$
  $\textsc{disjoint}(C_i, D_i) \vdash \textsc{disjoint}(R, S)$
  for some $i, 1 \leq i \leq n$
- $\textsc{manifest}(R, C_1 \ldots C_n),\ \textsc{disjoint}(R, R) \vdash$
  $\textsc{disjoint}(C_i, C_i)$
  for all $i = 1, \ldots, n$

(b)

- $C_1(i, a_1, \ldots, a_n),\ C_2 \sqsupseteq C_1 \vdash C_2(i, a_1, \ldots, a_n)$
- $\textsc{manifest}(R, C_1 \ldots C_{j-1},\ C\ , C_{j+1} \ldots C_n),$
  $R(j, b_1 \ldots b_{j-1},\ i\ , b_{j+1} \ldots b_n),\ D(i, a_1 \ldots a_n) \vdash$
  $C(i, a_1, \ldots, a_n)$

Figure 2: Inference rules for the derivation of disjointness, subsumption and ground instances.

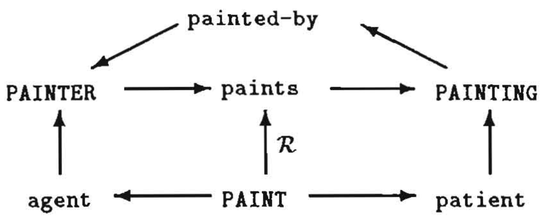

Figure 3: Reification in KL-ONE

and services (e.g. *inheritance* and subsumption between concepts with different arity, concretion [Jacobs,1992; Bayer and Vilain,1991; Vilain,1993]) is presented.

5 Reified Relations and QUA-links in KL-ONE

In this section we introduce the *Reified Relation* and *QUA-link* problems in the context of KL-ONE-based description logics.

Reification is an operator which allows for both relation-as-role and relation-as-concept interpretations of a binary relation. Formal semantics for reification can be given by the following axiom:

$$\forall x, y.\ R(x, y) \Leftrightarrow \exists z.\ \hat{R}(z) \wedge \Delta(z) = x \wedge \Theta(z) = y$$

where $\hat{R}$ is the reified relation of R, Δ is a function mapping to the domain of the relation and Θ is a function mapping to the range. This operation is crucial for NL applications, e.g. to represent that a noun phrase modified by a prepositional phrase can have the same meaning of the NP modified by a relative phrase. This is the case if the interpretation of the preposition is the same as the one of the relative verb. It is assumed, as it is usual, that prepositions are mapped to roles and verbs to concepts. For example the following sentences should have the same meaning:

Show me a fresco of *Giotto's.*

Show me a fresco painted by *Giotto.*

In this example – see figure 3 – the concept `PAINT` associated to the verb "paint" reifies the role `paints` associated to the preposition "of", the Δ functional role corresponds to the `agent` role of the verb, and the Θ functional role corresponds to the `patient` role of the verb.

Informal or incomplete solutions to the Reified Relation problem have been presented in [Bateman *et al.*,1990; Kobsa,1991; MacGregor,1993].

At the 1981 KL-ONE workshop [Schmolze and Brachman,1982] the QUA-link was informally introduced. The QUA-link was defined as

> a kind of inheritance cable pointing from the concept being defined to the *RoleSet* of another concept taken as its definitional context. From a purely syntactic point of view, the QUA-defined concept was considered *subC* to the *V/R* of the *RoleSet* it pointed to.

Formal semantics capturing the idea of the QUA-link is given by the following axiom[3]:

$$(\bowtie C\, R.\, D)^{\mathcal{I}} = \{x \mid x \in D^{\mathcal{I}} \wedge \exists y.\, y \in C^{\mathcal{I}} \wedge \langle y, x\rangle \in R^{\mathcal{I}}\}$$

As an example, consider how to represent the PAINT relation, using the QUA operator:

$$\begin{array}{lcl} \mathtt{PAINT} & \dot\sqsubseteq & \exists\mathtt{agent} \sqcap \exists\mathtt{patient} \sqcap \\ & & \leq 1\,\mathtt{agent} \sqcap \leq 1\,\mathtt{patient} \\ \mathtt{PAINTER} & \doteq & \bowtie \mathtt{PAINT}\,\mathtt{agent.PERSON} \\ \mathtt{PAINTING} & \doteq & \bowtie \mathtt{PAINT}\,\mathtt{patient.THING} \end{array}$$

The latter definitions are equivalent to the following definitions:

$$\begin{array}{lcl} \mathtt{PAINTER} & \doteq & \mathtt{PERSON} \sqcap \exists\mathtt{agent}^{-1}.\mathtt{PAINT} \\ \mathtt{PAINTING} & \doteq & \mathtt{THING} \sqcap \exists\mathtt{patient}^{-1}.\mathtt{PAINT} \end{array}$$

It is worth noting that the latter definitions do not involve terminological cycles, as one would expect[4]. In fact, a naive translation would include cycles, since PAINTER is being defined as being the agent of PAINT and, at the same time, PAINT is being defined as having a PAINTER as agent. So, we do not need the full expressive power of Converse Deterministic PDL, as it is shown in [De Giacomo and Lenzerini,1994] in order to represent correctly n-ary relations in a description logic[5]. In fact, this definition can be expressed in the polynomial description logic of [Buchheit *et al.*,1994a; Buchheit *et al.*,1994b].

In this context, the reified relation problem is exactly solved by means of the following equations:

$$\begin{array}{lcl} \hat{\mathtt{R}} & \dot\sqsubseteq & \exists\Delta \sqcap \exists\Theta \sqcap \leq 1\,\Delta \sqcap \leq 1\,\Theta \\ \mathtt{C}_\Delta & \doteq & \exists\Delta^{-1}.\hat{\mathtt{R}} \\ \mathtt{C}_\Theta & \doteq & \exists\Theta^{-1}.\hat{\mathtt{R}} \\ \mathtt{R} & \doteq & (\Delta^{-1}|\hat{\mathtt{R}}) \circ \Theta \end{array}$$

These equations can be generalized to the n-ary case:

$$\begin{array}{lcl} \hat{\mathtt{R}} & \dot\sqsubseteq & \exists\mathtt{a}_1 \sqcap \ldots \sqcap \exists\mathtt{a}_n \sqcap \leq 1\,\mathtt{a}_1 \sqcap \ldots \sqcap \leq 1\,\mathtt{a}_n \\ \mathtt{C}_\mathtt{i} & \doteq & \exists\mathtt{a}_\mathtt{i}^{-1}.\hat{\mathtt{R}} \\ \mathtt{R}_{\mathtt{ij}} & \doteq & (\mathtt{a}_\mathtt{i}^{-1}|\hat{\mathtt{R}}) \circ \mathtt{a}_\mathtt{j} \end{array}$$

It can be shown that $\mathcal{RDL}$ knowledge bases without disjointness can be mapped in KL-ONE knowledge bases using the equations cited above. However, even if this mapping maintains the tractability of the reasoning procedures, it does not consider disjointness, whose complexity is still polynomial in $\mathcal{RDL}$, but it is not yet known in KL-ONE.

[3]This was discovered after a discussion with Fabrizio Sebastiani.

[4]This was suggested by Werner Nutt.

[5]The goal of [De Giacomo and Lenzerini,1994] is to model n-ary relations: a n-ary relation denotes a set of tuples, whereas a relation defined with a QUA-link denotes a multi-set of tuples. In the latter case, we speak of *instances* of relations.

The comparison between $\mathcal{RDL}$ and KL-ONE should also take into account the *Entity-Relationship* (ER) semantic data models framework [Calvanese *et al.*,1994] and the Propositional Knowledge Representation framework [D'Aloisi and Castelfranchi,1993].

References

[Atzeni and Parker Jr., 1986] Paolo Atzeni and D. Stott Parker Jr. Formal properties of net-based knowledge representation schemes. In *Proc. of the 2nd Conference on Data Engineering*, Los Angeles, CA, February 1986. A long version appears as Technical Report CNR-IASI 135, Rome, Italy.

[Bateman *et al.*, 1990] John A. Bateman, Robert T. Kasper, Johanna D. Moore, and Richard A. Whitney. A general organization of knowledge for natural language processing: the PENMAN upper model. Technical report, USC/Information Science Institute, Marina del Rey CA, March 1990.

[Bayer and Vilain, 1991] Samuel Bayer and Marc Vilain. The relation-based knowledge representation of King-Kong. In *Working Notes of the AAAI Spring Symposium "Implemented Knowledge Representation and Reasoning Systems"*, pages 31–44, March 1991.

[Buchheit *et al.*, 1994a] M. Buchheit, F. M. Donini, W. Nutt, and A. Schaerf. Refining the structure of terminological systems: Terminology = schema + views. In *Proc. of AAAI-94*, 1994. To appear.

[Buchheit *et al.*, 1994b] Martin Buchheit, Manfred A. Jeusfeld, Werner Nutt, and Martin Staudt. Subsumption between queries to object-oriented databases. *Information Systems*, 19(1):33–54, 1994.

[Calvanese *et al.*, 1994] Diego Calvanese, Maurizio Lenzerini, and Daniele Nardi. A unified framework for class-based representation formalisms. In *Proc. of KR-94*, Bonn D, May 1994.

[D'Aloisi and Castelfranchi, 1993] Daniela D'Aloisi and Cristiano Castelfranchi. Propositional and terminological knowledge representation. *JETAI*, 5(2-3):149–166, 1993.

[De Giacomo and Lenzerini, 1994] Giuseppe De Giacomo and Maurizio Lenzerini. Description logics with inverse roles, functional restrictions, and n-ary relations. In *Proc. of the 4th European Workshop on Logics in AI, JELIA-94*, September 1994.

[Jacobs, 1992] Paul S. Jacobs. TRUMP: A transportable language understanding program. *International Journal of Intelligent Systems*, 7:245–276, March 1992.

[Kobsa, 1991] Alfred Kobsa. Reification in SB-ONE. In *International Workshop on Terminological Logics*, DFKI Document D-91-13, pages 72–74, May 1991.

[MacGregor, 1993] Robert M. MacGregor. Representing reified relations in LOOM. *Journal of Experimental and Theoretical Artificial Intelligence*, 5:179–183, 1993.

[Norvig, 1987] Peter Norvig. Unified theory of inference for text understanding. UCB/CSD Report 87/339, University of California, Berkeley CA, January 1987.

[Rabito, 1994] Vania Rabito. Studio di un linguaggio per la rappresentazione della conoscenza basato su relazioni. Master's thesis, University of Milan, Italy, July 1994. In italian.

[Schmolze and Brachman, 1982] James G. Schmolze and Ronald J. Brachman, editors. *Proceedings of the 1981 KL-ONE Workshop*, Fairchild Technical Report No. 618, May 1982.

[Vilain, 1993] Marc Vilain. Validation of terminological inference in an information extraction task. In *Proc. of the 1993 ARPA Human Language Workshop*. Morgan Kaufman, 1993.

[Wilensky, 1987] Robert Wilensky. Some problems and proposals for knowledge representation. UCB/CSD Report 87/351, University of California, Berkeley, CA, May 1987.

Studies about the Integration of Classification-Based Reasoning and Object-Oriented Programming

Amedeo Napoli
CRIN CNRS - INRIA Lorraine
B.P. 239 - 54506 Vandœuvre-lès-Nancy Cedex - France
Phone: (33) 83 59 20 68 - Fax: (33) 83 41 30 79
(e-mail: napoli@loria.fr)

1 Introduction

The general notion of "object" has led to several types of object-based representation systems, such as classification-based systems [MacGregor,1991], object-oriented systems [Masini *et al.*,1991], type-based systems [Aït-Kaci,1986] and conceptual graphs [Sowa,1984]. In the following, we will be concerned with classification-based and object-oriented systems. The family of classification-based systems mainly includes terminological logics. It aims at formalizing the description of real-world concepts, the subsumption relation between concept descriptions and classification-based reasoning. The family of object-oriented systems mainly includes prototype-based and class-based systems. It aims at formalizing typical real-world objects, nonmonotonic inheritance[1] object-oriented programming. As argued in [Karp,1993], the use of classification can be seen as a basic difference between the two families. Although these two families rely on seemingly different representation schemes, it would be interesting to combine their capabilities: adding the classification ability in object-oriented systems, adding forms of nonmonotonic property sharing and forms of (logical) programming in terminological logics [Doyle and Patil,1991].

Among the number of proposals combining nonmonotonic and classification-based reasoning, we can identify two important approaches. In the first, default logic is integrated into terminological logics [Baader and Hollunder,1992] [Baader and Hollunder,1993] [Padgham and Zhang,1993] [Padgham and Nebel,1993] [Quantz and Royer,1992]. The resulting logic can deal with special forms of nonmonotonic reasoning, but lacks the full power of nonmonotonic inheritance, e.g. overridding and exceptions, as well as programming capabilities. In the second approach, facilities to define conceptual descriptions and a classifier are provided for an object-oriented environment [Rathke,1993] [Yelland,1992]. In the resulting systems, the status and the use of concepts is unclear: are concepts real objects, what is the semantics associated with concepts and how concepts can be used within object-oriented programming?

[1]The so-called *nonmonotonic inheritance networks* [Horty *et al.*,1990] are related with the formal analysis of inheritance and exceptions rather than with representational purposes.

The integration of classification-based technology into object-oriented environments can be considered on another basis. This third approach is based on a specific object-based subsumption relation that is used to compare the intension of objects, as recommended in [Woods,1991]. In this way, classification-based reasoning becomes an integrated tool being on the same level as inheritance and procedural-based reasoning [Karp,1993] [Napoli,1992].

2 Steps Towards Integration

An object-oriented system allows one to represented real-world knowledge such as concepts, individuals and relations between concepts, in a hierarchy of objects. Generic objects are used to describe concepts, and instances of generic objects describe specific individuals. Generic objects can be classes, e.g. CLOS [Keene,1989], or three level structures frame-slot-facet, e.g. Y3 [Ducournau,1989]. A generic object encapsulates a collection of structural and behavioral properties, i.e. data and procedures acting on these data. The set of generic objects is organized in an inheritance hierarchy, where specific objects are defined as specializations of general objects. Every generic object inherits the properties of one or more ascendants. Instances usually depend on one single generic object, and they are the entities effectively handled in the applications by message sending. In the following, we consider that generic objects are frame-slot-facet structures, where slots can be attributes or methods (*hybrid systems* in [Masini *et al.*,1991]). Thus, both access-oriented and object-oriented programming are allowed in such object-oriented environments.

Relying on a logical basis rather than on an object-oriented basis, terminological logics or description logics [Nebel,1990] are used to reason about concepts describing sets of elements called individuals or instances. Concepts and individuals are related through binary relations called roles. Defined concepts are composite terms built from primitive concepts and restrictions on roles. Basic information such as the kinds or number of role fillers can be associated with roles. The subsumption relation is a partial ordering used to organize concepts and their instances in a hierarchy, according to a set-theoretical interpretation. Reasoning is based on classification, that can be used to handle concepts and individuals and to control the evolution and the truth maintenance of the hierarchy.

Most of the time, inheritance is the primary, most powerful representation primitive in object-oriented systems. But inheritance is a mechanism for knowledge sharing that can be deductively weak [Patel-Schneider,1990]. By contrast, subsumption and classification are powerful inference mechanisms, but terminological logics lack the procedural component that can be needed in the modeling of real-world applications. One basis for integration is then to associate a subsumption-like relation and classification-based techniques in an object-oriented environment. The resulting system, called *integrated object-based representation system*, has the following features:

- Descriptions: an object is defined by a state (definitional attributes) and a behavior (methods and/or reflexes). The set of objects is organized within a hierarchy including both generic objects and instances.
- Manipulations: objects can be accessed through object-oriented programming (methods) or access-oriented programming (reflexes). Reasoning relies on inheritance (property sharing), classification and truth maintenance.

In the following, we introduce and comment on the separation between *definitional* and non definitional slots. Definitional slots or attributes can be seen as necessary and sufficient characteristics associated with generic objects. Attached to a definitional attribute are *definitional properties*, e.g. a domain, a range, a cardinality and a value. Non definitional slots are related to implementation purposes and pieces of code, e.g. methods and reflexes.

An attribute $a1$ *subsumes* an attribute $a2$ if and only if (1) $a1$ and $a2$ have the same name, (2) $a2$ has at least the same definitional properties as $a1$ and additional definitional properties, or (2') $a2$ has more specialized definitional properties than $a1$ and possibly additional definitional properties. For example, the range and the cardinality of $a2$ must be specializations of the range and cardinality of $a1$.

Let o and o' be two objects: o *O-subsumes* o', and we write $o \succeq_{\mathcal{O}} o'$, if and only if (1) the set $\mathcal{A}o$ of definitional attributes of o is a subset of the set $\mathcal{A}o'$ of definitional attributes of o', and (2) for every attribute a in $\mathcal{A}o$, there exists a corresponding attribute a' in $\mathcal{A}o'$ that is subsumed by a.

The O-subsumption relation organizes objects in a hierarchy $\mathcal{H}$ and allows property sharing: *O-subsumption = ordering + property sharing*. Further, if $o \succeq_{\mathcal{O}} o'$, then the value of any definitional attribute $a(o')$ can be deduced from the value $a(o)$. Several kinds of property sharing rules can be considered, e.g. monotonic standard property sharing, monotonic and nonmonotonic cumulative property sharing.

In integrated object-oriented representation systems, the O-subsumption relation becomes the basis of classification-based reasoning, through a cycle making explicit the definitional dependencies holding between a new object `X` and the objects lying in the hierarchy $\mathcal{H}$. The classification cycle proceeds with three main steps (inspired by [Kaczmarek *et al.*,1986]), (1) *instantiation* of a new object `X`, (2) *classification* of `X`, i.e. searching for the most specific subsumers and the most general subsumees of `X`, (3) *updating operations* triggered by the insertion of `X` in the hierarchy. Thus, the definitional part of a new object is treated as a defined concept in terminological logics.

3 Points of Discussion

We have briefly stated what could be a different basis for the integration of classification-based technology in an object-oriented environment. Several points of discussion can be raised (note that these points of discussion follow the discussions held during the IJCAI Workshop "Object-Based Representation Systems", see [Dekker and Napoli,1994] and [OBRS,1993]):

- Inheritance and subsumption: when only monotonic property sharing is allowed – only definitional attributes are considered and values cannot be overridden – O-subsumption can be considered as equivalent to inheritance. Then, an object o inherits a property p if o is O-subsumed by an object owning p. However, if nonmonotonic property sharing is allowed, then the definition of O-subsumption has to be extended to take nonmonotonicity into account. Moreover, from a computational point of view, techniques such as linearization [Ducournau and Habib,1991] used to manage inheritance could be adapted to classification-based reasoning, e.g. to compute the most specific subsumers and the most general subsumees.
- Reflection and meta-information: every entity, including attributes, reflexes and methods, could and should be described by an object. In the same way, terminological meta-knowledge about a concept `C` such as the transitive closures of relations, the list of subsumers, subsumees and instances of `C`, could be represented by an object `MC` linked to `C` (see for example [Maes and Nardi,1987] and [Van Marcke,1988]).
- Terminological programming in integrated object-based representation systems: what kind of terminological programming can be defined, that could combine logical, access-oriented and object-oriented programming?
- Terminological typicality, and relations between classes, prototypes and concepts.

References

[Aït-Kaci, 1986] H. Aït-Kaci. An Algebraic Semantics Approach to the Effective Resolution of Type Equations. *Theoretical Computer Science*, 45(3):293–351, 1986.

[Baader and Hollunder, 1992] F. Baader and B. Hollunder. Embedding Defaults into Terminological Knowledge Representation Formalisms. In *Proceedings of* *the Third International Conference on Principles of Knowledge Representation and Reasoning (KR'92), Cambridge, Massachusetts*, pages 306–317, 1992.

[Baader and Hollunder, 1993] F. Baader and B. Hollunder. How to Prefer More Specific Defaults in Terminological Default Logic. In *Proceedings of the 13th IJCAI, Chambéry, France*, pages 669–674, 1993.

[Dekker and Napoli, 1994] L. Dekker and A. Napoli. Report on the IJCAI-93 Workshop on "Object-Based Representation Systems". *SIGART Bulletin*, 5(1):57–59, 1994.

[Doyle and Patil, 1991] J. Doyle and R.S. Patil. Two theses of knowledge representation: language restrictions, taxonomic classification, and the utility of representation services. *Artificial Intelligence*, 48(3):261–297, 1991.

[Ducournau and Habib, 1991] R. Ducournau and M. Habib. Masking and Conflicts, or To Inherit is Not To Own! In M. Lenzerini, D. Nardi, and M. Simi, editors, *Inheritance Hierarchies in Knowledge Representation and Programming Languages*, pages 223–244. John Wiley & Sons Ltd, Chichester, West Sussex, 1991.

[Ducournau, 1989] R. Ducournau. *Y3. Langage à objets. Version 3.22.* SEMA GROUP, Montrouge, 1989.

[Horty *et al.*, 1990] J.F. Horty, R.H. Thomason, and D.S. Touretzky. A Skeptical Theory of Inheritance in Nonmonotonic Semantic Networks. *Artificial Intelligence*, 42(2–3):311–348, 1990.

[Kaczmarek *et al.*, 1986] T.S. Kaczmarek, R. Bates, and G. Robins. Recent Developments in NIKL. In *Proceedings of AAAI'86, Philadelphia, Pennsylvania*, pages 978–985, 1986.

[Karp, 1993] P.D. Karp. The Design Space of Frame Knowledge Representation Systems. SRI AI Technical Note 520, SRI International, Menlo Park, 1993.

[Keene, 1989] S.E. Keene. *Object-Oriented Programming in Common Lisp. A Programmer's Guide to CLOS.* Addison Wesley, Reading, Massachusetts, 1989.

[MacGregor, 1991] R. MacGregor. The Evolving Technology of Classification-based Knowledge Representation Systems. In J. Sowa, editor, *Principles of Semantic Networks: Explorations in the Representation of Knowledge*, pages 385–400. Morgan Kaufmann Publishers, Inc., San Mateo, California, 1991.

[Maes and Nardi, 1987] P. Maes and D. Nardi, editors. *Meta-Level Architectures and Reflection.* North-Holland, Amsterdam, 1987.

[Masini *et al.*, 1991] G. Masini, A. Napoli, D. Colnet, D. Léonard, and K. Tombre. *Object-Oriented Languages.* Academic Press, London, 1991.

[Napoli, 1992] A. Napoli. Subsumption and Classification-Based Reasoning in Object-Based Representations. In *Proceedings of the 10th European Conference on Artificial Intelligence (ECAI'92), Vienna, Austria*, pages 425–429, 1992.

[Nebel, 1990] B. Nebel. *Reasoning and Revision in Hybrid Representation Systems.* Lecture Notes in Computer Science 422. Springer-Verlag, Berlin, 1990.

[OBRS, 1993] Proceedings of the IJCAI Workshop on Object-Based Representation Systems. Edited by A. Napoli, Rapport de Recherche CRIN 93-R-156, Nancy, France, 1993.

[Padgham and Nebel, 1993] L. Padgham and B. Nebel. Combining Classification and Nonmonotonic Inheritance Reasoning: A First Step. In J. Komorowski and Z.W. Raś, editors, *Methodologies for Intelligent Systems*, Lecture Notes in Computer Science 689, pages 132–141. Springer-Verlag, Berlin, 1993.

[Padgham and Zhang, 1993] L. Padgham and T. Zhang. A Terminological Logic with Defaults: A Definition and an Application. In *Proceedings of the 13th IJCAI, Chambéry, France*, pages 662–668, 1993.

[Patel-Schneider, 1990] P.F. Patel-Schneider. Practical, Object-Based Knowledge Representation for Knowledge-Based Systems. *Information Systems*, 15(1):9–19, 1990.

[Quantz and Royer, 1992] J. Quantz and V. Royer. A Preference for Defaults in Terminological Logics. In *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning (KR'92), Cambridge, Massachusetts*, pages 294–305, 1992.

[Rathke, 1993] C. Rathke. Implementing Subsumption in an Object-Oriented Language. In *IJCAI Workshop on Object-Based Representation Systems, Chambéry, France*, pages 101–109, 1993.

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

[Van Marcke, 1988] K. Van Marcke. The Use and Implementation of the Representation Language KRS. Technical Report 88-2, Vrije Universiteit, Brussel, 1988.

[Woods, 1991] W.A. Woods. Understanding Subsumption and Taxonomy: A Framework for Progress. In J. Sowa, editor, *Principles of Semantic Networks: Explorations in the Representation of Knowledge*, pages 45–94. Morgan Kaufmann Publishers, Inc., San Mateo, California, 1991.

[Yelland, 1992] P.M. Yelland. Experimental Classification Facilities for Smalltalk. In *Proceedings of 7th OOPSLA, Vancouver, ACM SIGPLAN Notices (27)10*, pages 235–246, 1992.

Where are all the Users?

Deborah L. McGuinness
AT&T Bell Labs
600 Mountain Avenue
Murray Hill, NJ 07974
U. S. A.
dlm@research.att.com

Implemented Description Logic (DL)-based knowledge representation systems have existed for over a decade, yet we do not see very large numbers of users. Description Logics may have more potential than is currently being realized. It is time for our community to address this issue directly. I propose working collectively to describe known classes of applications for DLs. I would also like to identify obstacles that may be limiting DL use. I will provide some specific questions that will facilitate this data collection. I propose that everyone who has some connection with a user of a DL try to answer the few specific questions at the end.

1 Historically Successful Application Areas

DLs have been used successfully in the broad area of expert systems [19] and particularly in configuration expert systems [20; 13]. Also, we have a long history of conceptual retrieval/knowledge base browser applications (see [5; 7; 15; 18], among others). Arguably, two newer successful application areas are natural language/translation [10; 12] and planning [11; 8; 9]. Are there other broad areas or niche applications that our community is aware of?

2 Limiting Factors?

Possibly description logics have not been more popular because they contain perceived or real limitations. I will present some possible limitations with a brief discussion.

1. Is there a fundamental limitation in the service they provide? I claim the answer is no. They provide a wealth of inferences: classsification, inheritance, propagation, rule firing, and a hundred or so other inferences. They work with descriptions, handle incomplete information, reason in the open world, etc. Arguably, this set is more valuable than some existing well-used AI tools such as expert systems shells.
2. Do people need what they provide? Since classification is one main distinguishing characteristic of most description logics, this question may come down to determining if an application can exploit classification. Informed users may not want the overhead of subsumption hierarchies if they are not going to use them. Still, classification can be useful in many contexts: it has been used in configuration problems to determine what kind of system is being configured before enforcing constraints; it has been used in conceptual retrieval applications to structure the knowledge base and provide quick cached answers to hierarchy questions; it can be exploited in incremental information gathering tasks, etc. So, I claim that a reasonable number of tasks can benefit from classification.
3. Can people use DLs effectively? Even more restricted description logics have a variety of constructs and inferences with which many users are not familiar. I claim that while sophisticated users can use DLs effectively, if we do not provide more introductory material, we will be limiting the scope of our users. Some DLs now have tutorials as well as user's manuals and some have discussions about when to use these systems (see LOOM [6] and CLASSIC [3; 17]). Most are making some concessions to better interfaces.
4. Is integration with other environments/tools essential? Do we need to interface our systems to the top ten databases on the market? Do we need to port them to PCs? Is Lisp (or PROLOG) just the wrong language for wider scale acceptance? Certainly some applications need to run on many platforms and also need to be integrated with diverse legacy systems. At least two systems, BACK and CLASSIC, are addressing the implementation language with C and/or C++ versions. In order for DLs to really take off, we may need to address more of these issuses.
5. Are the limits to the expressive power in some DLs too strict? Even one of the more limited systems - CLASSIC - has been used in very real applications so we have at least one data point that refutes this. At least one DL-based system is quite expressive, so if users really need extensive expressive power, they have an option. So, there is at least a range of expressive options. If, however, users need extensive expressive power and inferential completeness, then there is a problem. We might use LOOM applications to investigate current usage. Do Loom's users really

use all that expressive power? If they do, do they require inferential completeness?

3 General Questions

If potential users knew that Description Logics would provide a clear path to solving their problems, more users might use our tools. As a field, we might want to provide clear answers to the following questions:

1. What are the essential core of applications that are well suited to description logics?
2. What features of description logic based systems are crucial for these applications?
3. What are the inherent limitations of description logic based systems? (i.e., when might users need to use another technology?)
4. Why are Description Logics better (or worse) than their competitors?

4 Workshop Session Questions

In order to facilitate these high level goals, I propose that each application representative come prepared to answer the following questions:

1. How would you classify your application problem? (e.g., configuration, translation, conceptual retrieval, browsing/data mining, etc.)
2. What characteristics of description logics are useful in your application? (e.g. consistency checking for configuration, support for object centered modeling classification for conceptual retrieval, etc.)
3. How could these characteristics be attractive to users? (e.g. quick retrieval of classes and instances (assuming a cached subsumption hierarchy) with classification, etc.)
4. What deficiencies in your local DL based system did you need to overcome? (e.g., explanation and error handling in PROSE [20], lack of extensive support of graphical interfaces and knowledge acquistion tools in IMACS [5], etc.)
5. For your application, what is the best suited competitive technology? (e.g., expert system shells, object oriented databases, etc.) Why are DLs better or worse?

References

[1] A. Borgida, R. J. Brachman, D. L. McGuinness, and L. A. Resnick. CLASSIC: A Structural Data Model for Objects. In *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*, pages 59–67, June 1989.

[2] R. J. Brachman. "Reducing" CLASSIC to Practice: Knowledge Representation Theory Meets Reality. In *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning* October 1992.

[3] R. J. Brachman, D. L. McGuinness, P. F. Patel-Schneider, L. A. Resnick, and A. Borgida. Living with CLASSIC: When and How to Use a KL-ONE-Like Language. In *Principles of Semantic Networks: Explorations in the representation of knowledge*, J. Sowa, editor, Morgan-Kaufmann, pages 401–456, 1991.

[4] R. J. Brachman, A. Borgida, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Resnick. The CLASSIC Knowledge Representation System, or, KL-ONE: The Next Generation. In *Proceedings of the International Conference on Fifth Generation Systems*, Tokyo, June 1993.

[5] Brachman, Ronald J., Selfridge, Peter G., Terveen, Loren G., Altman, Boris, Borgida, Alex, Halper, Fern, Kirk, Thomas, Lazar, Alan, McGuinness, Deborah L., and Resnick, Lori Alperin. Integrated Support for Data Archaelogy. In *International Journal of Intelligent and Cooperative Information Systems*, 2(2):159–185, 1993.

[6] D. Brill. LOOM Reference Manual Version 2.0. University of Southern California, December 1993.

[7] Premkumar Devanbu, Ronald J. Brachman, and Peter G. Selfridge. LaSSIE—a classification-based software information system. In *Proceedings of the International Conference on Software Engineering.* IEEE Computer Society, 1990.

[8] P. Devanbu and D. Litman. Plan-based terminological reasoning. In J. Allen, R. Fikes, and E. Sandewall, editors, *Proc. 2nd International Conference on Principles of Knowledge Representation and Reasoning, Cambridge (Massachusetts, USA).* Morgan Kaufmann Publishers, April 1991.

[9] W.P. Johnson, R. Woodhead, and J. O'Brien. The ford motor company dlms. *AI Expert*, pages 42–52, August 1989.

[10] V. Kelly and M. Jones. Kitss: A knowledge-based translation system for test scenarios. In *Proc. 11th National Conference on Artificial Intelligence, Washington, D.C. (USA)*, pages 804–810, 1993.

[11] Mittal, V., Paris, C., Patil, R., Swartout, W. Organizing plan libraries in subsumption hierarchies: Specificity based plan selection Isi technical report # ISI-RS-93-305, ISI, Information Sciences Institute, USC, 4676 Admiralty Way, Marina del Rey, CA 90292.

[12] U. Nonnenman and J. K. Eddy. KITSS - A Functional Software Testing System Using a Hybrid Domain Model. In *Proceedings of the 8th IEEE Conference on Artificial Intelligence Applications*, Monterey, CA, 1992.

[13] Bernd Owsnicki-Klewe. Configuration as a Consistency Maintenance Task. In *Proceedings of GWAI-88—the 12th German Workshop on Artificial Intelligence*, 77–87, 1988.

[14] Peter F. Patel-Schneider, Deborah L. McGuinness, Ronald J. Brachman, Lori Alperin Resnick, and Alex Borgida. The CLASSIC Knowledge Representation

System: Guiding Principles and Implementation Rationale. In *SIGART Bulletin*, 2(3):108–113, June 1991.

[15] Patel-Schneider, P. F., Brachman, R. J., and Levesque, H. J., "ARGON: Knowledge Representation meets Information Retrieval," *Proc. First Conf. on Artificial Intelligence Applications*, Denver, CO, December, 1984, pp. 280–286.

[16] P. F. Patel-Schneider, B. Owsnicki-Klewe, A. Kobsa, N. Guarino, R. MacGregor, W. S. Mark, D. McGuinness, B. Nebel, A. Schmiedel, and J. Yen. Term subsumption languages in knowledge representation. *The AI Magazine*, 11(2):16–23, 1990.

[17] Lori Alperin Resnick, Alex Borgida, Ronald J. Brachman, Deborah L. McGuinness, and Peter F. Patel-Schneider. CLASSIC Description and Reference Manual for the COMMON LISP Implementation. AT&T Bell Laboratories, 1993.

[18] Peter Selfridge. Knowledge representation support for a software information system. In *IEEE Conference on Artificial Intellingence Applications*. The Institute of Electrical and Electronic Engineers, February 1991.

[19] Swartout, W., Paris, C., Moore, J. Design for explainable expert systems. *IEEE Expert*, *6*(3), 58–64, 1991.

[20] Jon R. Wright, Elia S. Weixelbaum,Karen Brown, Gregg T. Vesonder, Stephen R. Palmer, Jay I. Berman, and Harry H. Moore. A Knowledge-Based Configurator that Supports Sales, Engineering, and Manufacturing at AT&T Network Systems. In *Proceedings of the Innovative Applications of Artificial Intelligence Conferenc*, Washington, D. C., August, 1993.

Description Logics as core of a Tutoring System *

S. Bergamaschi°, S. Lodi, C. Sartori

Dipartimento di Elettronica, Informatica e Sistemistica
Università di Bologna - CIOC-CNR
°Facoltà di Ingegneria, Università di Modena - CIOC-CNR

1 Introduction

Intelligent Tutoring Systems (ITS) are information systems whose aim is to transfer domain knowledge and experience to a user (student). ITSs evolved from the earlier Computer Aided Instruction (CAI) systems; they represent knowledge declaratively and may adapt their behaviour to the student's characteristics. The problems of modelling the domain, the students' characteristics and the communication process are thus extremely relevant to the design of an ITS. In the following, we shortly describe an application of Description Logics to the design of an ITS prototype for the instruction of staff at G.D S.p.A., a metal and mechanics manufacturing company. The focus will be on the student's model and classification, showing how Description Logics can be a substantially sound support to the solution of this kind of problems.

2 Brief Description of the Problem

The ITS should integrate or substitute traditional courses, which are at present organized manually. This task must be carried out by highly qualified personnel, resulting in a waste of human resources.

An analysis of the structure of courses has revealed the following facts:

1. The course a student is assigned to depends on the skills the student needs to learn and on stereotypical information about the student.
2. Courses are differentiated according to the skills the students will master at the end of each; such set of skills is the "goal" of the course.
3. Every course has prerequisites, which are either other courses, or knowledge not included in the ITS, called "external" knowledge (previous experience or education).
4. Every course is a sequence of topics teaching the skills of the goal. Multiple "entry points" to a course are allowed, depending on the external knowledge;
5. the contents of a course are essentially static; content modification is not allowed, only repeated explanations are allowed;
6. explanations based on rules are seldom used;
7. general principles and general descriptions of components are widely used.

We relied on the assumption that a stereotype can approximate well the causal relation between as student's curriculum and the assigned course (1). The chosen approach was therefore to design and implement a DL core system as a part of larger ITS; the role of the DL core is to classify an individual student into a stereotype. The matching stereotype may then verify some of the prerequisites of a course a student might be interested in.

(2)–(7) show that representing the domain knowledge by rulesets, or other complex KR languages, is too complex, resulting in too small a granularity of the representation and high costs designing and maintaining the knowledge base, without offering substantial advantages with respect to simpler approaches. In particular, (3) suggests a further partitioning of the skills into their components, to be addressed independently during the teaching process. Knowledge transferred by the ITS will therefore be organized as *grains*, i.e. smaller units of knoweledge with prerequisites and postconditions, each of which is related to a multimedia presentation module. A personalized activation of grains, based on the student's goals and background, is the main service supplied by the ITS.

When a student is classified into a stereotype, all grains whose prerequisites are matched by the stereotype and whose postcondition is the given goal, are selected. In case no grain can be selected (because the skills are insufficient), a backward search will select the grains that have the required skills as postconditions.

3 A DL subsystem

When we evaluated the alternative approaches to building a core for the ITS, we met with several good arguments in favour of Description Logics. We summarize them:

- Students stereotypes essentially describe their technical knowledge. This information obviously has a hierarchical structure.
- The student's curriculum is actually a table of properties, most of which are attributes with structured values.

*This research has been funded by G.D S.p.A., Bologna, Italy.

- Complex implications are hardly needed to classify a student into a stereotype. The complexity and versatility of rule systems are unnecessary.
- Description Logic's proof theory is clear, even for expressive languages, and proof procedures are easy to implement, provided that efficiency for large KBs is not required.

The core was explicitly designed for the specific application we were supporting. Since the application's knowledge base was small enough to ignore efficiency issues and DL proof theory is now very well documented, we planned to cut down on implementation costs by implementing a simple DL interpreter based on the method of completion rules, and to concentrate fully on the Tutoring System's specific problems.

The cooperation with the domain's experts (the members of the company's automation department managing the courses) has been of primary importance; some DL constructors were included in the language because they were necessary to represent the domain's knowledge. Various arguments forced us not to reuse existing implemented systems. Among the technical ones, we needed to limit memory usage, but most systems have large executables. Secondly, it was recommendable to comply with industrial standards. The core system should be linked to an X11 client in the VAX/VMS environment with a minimal effort; the C language is therefore appropriate, but, to the best of our knowledge, stable versions of existing systems are implemented in LISP or Prolog. Last, but not least, Prolog and LISP are anyway not part of the company's know-how. Since the application is being written by the company staff, writing it in a relatively unknown language would significantly increase implementation costs.

Rather surprisingly, the time to implement a DL core prototype in C was not significantly longer than the time we needed in our last DL implementation in Prolog. This may be due to the thoroughness of the documentation available on Description Logic's completion rules [1][2][3], which we used in the core. The core language includes conjunction, union, negation, universal and existential quantification, inverse roles and role conjunction, concrete domain support, and is KRSS compatible.

4 Conclusions

Description Logics' theory is extremely well documented if inferences are done by completion, and is becoming thoroughly explained even for other proof techniques. It is thus feasible to implement a reasoning DL core in a relatively short time, provided that the implementors are acquainted with the basics of first order logic. On the other hand, our experience shows that we underestimated the time needed to explain the DL technology to our partners. Since a motivation for choosing the DL technology was to cut down KB maintenance costs (with respect to the Expert System technology), we have to invest more time in documenting DL constructs to our partners writing the application.

References

[1] F.M. Donini, M. Lenzerini, D. Nardi, and W. Nutt. The complexity of concept languages. In J. Allen, R. Fikes, and E. Sandewall, editors, *KR '91 - 2nd Int. Conf on Principles of Knowledge Representation and Reasoning*, pages 151–162, Cambridge - MA, April 1991. Morgan Kaufmann Publishers, Inc.

[2] Bernhard Hollunder and Werner Nutt. Subsumption algorithms for concept languages. DFKI Research Report RR-90-4, Deutsche Forschungszentrum für Künstliche Intelligenz GmbH, Kaiserslautern, Germany, April 1990.

[3] Bernhard Hollunder, Werner Nutt, and M. Schmidt-Schauss. Subsumption algorithms for concept languages. In *Proc. ECAI-90*, Stockholm, Sweden, 1990.

Using Description Logics in Intelligent Tutoring Systems

Gunnar Teege
Institut für Informatik, TU München
80290 München, Germany
email: teege@informatik.tu-muenchen.de

We identify the representation of domain knowledge in explanation systems as being the typical field for Description Logics to improve IT systems. We describe how several mechanisms from Description Logics can immediately be applied to solve problems in explanation systems. However, we also point out that an important mechanism needed in explanation systems is still missing in Description Logics: a mechanism for description decomposition.

1 Explanation and Training

Intelligent Tutoring Systems have been studied since a long time in the domain of artificial intelligence. In contrast to expert systems, however, IT systems have not yet found their way to a broader application. The existing systems are still research prototypes. The reason for this situation lies in the complexity of the task. Intelligent Tutoring Systems need not only know their domain, they also have to know how to teach this knowledge and they have to deal with user misconceptions.

We can identify different requirements for IT systems, depending on their application. Most of the existing systems are designed for *training* certain abilities (see [Harmon,1987]), e.g., debugging electronic circuits [Brown *et al.*,1982], solving algebraic equations [Sleeman,1982], or interpreting ground tracks [Swigger *et al.*,1987]. These systems usually contain their own strategies for solving the problems, and they try to match their own actions with the observed user's actions. The domain knowledge of these systems mainly consists of methods for solving problems in the domain. These methods are most naturally represented in the form of rules. The major drawback of this approach usually is the system's inability of explaining its own strategies and of explaining user errors [Clancey,1987; Clancey,1988]. Most systems try to improve their explanation ability, either with the help of additional knowledge [Reiser *et al.*,1992], or by replacing the rules by a less implicit representation mechanism.

However, apart from ability training there is the broad field of teaching fundamental principles, which have not necessarily an immediate application which can be trained. IT systems in this field must support the user in learning, clarifying, and consolidating fundamental knowledge of the kind which is taught in theoretical university courses. Here, the knowledge mainly consists of the explanation of concepts and their interrelations, the presentation of examples, and the definition and refinement of categories [Teege,1991]. I will denote this kind of IT systems by "explanation systems", in contrast to systems for training. A typical difference between these two kinds of systems is the tutoring strategy: "teaching" in explanation systems versus "monitoring" in training systems [Imbeau *et al.*,1990]. Although one of the first IT systems described in the literature, the SCHOLAR system [Carbonell,1970], was an explanation system, nearly all other IT systems were systems for training. One reason for this situation was the lack of a commonly accepted representation mechanism for the domain knowledge of explanation systems [Kearsley,1987, Chapter 8]. I will argue in this paper that Description Logics can do the job.

Note, that every IT system needs at least two kinds of knowledge, which can be clearly separated. One kind is the domain knowledge which has to be taught to the user. The other kind is the tutoring knowledge which tells the system *how* to teach. The latter kind is only used by the system, it need not be explained to the user. Hence it may be represented implicitly, such as in the form of rules.

Summing up, the main application of Description Logics in IT systems is their use for representing the (declarative) domain knowledge in explanation systems.

2 Knowledge Transfer by Communication

The main task of an explanation system is to communicate its domain knowledge to the user. The main problem of this task is the fact that the user typically is a novice in the domain. The user may not only be ignorant of the concepts presented by the system, but also of the words used by the system to denote the concepts. Moreover, the user may produce wrong input or meaningless input. The system may not simply detect and ignore such input, it has to deal with it for correcting the user. It must try to determine the reason for the input, identify the error(s) in it, and give the user an explanation.

This kind of processing requires a high flexibility in communication. On the one hand a pure natural language level is not sufficient, since the system needs the ability of paraphrasing and semantic processing. On the other hand a purely semantical level is not sufficient either, since the system must deal with errors which may result from wrong formulations. In particular, a semantics which maps all wrong statements or inconsistent inputs to a common "bottom element" is of no use for IT systems.

Here, Description Logics provide the necessary link. The refinement of the semantics provided, e.g., by predicate logic was a major design goal already for the development of KL-ONE [Brachman and Schmolze,1985]. Although Description Logics capture more details than the truth or falseness of statements, they still support the abstraction from the actual formulations in natural language and from the actual words used. The representation of description structures supported by Description Logics provides the basis for the flexible knowledge communication capabilities needed by explanation systems.

3 Description Logics and Language

The communication in explanation systems is most naturally done with the help of a natural language interface [Dessalles,1990]. There are other methods for explaining concepts and principles, such as with the help of diagrams [Waloszek *et al.*,1988]. However, these methods are complementary to natural language and can only be used in certain situations. Since the general aspects related to Description Logics are the same as those for natural language communication we solely concentrate on the latter.

If the domain knowledge is represented with the help of Description Logics the task of the interface is to translate between internal description structures and descriptions in natural language. An example of an existing system which performs this translation is the XTRA system [Kobsa,1991; Allgayer *et al.*,1989].

The main aspect of the translation in the case of tutoring systems is the adaptation of output generated by the system to the user's knowledge. If a description has to be communicated to the user, it is not sufficient to map the structure directly to a corresponding phrase in natural language. Instead, additional constraints have to be respected. Examples of these constraints are:

- restriction to the user's vocabulary,
- using specific related concepts in the formulation to emphasize the relation with them,
- omitting details which would currently be detracting,
- using concise formulations.

Other examples of adapting explanations to the user can be found in [Cullingford *et al.*,1981] and [Wilensky,1987].

Meeting these constraints typically involves processing on the Description Logics level. Consider a description with a translation which is not understood by the user. The system may replace it by a subsuming description which the user understands and a description of the difference between both descriptions. As an example, if the user does not know the term "description logic" the system replaces it by the subsuming term "logic" and adds the difference in the form of an explicit paraphrase "which supports representation and reasoning with description structures". A typical application is the explanation of an unknown concept with the help of most specific subsuming concepts which are known by the user.

4 Description Logics and Tutoring

Apart from supporting a flexible communication, Description Logics can also be used in the tutoring process itself. Here, a major task is the comparison between the domain knowledge of the system and that of the user. This comparison is based on a student model maintained by the system. Most existing systems with an adaptive student model use one of two approaches to representing knowledge the user is supposed to have [Polson and Richardson,1988]. In the first approach, called *overlay model*, this knowledge is simply a subset of the system's domain knowledge (e. g., in WEST [Burton and Brown,1982]). Using this method, the system can represent missing knowledge but it cannot represent wrong knowledge. The second approach incorporates predetermined typical bugs (e. g., in DEBUGGY [Burton,1982]). With this method a good reaction to user misconceptions can be achieved because a special explanation can be associated with every predetermined bug. The drawback is that it is not possible to handle unanticipated user bugs.

Using Description Logics, however, a new form of student model is possible: a domain model represented in the same way as the system's domain model. No fixed relation with the system's domain model is necessary. In particular, the student's model need not be a subset of the system's model. The student model is build and modified by translating the user's input. Hence it contains everything the user has "told" the system about the domain.

The representation of the models with the help of Description Logics facilitates their comparison. The words associated with concepts serve as "anchor points" to relate user concepts and system concepts. This way it is possible to represent bugs of the form "when the user calls something a description logic she means something different from what the system calls description logic". It should be clear that all possible bugs of this form can be represented by the system without the need of anticipating them in a "bug catalog".

Moreover, by comparing the structure of the two concepts the system can determine special cases of this situation. If the student's concept subsumes the system's concept the student is not aware of all its aspects. If the student's concept is subsumed by the system's concept the student's view is too narrow. Additionally, the system can *react* to each bug in a sensible way. It can construct a concept which covers the difference between the system's concept S and the user's concept U, and it may then use this difference for correcting the misconception. A simple way of doing this is by generating a natural language description from the difference and tell the user what is wrong with his concept. A system with more elaborate tutoring strategies, such as, e.g., socratic methods, can search the student model for a concept Z which is subsumed by S but not by U. Then it asks the user, e.g., "well, consider Z. Do you think that is not a U?". If there is no such Z in the user model the system may construct one. This works in every case where U does not subsume S, i.e., when the user has a too specific or differing view of the concept.

The applications for Description Logics in IT systems proposed here allow an important observation. The main operations needed are description comparison and description decomposition. The subsumption relation in Description Logics is well suited for the kind of comparisons needed in IT systems. However, operations for description decomposition have not yet been investigated in general for Description Logics. Hence we proposed a new subtracting operation which can be used for description decomposition as well as for other purposes [Teege,1994].

Finally, we will shortly mention two other applications of Description Logics in IT systems. Up to now we only used the terminological part of the logic. Many Description Logics also support the representation of information about individuals in an assertional part. In an explanation system individuals are of major importance for giving examples. An example used for clarifying a fundamental principle typically involves individualizations of the general concepts which occur in the principle. Hence, Description Logics with support for individuals provide the necessary operations which relate the example with the principle.

The second aspect is the association of information with concepts. In the domain knowledge of an explanation system concepts will have lots of associated informations, such as a definition, typical usage, relevance, and others. Not all of these informations need be represented in Description Logics. The IT system accesses and retrieves these informations via the concept. Hence, this situation corresponds to the use of Description Logics as query languages, as it has been considered by Lenzerini and Schaerf [Lenzerini and Schaerf,1991a; Lenzerini and Schaerf,1991b].

5 Conclusion

The most interesting application of Description Logics in the field of Intelligent Tutoring Systems is for representing the domain knowledge of explanation systems. Several existing mechanisms from Description Logics can immediately be applied to problems in IT systems. Additionally, means for description decomposition are needed. A new kind of student model can be designed by representing the student's domain knowledge independently from that of the system. Hence, the use of Description Logics could be a major source of improvements for explanation systems.

References

[Allgayer *et al.*, 1989]
Allgayer, J., et al.: Bidirectional Use of Knowledge in the Multi-modal NL Access System XTRA. In: Sridharan, N. S. (ed.): Proceedings of the International Joint Conference on Artificial Intelligence, San Mateo, Calif.: Morgan Kaufmann, 1989, pp. 1492–1497

[Brachman and Schmolze, 1985]
Brachman, R. J., Schmolze, J. G.: An Overview of the KL-ONE Knowledge Representation System. Cognitive Science **9**, 171–216 (1985)

[Brown *et al.*, 1982]
Brown, J. S., Burton, R. R., d.Kleer, J.: Pedagogical, natural language and knowledge engineering techniques in SOPHIE I, II and III. In: Sleeman, D., Brown, J. (eds.): Intelligent Tutoring Systems, Chap. 11, pp. 227–282, London: Academic Press, 1982

[Burton and Brown, 1982]
Burton, R. R., Brown, J. S.: An investigation of computer coaching for informal learning activities. In: Sleeman, D., Brown, J. (eds.): Intelligent Tutoring Systems, Chap. 4, pp. 79–98, London: Academic Press, 1982

[Burton, 1982]
Burton, R. R.: Diagnosing bugs in a simple procedural skill. In: Sleeman, D., Brown, J. (eds.): Intelligent Tutoring Systems, Chap. 8, pp. 157–184, London: Academic Press, 1982

[Carbonell, 1970]
Carbonell, J. R.: AI in CAI: An Artificial Intelligence Approach to Computer Assisted Instruction. IEEE Transactions on Man-Machine Systems **11**, 190–202 (1970)

[Clancey, 1987]
Clancey, W. J.: Knowledge-Based Tutoring: The Guidon Program. MIT Press, Cambridge, Massachusetts, London, England, 1987

[Clancey, 1988]
Clancey, W. J.: The Role of Qualitative Models in Instruction. In: Self, J. (ed.): Artificial Intelligence and Human Learning, Chap. 3, pp. 49–68, London: Chapman and Hall, 1988

[Cullingford *et al.*, 1981]
Cullingford, R. E., Krueger, M. W., Selfridge, M., Bienkowski, M. A.: Towards Automating Explanations. In: Proceedings of the International Joint Conference on Artificial Intelligence, 1981, pp. 362–367

[Dessalles, 1990]
Dessalles, J.-L.: Computer Assisted Concept Learning. In: Norrie, D. H., Six, H. W. (eds.): Proceedings of the 3rd International Conference on Computer Assisted Learning ICCAL'90. Lecture Notes in Computer Science 438, Berlin: Springer, 1990, pp. 175–183

[Harmon, 1987]
Harmon, P.: Intelligent Job Aids: How AI Will Change

Training in the Next Five Years. In: Kearsley, G. (ed.): Artificial Intelligence and Instruction, Chap. 8, pp. 165–190. Addison-Wesley, 1987

[Imbeau *et al.*, 1990]
Imbeau, G., Gauthier, G., Frasson, C.: Wordtutor: An Intelligent Tutoring System for Teaching Word Processing. In: Norrie, D. H., Six, H. W. (eds.): Proceedings of the 3rd International Conference on Computer Assisted Learning ICCAL'90. Lecture Notes in Computer Science 438, Berlin: Springer, 1990, pp. 400–419

[Kearsley, 1987]
Kearsley, G. (ed.): Artificial Intelligence and Instruction: Application and Methods. Addison-Wesley, 1987

[Kobsa, 1991]
Kobsa, A.: Utilizing Knowledge: The Components of the SB-ONE Knowledge Representation Workbench. In: Sowa, J. F. (ed.): Principles of Semantic Networks, pp. 457–487. Morgan Kaufmann, 1991

[Lenzerini and Schaerf, 1991a]
Lenzerini, M., Schaerf, A.: Concept Languages as Query Languages. In: Proceedings of the AAAI Conference, 1991, pp. 471–476

[Lenzerini and Schaerf, 1991b]
Lenzerini, M., Schaerf, A.: Querying Concept-based Knowledge Bases. In: Boley, H., Richter, M. M. (eds.): Proceedings International Workshop Processing Declarative Knowledge. Springer, July 1991, pp. 107–123

[Polson and Richardson, 1988]
Polson, M. C., Richardson, J. J.: Intelligent Tutoring Systems. Lawrence Erlbaum Associates Publishers, Hillsdale, New Jersey, 1988

[Reiser *et al.*, 1992]
Reiser, B. J., et al.: Knowledge Representation and Explanation in GIL, an Intelligent Totor for Programming. In: Larkin, J. H., Chabay, R. W. (eds.): Computer-Assisted Instruction and Intelligent Tutoring Systems, Chap. 4, pp. 111–149, Hillsdale, New Jersey: Lawrence Erlbaum, 1992

[Sleeman, 1982]
Sleeman, D.: Assessing aspects of competence in basic algebra. In: Sleeman, D., Brown, J. (eds.): Intelligent Tutoring Systems, Chap. 9, pp. 185–200, London: Academic Press, 1982

[Swigger *et al.*, 1987]
Swigger, K., et al.: An Intelligent Tutoring System for Interpreting Ground Tracks. In: Proceedings of the AAAI Conference, 1987, pp. 72–76

[Teege, 1991]
Teege, G.: Ein System zur Repräsentation von deklarativem Gebietswissen für intelligente Tutorsysteme. Technische Universität München, PhD thesis, 1991

[Teege, 1994]
Teege, G.: Making the Difference: A Subtraction Operation for Description Logics. In: Doyle, J., Sandewall, E., Torasso, P. (eds.): Principles of Knowledge Representation and Reasoning: Proc. of the 4th International Conference (KR94), San Francisco, CA: Morgan Kaufmann, 1994

[Waloszek *et al.*, 1988]
Waloszek, G., Weber, G., Wender, K. F.: Probleme der Wissensrepräsentation in einem intelligenten LISP-Tutor. In: Heyer, G., et al. (eds.): Wissensarten und ihre Darstellung. Informatik-Fachberichte 169, Berlin, Heidelberg: Springer, 1988, pp. 180–193

[Wilensky, 1987]
Wilensky, R.: The Berkeley Unix Consultant Project. In: Proceedings of 2. Internationaler GI-Kongreß München. Informatik-Fachberichte 155, Berlin, Heidelberg: Springer, 1987, pp. 286–296

Description Logics for Natural Language Processing

D. Fehrer, U. Hustadt, M. Jaeger, A. Nonnengart,
H. J. Ohlbach, R. A. Schmidt, Ch. Weidenbach, and E. Weydert
Max-Planck-Institut für Informatik,
Im Stadtwald
66123 Saarbrücken
Germany

1 Introduction

In this paper we focus on the application of description logics to natural language processing. In cooperation with the PRACMA Project[1] (SFB 314, Universität des Saarlandes, Germany) we have been developing a suitably extended knowledge representation system, called MOTEL.

In the late eighties inference in KL-ONE was shown to be undecidable. Since then the emphasis in research has been on developing and investigating systems that are computationally well behaved, i.e. are tractable or at least decidable.

As a result many commonly used description logics (also known as *terminological logics* or KL-ONE-*based knowledge representation formalisms*) have restricted expressiveness and are in their current form not suitable for natural language applications. This is evident, for example, from Schmidt [1993] who links knowledge representation with a relational approach to natural language semantics. For encoding knowledge formulated in a very limited fragment of English we already need the full expressive power of role constructs which have been eliminated in many languages.

In our approach to agent modelling and natural language processing we use an extension of the well-known description language $\mathcal{ALC}$. Our system MOTEL serves on one hand as a knowledge base for the natural language front-end, and on the other hand, it provides powerful *logical* representation and reasoning components. As our approach is logic based we hope that this enhances the overall capabilities of the natural language processing (NLP) system.

2 Natural Language Processing

The PRACMA project is concerned with pragmatic dialogue processing between two agents. These agents have the following properties:

(1) They communicate in natural language.
(2) They actively pursue complex goals, which may be conflicting.
(3) They have the means of analyzing (some of) the pragmatic content of what is being said, i.e., they have a deeper understanding of 'belief', 'intension' or 'argument'.

Figure 1 shows the architecture of the PRACMA system. The system is decomposed into modules. Each module is realized as an autonomous problem solver.

The module for recognizing propositional attitudes analyses certain linguistic expressions, e.g. modal verbs and modal adverbs. The results are stored in the agent model.

The module for assessment processing recognizes the positive and negative assessments of the agents towards certain objects, facts, and relations. The results are stored in the assessment knowledge base.

Instances of the plan processing module are the action planner controlling the agent's activities, e.g. collecting facts about objects, and the dialogue planner controlling the dialogue behaviour of the agent, e.g. opening the dialogue, raising a question. The planners rely on the agent model and the assessment knowledge base. In addition, they use the conceptual knowledge base, the argumentation strategy knowledge base, and the EGO knowledge base. The EGO stores behavioural patterns, e.g. the degree of cooperativity.

During the processing of a dialogue, each module can exist in multiple instantiations, called *actors*, working in parallel. The actors communicate and interact with each other using a protocol based on communication acts, i.e. on message exchange.

The test domain for the first prototype of the PRACMA system has been the processing of a dialogue between a car salesman and his customer. Figure 2 shows a small part of a dialogue and a schematic representation of its processing.

Note the following:

(1) The agents are not only exchanging facts, but beliefs, demands, etc. (e.g. 'So I don't *want* to buy it.').
(2) The beliefs of the salesman and the customer can contradict each other. The agents are able to detect such contradictions and can try to resolve them (e.g. 'No, that's not true.').

[1] PRACMA is short for 'PRocessing Arguments between Controversially Minded Agents.'

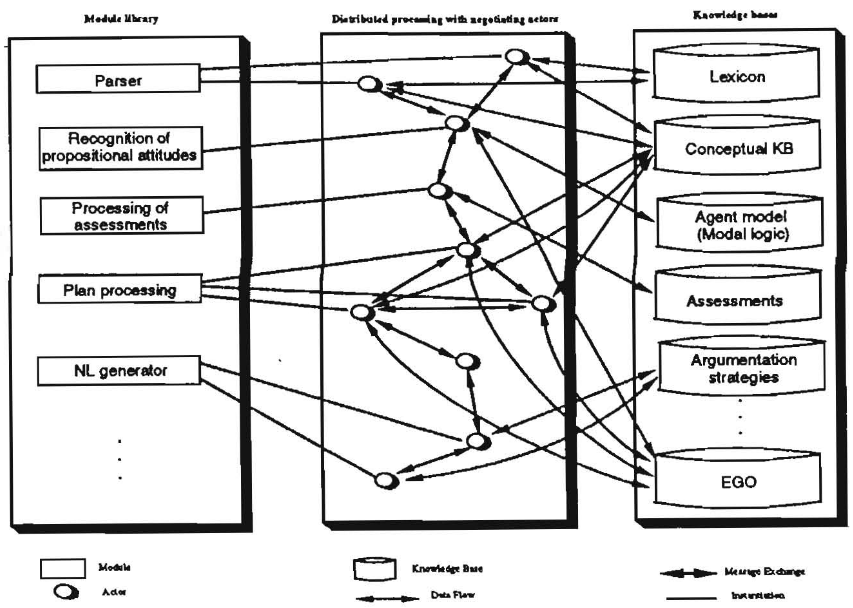

Figure 1: The architecture of the PRACMA system.

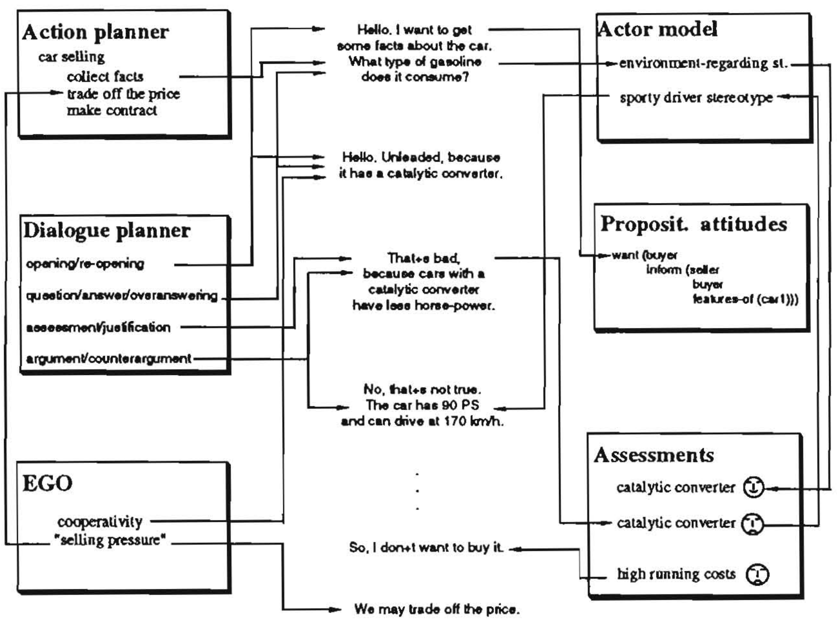
Action planner
car selling
collect facts
trade off the price
make contract
Dialogue planner
opening/re-opening
question/answer/overanswering
assessment/justification
argument/counterargument
EGO
cooperativity
"selling pressure"
Hello. I want to get some facts about the car. What type of gasoline does it consume?
Hello. Unleaded, because it has a catalytic converter.
That's bad, because cars with a catalytic converter have less horse-power.
No, that's not true. The car has 90 PS and can drive at 170 km/h.
So, I don't want to buy it.
We may trade off the price.
Actor model
environment-regarding st.
sporty driver stereotype
Proposit. attitudes
want (buyer inform (seller buyer features-of (car1)))
Assessments
catalytic converter
catalytic converter
high running costs

Figure 2: A sample dialogue.

(3) The agents are able to give reasons for their conclusions (e.g. 'Unleaded, because it has a catalytic converter.').

(4) Part of the information passed between the agents doesn't fit into a terminological system at all (e.g. 'That's bad.').

From the last item it is evident that terminological knowledge representation systems in their current form are not suitable for encoding the EGO and the assessment knowledge base. MOTEL is used to build the conceptual knowledge base and the agent model(s).

In the following sections we describe MOTEL and the different extensions we are working on.

3 Multi-Modalities

The traditional description logics can be used for representing common and individual knowledge about the world (domain of application). Recently description logics have been extended to allow the representation of the knowledge and the beliefs of multiple agents in one knowledge base [Donini *et al.*,1992; Kobsa,1992]. In MOTEL we formulate knowledge and belief as additional modal operators.

We are using $\mathcal{ALCNR}$ [Baader and Hollunder,1990] as a base language. That is, we assume three disjoint alphabets, the set of *concept names* C, the set of *role names* R, and the set of individual objects O. The set of *concept terms* (or just *concepts*) and *role terms* (or just *roles*) is inductively defined as follows. Every concept name is a concept term and every role name is a role term. Now assume that C, C_1, and C_2 are concepts, and R, R_1, and R_2 are roles. Then $C_1 \sqcap C_2$, $\neg C$, $\exists R.C$, $\exists_{\geq n} R.C$, and $\exists_{\leq n} R.C$ are concept terms, and $R_1 \sqcap R_2$, R^{-1}, $R|_C$ are role terms. The sentences of $\mathcal{ALCNR}$ are divided into *terminological sentences* and *assertional sentences*. If C_1 and C_2 are concepts and R_1 and R_2 are roles then $C_1 \sqsubseteq C_2$ and $R_1 \sqsubseteq R_2$ are terminological sentences. If C is a concept, R is a role, and O, O_1, and O_2 are individual objects then $O \in C$ and $(O_1, O_2) \in R$ are assertional sentences. As in [Baader and Hollunder,1990] we do not allow terminological cycles.

For the extended language Mod-$\mathcal{ALCNR}$ we assume in addition that we have an alphabet M of *modal operator names*. Also, there is a distinguished subset A of the individual objects, called the set of *agents*. We have a distinguished concept name '*all*' denoting the set of all agents with which we express mutual belief. The set of concepts and the set of roles of Mod-$\mathcal{ALCNR}$ contains all the concepts and roles of its sublanguage $\mathcal{ALCNR}$ and in addition it contains the concepts $\Box_{(m,a)} C$, $\Diamond_{(m,a)} C$, $\Box_{(m,C_1)} C_2$, and $\Diamond_{(m,C_1)} C_2$, and the roles $\Box_{(m,a)} R$, $\Diamond_{(m,a)} R$, $\Box_{(m,C)} R$, and $\Diamond_{(m,C)} R$, where m is a modal operator name and a is an agent name. The set of terminological and assertional sentences of Mod-$\mathcal{ALCNR}$ contains all the terminological and assertional sentences of $\mathcal{ALNCNR}$ and in addition it contains the expressions $\Box_{(m,a)} \Phi$, $\Diamond_{(m,a)} \Phi$, $\Box_{(m,C_1)} \Phi$, and $\Diamond_{(m,C_1)} \Phi$, where Φ is either a terminological or an assertional sentence.

We use a translational approach to provide the usual inference mechanisms, i.e. solving the consistency, the subsumption, the instantiation and the realization problem. Obviously, $\mathcal{ALCNR}$ knowledge bases can be translated into first-order logic theories. There are also well-known relational translation methods for modal logics. In [Hustadt and Nonnengart,1993] we have developed an improved translation method for Mod-$\mathcal{ALCNR}$ which provides an elegant translation of knowledge bases into first-order logic theories. In a prototypical implementation, the MOTEL system, we use a Prolog-based system with loop-checking as inference machine.

4 Quantitative Information

In MOTEL we use the cardinality-based approach proposed by Owsnicki-Klewe [1990] for dealing with number restrictions. Unfortunately, this approach is incomplete for languages in which concept disjointness is expressible.

The approach of Baader and Hollunder [1990], by contrast, provides a complete tableau method for $\mathcal{ALCNR}$, but has some disadvantages:

(1) The approach is not adequate for dealing with large numbers. Consider the following example: Suppose the universe consists of at most thirty objects. If there are at least twenty objects in C_1 and there are at least twenty objects in C_2, then there are at least ten objects in the intersection of C_1 and C_2.

The human ability to draw this conclusion is completely independent of the numbers we are using. Multiplying all numbers occurring in the example by a factor of 10 wouldn't make it any harder for us come up with the correct answer. Quite the opposite is true for the tableau method.

(2) The basic inference mechanism provided by tableau theorem provers is consistency checking for knowledge bases. This is adequate for answering queries that can be solved by checking the consistency of a suitably extended knowledge base, for example, for problems like subsumption, instantiation, and classification.

But the most suggestive class of queries for knowledge bases in $\mathcal{ALCNR}$, e.g. the question 'How many objects are in C_1 and C_2?' in the example above, cannot even be formulated.

A promising approach to quantitative reasoning with numerical quantifiers seems to be that of Hustadt et al. [1994], who investigate a translation technique which translates modal logics with graded modalities into a fragment of many-sorted first-order logic. For, $\mathcal{ALCNR}$ expressions can be associated directly with modal expressions.

5 Probabilistic Reasoning

Although Mod-$\mathcal{ALCNR}$ is a very sophisticated concept description language, the relationships among concepts that can be described are purely qualitative. Only inclusion, equality or disjointness relationships among concepts can be expressed. Jaeger [1994] investigates an extension of terminological knowledge representation languages that incorporates probabilistic statements. The language $\mathcal{PALC}$ based on $\mathcal{ALC}$ allows the following two additional kinds of sentences. *Probabilistic terminological sentences* are expressions $P(C_1|C_2) = p$, where C_1 and C_2 are concept terms and $p \in [0,1]$. *Probabilistic assertional sentences* are expressions $P(a \in C) = p$, where a is an element of O and $p \in [0,1]$. A knowledge base $\mathcal{KB}$ in $\mathcal{PALC}$ consists of a set $\mathcal{T}$ of terminological sentencesrestricted to $\mathcal{ALC}$, a set $\mathcal{PT}$ of probabilistic terminological sentences and a set $\mathcal{P}_a$ of probabilistic assertional sentences for every object name a: $\mathcal{KB} = \mathcal{T} \cup \mathcal{PT} \cup \bigcup\{\mathcal{P}_a | a \in \mathsf{O}\}$.

It is important to realize that these two kinds of probabilistic statements are completely different from each other. The former codifies *statistical information* that, in general, is obtained by observing a large number of individual objects and checking their membership of the various concepts. The latter expresses a *degree of uncertainty* of our belief in a specific proposition. Its value is usually justified only by a subjective assessment of likelihood.

Both kinds of probabilistic statements are interpreted in one common probability space which essentially consists of the set of concept terms that can be formed in the language of the given knowledge base. Defining all *the probability measures on the same probability space* allows us to compare the measure assigned to an object a with the generic measure defined by the given statistical information. The most reasonable assignment of a probability measure to a, we choose then, among all the measures consistent with the constraints known for a is the one that most closely resembles the generic measure. The key question to be answered, therefore, is how resemblance of probability measures should be measured. We chose the method of minimizing the cross entropy of the two measures.

6 Non-monotonic Reasoning

We have considered two different extension of the language $\mathcal{ALC}$ and its inference mechanisms to incorporate non-monotonic reasoning in MOTEL.

The first approach extends the language with an operator $\mathcal{A}$ of *assumability*. This operator can be applied to any concept term or role term, but it can only occur on the left-hand side of terminological sentences. The resulting language is called $\mathcal{ALCP}$.

A knowledge base $\mathcal{KB}$ in $\mathcal{ALCP}$ entails an assertion $a \in \mathcal{A}(C)$ iff $a \in \mathcal{A}(C)$ holds in all preferred models of $\mathcal{KB}$. Preference is defined with respect to the so-called *assumption order*.

In essence the implementation uses the negation as failure operator of PROLOG.

The second approach adds a new sentential operator T to $\mathcal{ALC}$ and a new subset declaration symbol $\sqsubseteq_T$. If C_1 and C_2 are concept terms and Φ is a terminological sentence, then $C_1 \sqsubseteq_T C_2$, and $T\Phi$ are terminological sentences.

To provide a proof theory and a semantics for the extended language, we define a translation function mapping knowledge bases $\mathcal{KB}$ to default theories (W, D), where W is the set of first-order formulae and D is a set of supernormal defaults. The semantics of a knowledge base $\mathcal{KB}$ is the set of all possible extensions of (W, D). A knowledge base $\mathcal{KB}$ entails a sentence Φ iff Φ is entailed by every extension of (W, D).

7 Abductive Reasoning

Abduction was introduced by the philosopher Pierce as one of the three main forms of reasoning (the other two being deduction and induction). Abduction has widespread application in natural language processing systems. For example, Guessoum et al. [1993] describe the use of abduction for pronoun resolution. Most of the existing NLP systems use linguistic constraints for eliminating candidate referents, but it is widely recognized that non-linguistic knowledge is required to resolve ambiguities in general (c.f. the textbook example 'If the baby doesn't thrive on cows' milk, boil <u>it</u>'). More interesting for our testbed is the work of Quaresma and Lopes [1993] on abduction of plans and intentions in dialogues.

Hustadt [1993] proposes an abductive proof procedure for disjunctive logic programs with integrity constraints. Extending the class of normal logic programs to a class of *programs including disjunction and integrity constraints* permits arbitrary first-order problems to be stated in proper input format.

8 Reason Maintenance

There are at least two reasons why it is interesting to incorporate reason maintenance into a system like the one proposed here. The first is, that it may prove valuable not to dispose of the answers found to queries, but to keep them in order to be able to respond faster if the same queries or instances thereof occur again (similar to the use of lemmata in mathematics). As the knowledge base, however, is of dynamic nature, lemmata are only useful if their origins are remembered. The second reason is that we can't be sure that a knowledge base is globally consistent. So it is worthwhile looking for nogoods and reporting them, so that the master component is aware of them, or at least it can be guaranteed that in a single 'explanation' (proof) no inconsistent material is used (a kind of paraconsistency).

Fehrer [1993] shows how a reason maintenance system based on an arbitrary *basic logic* can be described logically. He also shows there, how an inference system can be obtained, given a calculus (axioms and set of inference rules) for the basic logic. As a special case we can get a system for $\mathcal{ALC}$.

At this stage this result is only of theoretical interest. The main advantage for using terminological logics, instead of full first order logics, lies in the fact that they have efficient algorithms for decidable fragments. The compound logic resulting from putting the reason maintenance onto $\mathcal{ALC}$ unfortunately cannot always make use of these algorithms (If we are content with only keeping track of the origins of lemmata generated so far, there is no problem, for the derived calculus inherits all the important properties from its ancestor, so the algorithms can be adapted in a simple manner). This is in essence due to the fact that in order to check for nogoods we have to generate *all possible derivations* of the falsum. If, however, we start with a possibly inconsistent knowledge base some *proof strategies* do not yield all possible derivations, for example, strategies incorporating set of support. But, since decidability as well as completeness is preserved in the compound system, it should be possible to devise algorithms with acceptable properties for that task.

9 Future Work

We want to focus on three parts of the architecture of natural language processing systems like PRACMA: The parser, the plan processing/NL generating modules, and knowledge representation system.

The natural language generating part of the system [Reithinger,1992] is a classical hierarchical planning system. In the current state, it doesn't make any use of the reason maintenance and abductive reasoning abilities of the knowledge representation system. The integration of these services of our system should improve the PRACMA system considerably.

The second prototype of PRACMA will use a parser translating natural language utterances into the semantic representation language $\mathcal{NLL}$. The language contains a first-order logic core, Boolean sentential operators, generalized quantifiers, plural reference expressions, λ-abstraction predicates, etc.

On the one hand, $\mathcal{NLL}$ provides more expressive power than we do in our terminological language. However, on the other hand, our language has syntactic constructs (like modal and probabilistic operators) not available in $\mathcal{NLL}$. If we would extend the syntax of both logics to a common language, we cannot provide correct and complete inferential mechanism for this logic. Therefore, we will have a core logic (based on Mod-$\mathcal{ALC}$) with correct and complete inference mechanisms and an extended logic (based on $\mathcal{NLL}$) with neither correct nor complete inference mechanisms.

10 Conclusion

We share the view of Doyle and Patil [1991] who argue for expressiveness as opposed to computational efficiency. Our experience with users interested in agent modelling and natural language simulations can be summarized as follows:

(1) Users want expressiveness.

(2) They want representation languages with more basic features than just concepts, roles and individuals (i.e. A-Box elements) and operations on these.

(3) And, they want special inference tools.

References

[Baader and Hollunder, 1990] F. Baader and B. Hollunder. $\mathcal{KRIS}$: Knowledge Representation and Inference System. System Description. Technical Memo DFKI-TM-90-03, Deutsches Forschungszentrum für Künstliche Intelligenz, Saarbrücken, Germany, 1990.

[Donini *et al.*, 1992] Francesco M. Donini, M. Lenzerini, D. Nardi, A. Schaerf, and W. Nutt. Adding Epistemic Operators to Concept Languages. In B. Nebel, C. Rich, and W. Swartout, editors, *Proceedings of the KR '92*, pages 342–353, Cambridge, MA, 1992. Morgan Kaufmann.

[Doyle and S., 1991] J. Doyle and Patil R. S. Two theses of knowledge representation: language restrictions, taxionomic classification, and the utility of representation services. *Artifical Intelligence*, 48:261–297, 1991.

[Fehrer, 1993] Detlef Fehrer. A Unifying Framework for Reason Maintenance. In M. Clarke, R. Kruse, and S. Moral, editors, *Proceedings of the European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty (ECSQARU '93)*, volume 747 of *LNCS*, pages 113–120, Granada, Spain, 1993. Springer.

[Guessoum *et al.*, 1993] A. Guessoum, B. Black, and J. Gallagher. Abduction for pronoun resolution. In P. Codognet, P. M. Dung, A. C. Kakas, and P. Mancarella, editors, *ICLP93 Postconference Workshop on Abductive Reasoning*, Budapest, Hungary, 1993.

[Hustadt and Nonnengart, 1993] U. Hustadt and Andreas Nonnengart. Modalities in knowledge representation. In C. Rowles, H. Liu, and N. Foo, editors, *Proceedings of the 6th Australian Joint Conference on Artificial Intelligence*, pages 249–254, Melbourne, Australia, 1993. World Scientific.

[Hustadt *et al.*, 1994] Ullrich Hustadt, H. J. Ohlbach, and Renate A. Schmidt. Qualified number restrictions and modal logic. Forthcoming MPI Report, 1994.

[Hustadt, 1993] Ullrich Hustadt. Abductive disjunctive logic programming. In P. Codognet, P. M. Dung, A. C. Kakas, and P. Mancarella, editors, *ICLP 93 Postconference Workshop on Abductive Reasoning*, Budapest, Hungary, 1993.

[Jaeger, 1994] Manfred Jaeger. Probabilistic reasoning in terminological logics. In J. Doyle, E. Sandewall, and P. Torasso, editors, *Proceedings of the KR '94*, Bonn, Germany, 1994. Morgan Kaufmann.

[Kobsa, 1992] Alfred Kobsa. Towards Inferences in BGP-MS: Combining Modal Logic and Partition Hierarchies for User Modeling (Preliminary Report). Bericht WIS-2, Projekt BGP-MS, Universität Konstanz Informationswissenschaft, June 1992.

[Owsnicki-Klewe, 1990] B. Owsnicki-Klewe. A cardinality-based approach to incomplete knowledge. In *Proceeding of the 9th European Conference on Artifical Intelligence*, pages 491–496, 1990.

[Quaresma and Lopes, 1993] P. Quaresma and J. G. Lopes. Abduction of plans and intentions in dialogues. In P. Codognet, P. M. Dung, A. C. Kakas, and P. Mancarella, editors, *ICLP93 Postconference Workshop on Abductive Reasoning*, Budapest, Hungary, 1993.

[Reithinger, 1992] Norbert Reithinger. *Eine parallele Architektur zur inkrementellen Generierung multimodaler Dialogbeiträge*, volume 1 of *DISKI - Dissertationen zur KI*. infix, St. Augustin, 1992.

[Schmidt, 1993] R. A. Schmidt. Terminological representation, natural language & relation algebra. In H. J. Ohlbach, editor, *Proceedings of the GWAI-92*, volume 671 of *LNAI*, pages 357–371, Bonn, Germany, 1993. Springer.

Lexical Semantics, Description Logics and Natural Language Systems

Peter Forster
Institut für Informatik, Universität Stuttgart
Breitwiesenstr. 20-22
D-70565, Germany
email: forster@informatik.uni-stuttgart.de

l Introduction

Knowledge representation techniques play an important ·ole in natural language understanding. While theories ıbout the recognition of the syntactic structure of a sen-.ence are accepted widely, theories about the semantic ıtructure of a sentence are currently under discussion. Iowever, most researchers agree that the contents of vords are essential constituents of the semantics of a sen-.ence. Thus, the representation of the semantic structure ıf a sentence strongly depends on the representation of .he contents of words.

As dictionary definitions based on "genus proximum ınd differentia specifica" and concept descriptions in De-:cription Logics are similar, it seems useful to investigate ıow Description Logics can be explored to represent the :ontent of words.

Our first attempt was to represent the content of sever-ıl dictionary definitions in a Description Logic formalism. Chese investigations lead to the definition of further re-ıuirements for the representation formalism, e.g. defaults ır qualifying number restrictions.

With these additional requirements in mind, we are ntegrating a knowledge representation component in a ıatural language system. A first version of the natural anguage system is used for a question-answering system ıbout geographics. Geographical knowledge is described ıy the knowledge representation component.

In this position paper we give in the first section a short ummary about our work on the representation of the ontents of words using Description Logics. In the next ection, we briefly describe the components of the natural anguage system with espacially emphasis to the knowl-dge representation system. Additional requirements for he representation of lexical semantics are outlined and .re discussed in the final chapter.

! Word Meanings and Description Logics

n order to formulate some basic requirements, a rep-esentation language based on a Description Logic for-ıalism has to meet, we have analyzed in [Burkert and 'orster,1992] and [Burkert,1992] a number of simple dic-ionary definitions.

2.1 Concrete Objects

In dictionaries, the meaning of concrete objects are expressed by nominal definitions. The investigations of nominal definitions reveals that in addition to the subsumption relation other relations occur quite frequently. Among these relations are:

1. relations between an object and its parts, e.g. a bicycle has two wheels
2. relations between an object and its uses or function, e.g. a bicycle is used for riding
3. relations between an object and its properties, e.g wheel is an object with property round.
4. *typical* relations between an object and its parts, properties, uses or function

For the representation of relations of type 1 – 3 *qualifying number restrictions* [Hollunder and Baader,1991] are an adequate solution. Typical relations between an object and its parts, properties, uses or function can be transformed into Description Logics extended by a default formalism[1]. These relations are not part of the representation formalism and have to defined prior to more specific relations and concepts. We have to decide which concepts and relations we consider to be the basic concepts and relations. Then the description of new concepts and relations can be based on this ontology.

A bicycle is a two-wheeled machine for riding [EDE,1976]

The definition of *bicycle* could be transformed into Description Logic by:

```
(defprimconcept bicycle
   (and machine
        (exact 2 part wheel)
        (some function riding)))
```

2.2 Representation of Activities and Events

A source for the meaning of activities and events are verb definitions in dictionaries. The lexical semantics of verbs

[1]E.g. see the publications in [Bajcsy,1993]

strongly depends on the fillers of so-called *case relations* linking the verb to its arguments. Since case relations often have exactly one filler, the notion of a role can be restricted to single valued roles, subsequently called attributes. The introduction of attributes allows further concept description operations, e.g., attribute chains, equality of attributes or attribute chains, etc.

> *to ride is to travel on a horse, or in a carriage, or on a bicycle* [EDE,1976]

The definition of *to ride* suggests, that riding is a kind of *travelling*. This relation between *to ride* and *to travel* resembles hyponymy between nouns. The hyponymy relation between concepts representing verbs is called *troponymy* [Fellbaum,1990]. We suggest, that the use of the subsumption relation for the representation of the troponymy is an adequate solution. The difference between a verb and its troponym can often be expressed by restricting one or more of its case relations. In this example, the concept *to ride* has a case relation that can be filled by objects of type *horse*, *carriage* or *bicycle*.

In addition to the troponymy relation and the case relations other relations, including temporal relationships, can be used to describe verb meanings. Methods for reasoning with temporal relationships are not supported by standard Description Logics. It seems more natural, to integrate such relationships directly, with reference to a given concrete domain. An approach therefore is given in [Baader and Hanschke,1991]. Based on these ideas, relationships between time intervals can be described and used to enhance the specification of concepts representing verbs.

2.3 Drawing Inferences outside the Scope of Description Logics

In section 2.1 we did not mention that relations between an object and its parts are partial transitive relations. Consider the merological relations `(part house door)` and `(part door handle)` [Cruse,1986]. In this context, it is obvious that `part` is not transitive. In the context `(part car exhaust)`, `(part exhaust catalyst)` the transitive conclusion `(part car catalyst)` seems to be valid.

In standard terminological representation systems relationships between concepts are restricted to binary relations. On the other hand there is often a need in applications to represent arbitrary n-ary relations, for example to represent spatial relationships like `between(x,y,z)`, meaning that an concrete object `x` is located between the concrete objects `y` and `z`.

However, these inferences can not be drawn by Description Logics because they are outside the scope of classification or subsumption. These inferences can be carried out by a separate component for the representation of rule-like formulas in which inferences over structures in the assertional component are drawn. In chapter 4 we will give a short summary about our approach.

3 Embedding TED&ALAN in a Natural Language System

To process German natural language input we provide several analyzing tools which enables us to build up common knowledge bases and natural language interfaces. Because of the different kinds of natural language input, e.g., user input or machine-readable texts, a high degree of flexibility is necessary. This leads to combine the following software modules into a *"text processor"*.

- **Lex-Module**: analyzes the lexical and morphological features of words
- **Syn-Module**: analyzes the syntactic structure of sentences
- **Sem-Module**: transforms the syntactic structure of a sentence into a semantic representation which is mapped into an knowledge base query language.

The knowledge representation system TED&ALAN [Forster and Burkert,1991] which is based on the $\mathcal{NTF}$ formalism [Nebel,1990] is the kernel of the semantic module.

A consequence of our investigations in the area of lexical semantics was the extension of the expressive power of our knowledge representation system by *attributes*, *qualifying number restrictions*, *concrete domains* and *rules* [Forster and Novotny,1994].

A first version of the text processor is used for the development of a natural language system for answering questions about geographics. The lexicon consists of concept descriptions representing the description of words. To get a (correct) answer of a natural language query – the query is transformed into a semantic representation which is translated into the query language of TED&ALAN. The following example of the natural language processing system[2] demonstrates the transformation of a simple German question into the query language of TED&ALAN.

```
=> Wie heisst die Hauptstadt der USA?
   [What is the capital of the USA?]
;;Semantic representation
   (:the
     ((:fillers-of (:attribute capital)
                   (:instance usa))))
;;ALAN-Query:
   (get-value usa capital)
;;Answer
   (washington_dc)
```

4 Description Logics and Rules

In chapter 2.3 we proposed to combine Description Logics with a rule based formalism for drawing inferences outside the scope of Description Logics. A basic requirement for the integration of rules into a representation

[2]The natural language system and its components are implemented in Common Lisp

system based on Description Logics is the existence of a well-defined semantics of the rule-based inferences. This is necessary to understand the behaviour of the whole system. Therefore, we restrict our rules to expressions based on first order logic. Using logical implications as rules enables to define the semantics of the rule inferences by means of the *predicate calculus*. We restrict our rules to Horn clauses. Some of the predicates used here may correspond to concept names or role names defined in the TBox. Concept names are considered as symbols of unary predicates, role names as symbols of binary predicates.

This example demonstrates the integration of *partial transitive* relations. The user-defined rule restricts the transitivity of the part-of relation to Catalysts, Exhausts and Cars.

```
        (Car car#1) )
        (Exhaust exhaust#1)
        (Catalyst catalyst#1))
        (part-of catalyst#1 exhaust#1)
        (part-of exhaust#1 car#1)
        (House house#1)
        (Door door#1)
        (Handle handle#1)
        (part-of handle#1 door#1)
        (part-of door#1 house#1)
The user-defined rule:
part-of(x,y) ← Catalyst(x),
               Car(y), Exhaust(z),
               part-of(x,z),
               part-of(z,y).
```

This rule enables the derivation of (part-of catalyst#1 car#1) and prohibits to derive (part-of handle#1 house#1). Our approach of integrating rules into a terminological representation system can lead to inconsistencies. In [Forster and Novotny,1994] we have shown a solution which allows to detect inconsistencies. However, the problem of how we can manage inconsistencies is still under discussion.

5 Summary and Discussion

The examination of dictionary definitions revealed that Description Logics are partially suitable for applications in lexical semantics. We have combine several analyzing tools into a text processor including a semantic processing module. The kernel of this module is the knowledge representation system TED&ALAN. A first version of the text processor is used for a natural language application. Within such applications we are able to get a feedback of the theoretical work done in the area of Description Logics and to formulate new basic requirements which a representation language, based on a Description Logic, has to meet to be useful in "real world applications". Drawing inferences outside the scope of Description Logics is one of these requirements. We propose that these inferences can be carried out by a separate component which enables the combination of Description Logics and rule-based formalisms.

References

[Baader and Hanschke, 1991] Franz Baader and Philipp Hanschke. A scheme for integrating concrete domains into concept languages. Technical Report RR-91-10, Deutsches Forschungszentrum für Künstliche Intelligenz, 1991.

[Bajcsy, 1993] Ruzena Bajcsy, editor. *International Joint Conference on Artificial Intelligence*, volume 1. IJCAI, Morgan Kaufmann Publishers, Inc., 1993.

[Burkert and Forster, 1992] Gerrit Burkert and Peter Forster. *Lexical Semantics and Knowledge Representation*, chapter Representation of Semantic Knowledge with Term Subsumption Languages, pages 75–84. Lecture Notes in Artificial Intelligence. Springer-Verlag, 1992.

[Burkert, 1992] Gerrit Burkert. Representation of semantic knowledge with term subsumption languages. In E. Viegas P. Saint-Dizier, editor, *Computational Lexical Semantics*, pages 47–68, Toulouse, France, January 1992. IRIT.

[Cruse, 1986] D.A. Cruse. *Lexical Semantics*, chapter 7, pages 157–180. Cambridge University Press, 1986.

[EDE, 1976] EDE. *Elementary Dictionary of English.* Langenscheidt-Longman GmbH, München, 1976.

[Fellbaum, 1990] Christiane Fellbaum. English verbs as a semantic net. *International Journal of Lexicography*, 3:278–301, 1990.

[Forster and Burkert, 1991] Peter Forster and Gerrit Burkert. Term subsumption languages and lexical semantics. In Christof Peltason, Kai von Luck, and Carsten Kindermann, editors, *Proceedings of the Terminological Logic Users Workshop, KIT Project, Technische Universität Berlin*, pages 91–96, 1991.

[Forster and Novotny, 1994] Peter Forster and Bernd Novotny. Integration of rule inferences into a terminological component. In *Proceedings of the 8th International Symposium on Methodologies for Intelligent Systems*, Charlotte, North Carolina, USA, October 1994. Oak Ridge National Laboratoy. forthcoming.

[Hollunder and Baader, 1991] Bernhard Hollunder and Franz Baader. Qualifying number restrictions in concept languages. Technical Report RR-91-03, Deutsches Forschungszentrum für Künstliche Intelligenz, 1991.

[Nebel, 1990] Bernhard Nebel. *Reasoning and Revision in Hybrid Representation Systems.* Lecture Notes in Artificial Intelligence. Springer Verlag, 1990.

Using Description Logics for Disambiguation in Natural Language Processing*

J. Joachim Quantz and **Birte Schmitz** and **Uwe Küssner**
Technische Universität Berlin, Projekt KIT-VM11, FR 5-12, Franklinstr. 27/28, D-10587 Berlin
{jjq,birte,uk}@cs.tu-berlin.de

1 Preferential Interpretation by Minimizing Exceptions

One of the major problems in Natural Language Processing (NLP) is to determine the informational content of an expression in the particular context in which it is used. Part of the informational content is determined by the sign system the expression belongs to. These constraints on signs as elements of the system are usually not fine-grained enough, however, to completely determine the informational content of a sign in use, i.e. a particular utterance of an expression. In order to determine this informational content, one has to take into account contextual information, i.e. one has to disambiguate or interpret the expression.

Note that the ambiguity of NL expressions is not an accidental deficiency of natural languages, but is rather a consequence of their efficiency—natural languages provide a "small" set of signs in order to express a "large" number of propositions, i.e. they can be used in a "large" number of different situations.

Note further that ambiguities of NL expressions arise wrt different types of information, e.g. the syntactic structure of an expression, its semantic content, its referent, its communicative function, etc. In [14] it is shown that such different kinds of ambiguities can all be accounted for with a uniform disambiguation strategy. The main hypotheses underlying this disambiguation strategy are the following:

1. The contextual information relevant for disambiguation is of quite *heterogeneous* nature and comprises syntactic, semantic, conceptual, encyclopedic, and pragmatic information.
2. The contextual information to be used for disambiguation does not impose strict constraints which would rule out alternative interpretations completely, but rather induces *preferences* for particular interpretations.
3. The preferences induced by different pieces of contextual information vary wrt their respective *relevance*.
4. When disambiguating an expression one has to combine all the preferences stemming from the context in order to determine an overall global preference. To do so a *homogeneous representation* of contextual information and the corresponding preferences is advantageous.

As illustrated in [11], DL systems provide adequate representation formats for such a homogeneous interpretation of contextual information. The preference rules can then be represented by using a Preferential Default Description Logic (PDDL) based on *weighted defaults* [10]. Within this framework disambiguation can be formally characterized as preferential interpretation by minimizing exceptions [7].

In this paper we will illustrate how Description Logics are used in the disambiguation process. In particular, we will show that the following functionality of DL systems is essential in our application:

1. We need a *rather expressive* DL containing, for example, role composition, role inversion, role value maps (at least for features) and qualifying number restrictions. We will use definitions and rules to represent all kinds of relevant information (e.g. syntactic, semantic, etc.) in a hierarchical, object-centered manner (examples will be given in Sections 5 and 6).
2. The *distinction between types (concepts and roles) and objects* is essential in our application. We will discuss this issue in detail in the next section.
3. We need the epistemic **k** operator [3] in queries, as will be illustrated in Section 5.
4. Instead of having just a single, monolithic ABox, we partition our ABox into situations, giving rise to *situated descriptions* of the form 'o :: c in s'. We will discuss this DL extension in Section 3 and illustrate its usefulness in Sections 5 and 6.
5. Finally, we need a default extension of DL based on *weighted defaults*. So far we use the framework of PDDL as presented in [10] and sketched in Section 4. The example illustrating the use of weighted defaults given in Section 6 indicates that a weaker semantics, more in the spirit of Reiter's Default Logic, might be sufficient as well.

*The project KIT-VM11 is supported by the German Federal Minister of Research and Technology (BMFT) under contract 01 IV 101Q8.

The main motivation for using DL as a framework is that it provides a sound theoretical foundation, as well as existing system technology. Obviously any implemented formalism providing the above functionality could be used to implement our approach to disambiguation as well.

2 Objects, Types, and Epistemic Operators

DL systems have been used in a number of NLP systems for representing conceptual and encyclopedic information. Due to the similarity between DL and the *typed feature structures* underlying many unification grammars [2], it is also straightforward to represent syntactic information within DL [8]. To do so we need a rather expressive DL, however, containing, for example, role composition, role inversion, role value maps (at least for features) and qualifying number restrictions.

The main difference between DL and typed feature structures is the distinction between types (i.e. concepts and roles) and objects, which is fundamental in DL but missing in typed feature structures. There are several relevant aspects of such a distinction:

1. Objects can be described by using complex descriptions and are therefore "structured", whereas atomic types in typed feature structures are not.
2. Objects can be described incrementally, whereas the atomic types used in typed feature structures are usually static.
3. Objects can be used to make information contained in a typed feature structure "persistent". Instead of specifying complex feature paths to refer to a particular embedded feature structure, one can use the object name as a more convenient pointer.
4. Persistency allows retrieval of instances for a given type. In contrast, there are theoretically infinitely many subtypes for most types.
5. Based on objects, epistemic operators can be defined [3]. These operators allow the distinction between knowing the type of a filler and actually knowing the filler, which can be used to control the inference process [7; 8] (see Section 5).
6. The objects can be used to define exceptions to defaults (see Section 4) and are thus the basis for the integration of weighted defaults.

The examples given in Sections 5 and 6 will show how the distinction between objects and types is used in our application. Before presenting these examples, however, we will briefly sketch two extension of DL used in them, namely situated descriptions and weighted defaults.

3 Situated Descriptions

In order to represent the alternative interpretations of an NL expression explicitly, we use *situated descriptions*. Instead of having descriptions 'o :: c' being universally true in the whole ABox, the ABox is partitioned into various situations. A situated description has the form 'o :: c in s' and means that 'o' is an instance of 'c' in the situation 's' and in all its extensions. *Extension* of situations ($s_1 \leq s_2$) is used to represent information common to several situations. Thus when saying that s_2 extends s_1 we are saying that '$\Gamma \models$ o :: c in s_1' implies '$\Gamma \models$ o :: c in s_2'.

Situated descriptions are used in three different but related ways:

1. they can be used to trigger *reasoning by cases* from the outside of the DL system. Suppose we know that 'o_1 :: f:$o_2 \sqcup$ f:o_3 in s_1'. We can then create two situations s_2 and s_3 with

$$\begin{array}{lcl} s_1 & \leq & s_2 \\ s_1 & \leq & s_3 \\ o_1 & :: & \text{f:}o_2 \text{ in } s_2 \\ o_1 & :: & \text{f:}o_3 \text{ in } s_3 \end{array}$$

   We can then check whether s_2 or s_3 are inconsistent, whether a description is valid both in s_2 and s_3, etc.
2. In some cases local ambiguities can be resolved by showing inconsistency with the context. Situated descriptions can be used to implement this disambiguation process via *backtracking*. In the above example we can start with situation s_1, then create situation s_2 as an extension to s_1. If 'o_1 :: f:o_2 in s_2' is inconsistent we can backtrack to situation s_1. The parser described in [8] makes heavy use of this form of backtracking (see Section 5).
3. Finally, we can compare different situations wrt their exceptionality, thereby obtaining a *preference ordering* on situations. We will give an example in Section 6.

4 Preferential Default Description Logics

In order to represent the preference rules used for disambiguation, weak constraints are needed in addition to the hard constraints provided by DL. As illustrated in [11], weighted defaults of the form $c_1 \leadsto_n c_2$ allow an adequate modeling of preference rules and their respective relevance. Such a weighted default is usually abbreviated by δ and c_1 is called its *premise* (written δ_p), c_2 its *conclusion* (δ_c), and the natural number n its *weight* ($w(\delta)$)—the higher the weight the more relevant the default.

The main characteristic of weighted defaults is that they induce an ordering on multisets of defaults. Note that this kind of priorization differs considerably from the lexicographic orderings used in most priorized Nonmonotonic Logics and their application to Description Logics [9; 1]. For lexicographic orderings a strong default overwrites an arbitrary number of weak defaults, whereas for weighted defaults, several weak defaults taken together can overwrite a strong default. This is exactly what is needed for the aggregation of the local preferences to a global preference in the interpretation process of NL utterances.

In [10] a formal semantics for weighted defaults is specified based on standard techniques from Nonmonotonic Reasoning. The basic idea is to qualitatively minimize the exceptions to defaults. The *exceptions* to a default δ in a model M are defined as

$$E_M(\delta) \overset{\text{def}}{=} \{o \in \mathcal{O} : [\![o]\!]^{\mathcal{I}} \in [\![\delta_p]\!]^{\mathcal{I}} \wedge [\![o]\!]^{\mathcal{I}} \notin [\![\delta_c]\!]^{\mathcal{I}}\}$$

where $\mathcal{O}$ is a set of object names. All exceptions to all defaults are added up and determine the *score* of a model:

$$\text{score(M)} \overset{\text{def}}{=} \Sigma_{\delta\in\Delta}(|E_M(\delta)| \cdot w(\delta))$$

Based on this score a preference ordering on models is defined:

$$M \sqsubset_\Sigma N \quad \text{iff} \quad \text{score(M)} > \text{score(N)}$$

Thus the $\sqsubset_\Sigma$-maximal models are models with minimal score. The preferential entailment relation $\Gamma; \Delta \mid\!\approx_\Sigma \gamma$ then holds iff all $\sqsubset_\Sigma$-maximal models of Γ are models of γ.

As shown in detail in [10], it is straightforward to prove that $\mid\!\approx_\Sigma$ satisfies all the properties of system **P** [4] and *Rational Monotonicity* [5]. Furthermore, $\mid\!\approx_\Sigma$ is decidable if the underlying DL is, as shown in [12], where an exponential algorithm for computing $\mid\!\approx_\Sigma$ is presented.

In Section 6 we will show how the ideas underlying the scoring and ordering of models can also be applied to score and order situations and thus interpretations.

5 DL-Based Parsing

We will now illustrate the use of a DL system in the analysis process by sketching a syntactic parser based on the DL system FLEX (for details see [8; 13]). The parser uses FLEX to check the consistency of possible combinations of signs. The linguistic knowledge about lexical forms and syntactic principles is explicitly modeled in FLEX. Using the standard mechanisms of *concept definitions* and *implication links* or *rules* we can straightforwardly represent the ideas of a *hierarchical lexicon* proposed, for example, in [6]. For illustration consider the following definitions and rules:

```
           noun  :=   maj:n & lex:+
             np  :=   maj:n & lex:-
           verb  :=   maj:v & lex:+
noun & nform:c   =>   exactly(1,comp_args) &
                      the(comp_arg1,det) &
                      case=comp_arg1.case &
                      gen=comp_arg1.gen
noun & nform:p   =>   no(comp_args)
           verb  =>   the(comp_arg1,np & case:nom)
    lexeme:frau  =>   noun & nform:c & gen:f
   lexeme:sehen  =>   verb & exactly(2,comp_args)
                      the(comp_arg2,np & case:acc)
```

Thus information is specified at the most general level possible. For example, all verbs have at least one argument, which is an 'np' in nominative case (namely the subject); all common nouns (nform:c) need an article, whereas pronouns (nform:p) cannot have one, etc.[1] Note that value restrictions are used to constrain the type of arguments, and that role value maps are used to express that certain features have identical values.

In addition to these lexical entries we have to model syntactic principles, which contain constraints for the combination of signs. *Immediate Dominance (ID) Schemata*, for example, characterize valid phrase structures and are modeled like this:

```
comp_struct  :=  some(head_dtr) &
                 functor_dtr=head_dtr &
                 no(adj_dtr) & no(filler_dtr)
        id1  :=  comp_struct &
                 the(head_dtr,np & nform:c) &
                 some(comp_dtr1) & no(comp_args)
        id2  :=  comp_struct &
                 the(head_dtr,noun & nform:c) &
                 no(comp_dtrs) & some(comp_arg1)
        id3  :=  comp_struct &
                 the(head_dtr,verb) &
                 no(comp_args) & mc:-
```

The standard approach to parsing in Unification Grammars is to view lexical entries and syntactic principles as huge disjunctions, and to use all constraints for eliminating alternatives until only the valid parses remain [2]. In order to achieve this sort of parsing we would need a DL system performing complex reasoning by cases. Instead, we rather trigger the case reasoning from the outside and use FLEX to check the consistency of alternative parses.

The basic idea of the parser in [8] is to take a sign which is a functor and to look for suitable arguments. In doing so the **k**-operator is used to distinguish between signs already used in a phrase and signs which are still available for "phrase construction":

```
new_phrase(Sit,FinSit) :-
    Sign ?: functor & no(k(inv(dtrs))) in Sit,
    select_id_schema(Sign,Sit,Phrase,NewSit),
    complete_arguments(Sign,NewSit,NextSit).
```

Note that the first predicate called in 'new_phrase' is a query to the underlying FLEX system. Since 'Sign' is a variable, FLEX will backtrack all object names satisfying the concept description on the right-hand side of the query. Note further that we use the constraint 'no(k(inv(dtrs)))' to determine whether a sign has already been used in phrase construction, i.e. whether we already have a phrase which has this sign as its daughter.

Selection of an ID schema is realized in a rather naive and simple way—we just take an ID schema and try to create a new phrase as an instance of this schema, where the feature 'functor_dtr' is filled by 'Sign'.

```
select_id_schema(Sign,Sit,Phrase,NewSit) :-
    id_schema(ID),
```

[1]Note that these rules have been considerably simplified and are used for illustrative purpose only. Some of them are thus not completely adequate from a linguistic point of view.

```
        extend_sit(Sit,NewSit),
        Phrase :: ID & functor_dtr:Sign in NewSit.
```

Information about existing ID schemata thus has to be encoded as facts of the form 'id_schema(id1)', etc. The predicate 'extend_sit(Sit,NewSit)' is used to tell the FLEX system to create a new situation which is an extension of the current situation, i.e. formally we have 'Sit $\leq$ NewSit'. Note that no further knowledge about the actual modeling of ID schemata is used in the parser except for the fact that each ID schema has a 'functor_dtr'.

Note further that the last predicate in the 'select_id_schema' is a FLEX tell. It will fail if the information known about the functor cannot be unified with the information required by the ID schema for the filler of 'functor_dtr'.

In order to complete the arguments of the functor, the parser checks for each argument feature ArgFeat whether an argument is required (some(ArgFeat)) but not yet specified (no(k(ArgFeat))). If so, 'find_arg' looks for such an argument and enters it as filler for ArgFeat. Then the remaining arguments are completed.

```
    complete_arguments(Functor,Sit,FinSit) :-
        arg_feature(ArgFeat),
        Functor ?: some(ArgFeat) & no(k(ArgFeat)) in Sit,
        !,find_arg(Functor, Sit,ArgFeat,NewSit),
        complete_arguments(Functor,NewSit,FinSit).
    complete_arguments(_,Sit,Sit).
```

Again, facts specifying the arguments used in the fragment are needed, e.g. 'arg_feature(comp_arg1)'.

If an argument is required it has to be filled, therefore the Cut. Thus the recursion terminates successfully only when all required arguments are actually filled. Note that the only information about argument structure needed by the parser are facts of the form 'arg_feature(comp_arg1)' for all argument features.

To find an argument the parser looks for a sign which has not yet been used for phrase building and asserts it as filler for the argument feature. Again, if unification is not possible due to conflicting constraints (e.g. agreement), the FLEX tell will fail.

```
    find_arg(Functor,Sit,ArgFeat,FinSit) :-
        Arg ?: sign & no(k(inv(dtrs))) in Sit,
        extend_sit(Sit,FinSit),
        Functor :: ArgFeat:Arg in FinSit.
    find_arg(Functor,Sit,ArgFeat,FinSit) :-
        new_phrase(Sit,NewSit),
        find_arg(Functor,Sit,ArgFeat,FinSit).
```

The second clause is needed to create a required argument which has not yet been build up. In this case 'new_phrase' is called to create a new potential argument.

Note that failure of a FLEX tell causes backtracking. Thus if telling a potential argument fails, another potential argument is backtracked by the FLEX query. If no argument is found at all 'complete_arguments' will fail and another ID schema is selected. In the course of backtracking the information remaining valid is determined by the situation currently used.

The main motivation for presenting this example is to show how a DL system can be used to

1. represent lexical information hierarchically;
2. to represent information about selectional restrictions via value restrictions;
3. to represent agreement information via role value maps;
4. to represent syntactic principles;
5. to use these declarative representations for consistency checking;
6. to retrieve signs satisfying a given description (possibly containing the **k** operator;
7. to represent alternative parses (corresponding to the charts in a chart parser) and to control backtracking via situated descriptions.

Thus the only task left for the parser is to control the process of phrase construction, whereas the DL system guarantees that the relevant lexical and syntactic constraints are taken into account. As result the parser returns a situation containing the complete phrase structure tree; alternative parses can be obtained by backtracking. The objects described in these situations correspond to the nodes in traditional phrase structure trees, each node will be explicitly represented by an object.

Given the two possible parses represented in two different situations we can use preference rules represented as weighted defaults to choose the preferred interpretation. We will illustrate this by considering an example of anaphora resolution in the next section.

6 Preferential Interpretation

Finally, we will now illustrate the use of weighted defaults in the disambiguation process. Consider the example taken from [11]:

1. Fortgeschrittene Systeme erkennen die Information in der Form, in der $\underline{\text{sie}_1}$ generiert wird. $\underline{\text{Sie}_2}$ integrieren $\underline{\text{sie}_3}$ in das gespeicherte Wissen.
2. Advanced systems perceive information in the form in which $\underline{\text{it}}$ is generated. $\underline{\text{They}}$ integrate $\underline{\text{it}}$ into the stored knowledge.

The task of anaphora resolution consists in determining the antecedent of the anaphoric pronouns 'sie$_1$', 'sie$_2$', and 'sie$_3$', i.e. the phrases they stand for.

The following defaults are used for anaphora resolution:

δ_1 : anaphor	$\leadsto_{1000}$	num = ant.num &
		gen = ant.gen
δ_2 : anaphor	$\leadsto_{60}$	the(ant,subject)
δ_3 : anaphor	$\leadsto_{20}$	the(ant,topic)
δ_4 : anaphor	$\leadsto_{80}$	sem-role=ant.sem-role
δ_5 : anaphor	$\leadsto_{70}$	the(ant,non-adjunct)

Furthermore, we assume that the following information has already been derived in the interpretation process:

sie_1	::	anaphor & sem-role:af & gen:f & num:sg & the(ant,oneof([systeme, information,form,relpron])) in s_1
systeme	::	subject & topic & non-adjunct & sem-role:ag & gen:n & num:pl in s_1
information	::	non-adjunt & sem-role:af & gen:f & num:sg in s_1
form	::	sem-role:loc & gen:f & num:sg in s_1
relpron	::	sem-role:loc & gen:f & num:sg in s_1

We can then use information about the possible antecedents to generate the following situations (all extending s_1)

sie_1	::	ant:systeme in s_2
sie_1	::	ant:information in s_3
sie_1	::	ant:form in s_4
sie_1	::	ant:relpron in s_5

To compare these situations we use the ideas underlying the scoring and ordering of models (see Section 4). We define the exceptions to a default δ in a situation s as:

$$E_s(\delta) \stackrel{\text{def}}{=} \{o \in \mathcal{O} : \Gamma \models o :: \delta_p \text{ in } s \wedge \Gamma \not\models o :: \delta_c \text{ in } s\}$$

We then define the score of a situation s as

$$\text{score(s)} \stackrel{\text{def}}{=} \Sigma_{\delta \in \Delta}(|E_s(\delta)| \cdot w(\delta))$$

And finally, we define a preference ordering $\sqsubset_\Sigma$ on situtation:

$$s_1 \sqsubset_\Sigma s_2 \quad \text{iff} \quad \text{score}(s_1) > \text{score}(s_2)$$

In the above example we thus obtain

situation	exceptions	score
s_2	δ_1, δ_4	1080
s_3	δ_2, δ_3	80
s_4	$\delta_2, \delta_3, \delta_4, \delta_5$	230
s_5	$\delta_2, \delta_3, \delta_4, \delta_5$	230

where a default δ_i is put in the 'exceptions' column iff '$sie_1 \in E_s(\delta_i)$'. We thus prefer the interpretation represented in s_3 according to which 'information' is the antecedent of 'sie_1'.

Note that '$\Gamma; \Delta \approx_\Sigma sie_1$ :: ant:information' also holds given the above modeling. But here we have used the defaults only to choose between alternative interpretations and not as inference rules for deriving additional information.[2] The exact relationship between this use of defaults and the preferential entailment relation is still under investigation at the moment.

7 Conclusion

We have illustrated how Description Logics can be used to support the disambiguation process in Natural Language Processing. In addition to the standard functionality provided by an expressive DL (role composition and inversion, feature value maps, qualifying number restrictions), we had to use epistemic operators, situated descriptions, and weighted defaults in our application.

It should be noted that the DL system is used for performing consistency checking and retrieval, but that an additional program is needed (e.g. the parser) to control the overall interpretation process. Due to the Prolog interface of FLEX this integration is straightforward.

It would be interesting to see whether the use of epistemic operators in the parsing process or the use of weighted defaults for ordering situations could be generalized to configuration or diagnosis tasks. It seems that parsing and disambiguation are just special cases of these general problem classes.

[2]It is obvious that the scoring of situations can be performed by using standard DL functionality (retrieving instances of left-hand sides of defaults and checking whether they are instances of the corresponding right-hand sides).

References

[1] F. Baader, B. Hollunder, "How to Prefer More Specific Defaults in Terminological Default Logic", *IJCAI-93*, 669–674

[2] B. Carpenter, *The Logic of Typed Feature Structures*, Cambridge: Cambridge University Press, 1992

[3] F.M. Donini, M. Lenzerini, D. Nardi, A. Schaerf, W. Nutt, "Adding Epistemic Operators to Concept Languages", *KR-92*, 342–353

[4] S. Kraus, D. Lehman, M. Magidor, "Nonmonotonic Reasoning, Preferential Models and Cumulative Logics", *Artificial Intelligence* **44**, 167–207, 1990

[5] D. Makinson, "General Patterns in Nonmonotonic Reasoning", to appear in D. Gabbay, C. Hogger, J. Robinson (eds), *Handbook of Logic in Artificial Intelligence*, Oxford: Oxford University Press, in print

[6] C. Pollard, I.A. Sag, *An Information Based Syntax and Semantics*, Vol. I Fundamentals, Stanford: CSLI Lecture Notes 13, 1987

[7] J.J. Quantz, "Interpretation as Exception Minimization", *IJCAI-93*, 1310–1315

[8] J.J. Quantz, "An HPSG Parser Based on Description Logics", *to appear in COLING-94*

[9] J.J. Quantz, V. Royer, "A Preference Semantics for Defaults in Terminological Logics", *KR'92*,294–305

[10] J.J. Quantz, M. Ryan, *Preferential Default Description Logics*, KIT-Report 110, Technische Universität Berlin, 1993

[11] J.J. Quantz, B. Schmitz, "Knowledge-Based Disambiguation for Machine Translation", *Mind and Machines*, 4, 39–57, 1994

[12] J.J. Quantz, S. Suska, "Weighted Defaults in Description Logics—Formal Properties and Proof Theory", *submitted for publication*

[13] J.J. Quantz, G. Dunker, V. Royer, "Flexible Inference Strategies for DL Systems", *International Workshop on Description Logics*, 1994

[14] B. Schmitz, J.J. Quantz, *Defaults in Machine Translation*, KIT Report 106, Technische Universität Berlin, 1993

Construction of a Domain Model for Speech Translation

Manfred Gehrke
ZFE ST SN 54
Siemens AG
Otto-Hahn-Ring 6
81730 München
email: Manfred.Gehrke@zfe.siemens.de

The goals of VERBMOBIL are research in the area of speech translation and the development of a prototyp for this task[1]. In this project the KIT-group of TU Berlin and Siemens have the task of providing the domain model of the application, i.e. fixing the date for a meeting, where the participants speak different languages. As representation formalism BACK is used.

Amongst the services the domain modelling has to provide for are:

Selectional Restrictions: Though speaker independent recognition of continous speech has made an enormous progress in the last years, even the best recognizers still produce a lot of valid, but meaningless readings of an utterance due to the language model (mostly a statistical bigram model) of the recognizer. Not all of those readings can be discarded by syntactic means. Therefore testing the selectional restrictions gains an additional importance. Checking the sorts of arguments, which are represented in a concept hierarchy, against the argument structure of verbs, relational nouns, etc. gives a distinctive reduction of the number of ambiguities even with a quite general concept hierarchy.

Semantic Construction and Evaluation: This task is also supported by checking the argument structure in order to determine the intended reading of (lexically and syntactically) ambiguous lexemes and utterances. Additionally the reference identification of pronouns and definite descriptions, the construction of a dialogue model and the determination of the event structure have to be supported.

Transfer and Generation: The determination of the intended reading of an utterance is also important for the transfer into the target language in order the select the adequate words and constructions.

Generally spoken, the task of domain modelling supports the disambiguation of an utterance to such an extent that a more or less unambiguous translation is achieved. "More or less" because cases, where the same type of ambiguity appears in the target language need not be disambiguated.

[1]The project is funded by the Federal Ministery of Research and Technology and includes more than 30 mostly german universities, research institutes and companies.

The domain model provides a moderately deep conceptual modelling of the application domain to capture the translation-relevant distinctions. The modelling is based on following considerations:

- The domain model is the only source of conceptual knowledge in the system. This includes that the information on selectional restrictions are drawn from this source.
- The modelling is intended to be reusable, so that incrementally a broad coverage may be achieved.
- The ambiguity of lexemes (words) will be resolved by a mapping from lexemes to concepts, which is based on a access tree to the concept base and which takes into account the local and global context of the lexeme.
- Using the role hierarchy of BACK the modelling depends to a great extent on the roles, so that concepts are just configurations of roles and their fillers.

1 Modelling Situations

At present the domain modelling is concentrated on the representation of situations, i.e states and events, since with these concepts verbs and a lot of nouns can be represented. To represent situations basic ingredients are roles to describe temporal constituency, thematic relations and lexical semantic relations as well as roles to distinguish between semantic clusters, as e.g. situations of information exchange, agreement or disagreement, movement, etc.

Following a proposal of M. Egg and M. Herweg[3] the modelling of temporal constituency - the distinction between state, events etc. - is based on following features:

interval_based : A situation with this role must hold at least at two different points of time.

bounded : A situation is temporaly limited.

telic : A situation describes a change of state.

punctual : A situation takes place at a point of time.

Representing these features as roles yields

int_bas	:< domain(situation) and range(boolean) type feature.
bounded	:< domain(situation) and range(boolean) type feature.
telic	:< domain(situation) and range(boolean) type feature.
punctual	:< domain(situation) and range(boolean) type feature.

with which aspectual classes are defined as below

state	:= situation and bounded:no and telic:no and punctual:no.
occurrence	:= situation and int_bas:yes.
event	:= occurrence and bounded:yes.
intergressive	:= event and telic:no.
change	:= event and telic:yes.

With these roles temporal constituency is sufficiently modelled at the conceptual level. Changing the type of temporal constituency, as it is e.g. necessary with progressive form is performed during parsing the utterance by an action that is analogous to raising in syntax[2].

Under certain circumstances, especially when there is an ambiguity in case marking, the determination of thematic roles causes some problems. In Dowty's approach to the definition of **agent** and **patient** via proto-roles ([2]), which is used to overcome this problems, several features contribute to agenthood resp. patienthood, namely:

agent : volitional involvement, sentience causation, movement relative to the patient,

patient : undergoes a change of state, causally affected, incremental theme, stationary relative to the agent.

These features are represented as roles. From the argument structure it is just known, that if a constituent is subsumed by some argument position, then it has to be the agent resp. patient. The requirements on agenthood resp. patienthood are represented in the IBox of BACK. The most specific IBox object, i.e. that one with the most features for one of the thematic roles, is then selected.

2 Checking selectional restrictions

Besides augmenting lexemes with conceptual information a major service of domain modelling to syntactic processing is rejecting meaningless syntactic constructions. In this application the test of selectional restrictions is part of unification-based NLP system(LKP)[3]. From the point of view of the NLP system checking selectional restrictions is a test that answers with a concept or fails. It resembles in this respect to the *test*-procedure in DLs. There are two types of testing selectional restrictions:

[2]On the other hand the determination of the type of temporal constituency can support the decision whether a (temporal) prepositional phrase is either an adjunct or a modifier.

[3]The LKP (Linguistic Kernel Processor)[1] is a unification-based grammar system with acomparably broad coverage for german, both spoken and written language. It includes a Tomita-Parser and a "shake and bake"-style generator.

- a subsumption test between the sort of an argument position and the sort of the possible argument or between the sort of a phrase and the sort of an adjunct or modifier,
- checking whether a specific relation holds between two arguments, as it is case for semantically empty verbs such as *to be*, *to have* or between conjoined noun phrases.

In addition to these tests the information on the sorts is passed by the grammar rules from the lexical entries to the respective phrases.

At present the test of selectional restrictions is performed after the application of the parser, while the sortal information is disseminated during its run. This has the drawback that the analysis is not directly cancelled, when a sortal mismatch is detected. But this strategy opens the opportunity to apply further reasoning processes, as e.g. for interpreting metonymies, on sortally illegal analyses, if no correct analysis has been found. One of the next steps is a deeper integration of this test into the parser, namely to apply it, when it performs a reduce action, i.e. conjoins two constituents, while maintaining the possibility of the above mentioned further reasoning processes.

In a previous project the same approach was taken with a less expressive sortal hierarchy. Besides rejecting most meaningless sentences, it reduces both the number of readings and the parsing time by about 50 %.

References

[1] Block, H.U. and Schachtl, S., Trace & Unification Grammar, 14th International Conference on Computational Linguistics (COLING-92), 1991

[2] Dowty, D., Thematic Proto-roles and Argument Selection. In: Language, 67:3 (1991), pp. 547-619

[3] Egg, M. and Herweg, H. A Phase-theoretical Semantics of Aspectual Classes. Verbmobil Report IBM 11-2/94, 1994

Description Logics for Querying Databases

Alex Borgida
Dept. of Computer Science
Rutgers University
New Brunswick, NJ 08903

1 Motivation

DLs have shown themselves to be useful in application that involve information processing normally performed by standard database management systems (see [3] for a review). Of particular interest to us are several reports on the use of DLs as *query languages* for retrieving information [6; 2; 1; 7; 8] — information that may well come from a standard relational DBMS. This has been particularly fruitful in *data exploration* applications, where one or more users, who may not be fully familiar with the database contents, are looking for interesting facts, correlations, etc. The special abilities of DLs lead to a number of advantages:

- Detecting and reporting *incoherent queries*, which may occur in situations when the user is not fully familiar with the semantics of the stored data.
- *Query generalization*: having asked a query that returns no individuals, the system can systematically try to generalize the description until it results in a concept with a non-empty answer [1].
- *Query organization:* When teams of users explore databases over periods of time, it is useful to be able to organize the queries themselves so that one can find similar queries that have been asked in the past [7]. Moreover, users may record observations about the query and its answer, using "meta" objects.
- *Query formulation by refinement:* A user exploring the database can use the classification hierarchy of concepts to refine her queries [12; 13; 2].
- *Intensional query processing:* The general processing strategy of DL queries is to "classify" the query description with respect to the pre-computed views (or saved previous queries), and then only test their instances rather than processing the entire database [2; 11].
- *Schema browsing:* Since concepts in the schema as well as queries can be compared for subsumption, users can explore and discover previously unknown *generic* facts, such as coding standards or constraints in Software Information Systems [9].

2 Research issues and preliminary results

There are several issues that seem relevant in this context.

2.1 Expressive power

Since Codd's early papers on relational databases, a question normally asked about any query language is how expressive it is (recall the notion of "relational completeness"), especially in relation to first order predicate calculus, which is in some sense the "golden standard". If the DL has decidable subsumption, then of course we cannot expect it to be able to express all FOL queries. But what about the fully expressive DLs? Are the rewards for undecidability full expressive power?

Our recent results [4] suggest that this is not the case: concept definitions in even the most expressive non-recursive DLs can be translated into variants of Predicate Calculus (extensions include numeric quantifiers and infinitary disjunction) with *at most three variable symbols*, and these are know to be unable to represent certain simple notions, such as graphs with 4-subcliques.

This leads to the next interesting question: what is the most elegant or appropriate way to supplement DLs in a query interface, since users will simply abandon these systems if we do not provide enough expressive power.

2.2 Database coupling.

In a situation where information from a relational database is being accessed through a DL-based interface, we need to ask what is the most appropriate way to couple the KBMS and the DBMS. The standard approach is to represent the schema of the database directly as concepts in the KBMS. It is not unusual for the database schema not to correspond to the desired conceptual view of the users (the relational model is not a "semantic model"). When dealing with expressively limited DLs, this can lead to problems because we may not be able to define the conceptual view in terms of the base concepts. (For example, if the database has relation `PERSON(name, mother,father)`, but we want concept PEOPLE, with role `children`.)

For this reason, we suggest [5] coupling primitive concepts in the conceptual model with view definitions in the database. But our experience indicates that "loading" even a small database of 1000 individuals takes far too long, because facts need to be checked for consistency, and inferences need to be looked for (even if they are not present). For this reason, we have developed as part of the CLASSIC project, a system that converts most of the inferences made by the KBMS into a collection of SQL queries. As a result, we rely on the optimization facilities of existing DBMS to gain efficiency, while maintaining an object-centered view of the world with a substantive semantics and significantly different reasoning facilities than those provided by Relational DBMS and their deductive extensions. We address a number of optimization issues that arise in the translation process due to the fact that SQL queries with different syntax (but identical semantics) are not treated uniformly by current database management systems.

There are many unresolved issues left in this approach, particularly about how much information from the database should be loaded in the KBMS, and when, as well as propagating back changes/inferences from the KB to the DB.

References

[1] Anwar, T.M., Beck H. and Navathe S., "Knowledge Mining by imprecise querying: a classification-based approach", *Proc. 8th IEEE Data Engineering Conf.*, Tempe, AZ, February 1992, 622–630.

[2] Beck, H. W., Gala, S. K., and Navathe, S. B., "Classification as a Query Processing Technique in the CANDIDE Semantic Data Model," *Proc. 5th IEEE Data Engineering Conf.*, February 1989, pp. 572–581.

[3] Borgida, A., "A new look at the foundations and utility of Description Logics (or Terminological Logics are not just for the `Flightless Birds`)", Technical Report DCS TR 295, Rutgers University, June 1992.

[4] Borgida, A., "On the relationship between description logic and predicate calculus queries", Technical Report DCS-TR-295A, Dept. of Computer Science, Rutgers University, June 1993.

[5] A. Borgida, R. Brachman, "Loading data into description reasoners", *ACM SIGMOD Conf. on Data Management*, May 1993, Washington, DC, pp. 217 - 226.

[6] Borgida, A., Brachman, R. J., McGuinness, D. L., and Resnick, L. A. "CLASSIC: A Structural Data Model for Objects," *Proc. 1989 ACM SIGMOD Conf.*, June 1989, pp. 59–67.

[7] Brachman, R., P. Selfridge, L. Terveen, B. Altman, A. Borgida, F. Halper, T. Kirk, A. Lazar, D. McGuiness, L. Resnick, "Knowledge Representation Support for Data Archaeology", *Proc. International Conference on Information and Knowledge Management*, Baltimore, MD, November 1992, pp.457–464.

[8] M.
Buchheit, M. Jeusfeld, W. Nutt, M. Staudt, "Subsumption of queries to Object-Oriented Databases", *Proc. EDBT'94*, to appear in Springer Verlag LNCS Series.

[9] Devanbu, P., Brachman, R. J., Ballard, B. W., and Selfridge, P. G., "LaSSIE: A Knowledge-Based Software Information System," *Communications of the ACM*, 34(5), May, 1991.

[10] Lenzerini, M. and Schaerf, A., "Concept Languages as Query Languages", *Proc. AAAI'91*, pp. 471-476.

[11] Nebel, B. and C. Peltason, "Terminological Reasoning and Information Management", D. Karagianis editor, *Information Systems and Artificial Intelligence.*

[12] Patel-Schneider, P. F., Brachman, R. J., and Levesque, H. J., "ARGON: Knowledge Representation Meets Information Retrieval," *Proc. First Conf. on Artificial Intelligence Applications*, Denver, CO, December, 1984, pp. 280–286.

[13] Tou, F., M. Williams, R. Fikes, A. Henderson, T. Malone, "RABBIT: An intelligent database assistant", *Proc. AAAI'82.*

Subsumption for Semantic Query Optimization in OODB*

Domenico Beneventano°, Sonia Bergamaschi°, Claudio Sartori°

CIOC-CNR

Abstract

The purpose of semantic query optimization is to use semantic knowledge (e.g. integrity constraints) for transforming a query into an *equivalent* one that may be answered more efficiently than the original version. This paper proposes a general method for semantic query optimization in the framework of OODBs (Object Oriented Database Systems). The method is applicable to the class of conjunctive queries and is based on two ingredients: a description logic able to express both class descriptions and *integrity constraints rules* (IC rules) as types; *subsumption* computation between types to evaluate the logical implications expressed by IC rules.

1 Introduction

In database environment, semantic knowledge is usually expressed in terms of IC rules, that is *if then* rules on the attributes of a *database schema* (i.e., roughly a Tbox of a Terminological Knowledge Representation System (TKRS)). Informally, *semantic equivalence* means that the transformed query has the same answer as the original query on all databases satisfying the IC rules. The notion of semantic query optimization for relational databases was introduced in the early 80's by King [13; 14]; Hammer and Zdonik [10] independently developed very similar optimization methods. The key idea in [13; 14], as well as in [10], is that IC may not only be utilized to enforce consistency of a database but also to optimize user queries. During the last decade, many efforts have been made to improve this technique and to generalize it to deductive databases [17; 18; 19; 8].

More recently, some efforts have been made to perform semantic query optimization in OODBs [9; 11; 16; 6; 1; 2]. The main point is that OODBs provide a very rich type (class) system able to directly represent a subclass of integrity constraints in the database schema. By exploiting schema information as, for instance, inheritance relations between types (classes), it is possible to perform semantic query optimization.

This work represents an improvement and a generalization of the cited works in the database area for many aspects. Firstly, our method applies to OODBs and, with respect to previous works, we generalize the notion of a database schema as a set of rules expressing *inclusion statement* between general types. Secondly, we provide the theoretical framework (in term of *subsumption*) for developing a theory of semantic query optimization which includes the main query transformation criteria proposed in the database literature. Inclusion statements between concepts, recently proposed in [5] to express a TKRS, constitute a generalization of description logic perspective. This new perspective perfectly fits the usual database viewpoint. In fact, actual database schemata are given in terms of base classes (i.e. primitive concepts); further knowledge is expressed as IC rules. In particular, structural class descriptions are expressed as rules where the antecedent is a name of the class and the consequent is the class description. More generally, rules allow the expression of integrity constraints with an antecedent and a consequent which are types of the formalism. Since query languages for OODBs are more expressive than our formalism we, following [6], ideally introduce a separation of a query into a *clean* part, that can be represented as a type in our formalism, and a *dirty* part that goes beyond the ·type system expressiveness. Semantic optimization will be performed only over the clean part of a given query. The clean part of a query, in the following referenced as query, corresponds to the so-called conjunctive queries or single operand queries [12] in OODBs and is a *virtual class* (i.e. a defined concept).

The chosen strategy for optimization is the following. Prior to the evaluation of any query, we *compile*, once at all, the given schema (classes + IC rules), giving rise

*This research has been partially funded by the LOGIDATA+ project of the National Research Council (CNR).

° Dipartimento di Elettronica, Informatica e Sistemistica, Università di Bologna, Viale Risorgimento 2,I-40136 Bologna, Italy.

° Facoltà di Ingegneria, Università di Modena, Via G. Campi 213/A, I-41100 Modena, Italy.

e_mail:
{ dbeneventano, sbergamaschi, csartori }@deis.unibo.it

to an enriched schema obtained by adding (*all the new*) *isa relationships* which are logically implied by the original schema. The compilation process is based on the generation of the *semantic expansion* in *canonical form* (i.e. a form which permits to abstract from different syntactical representation of semantically equivalent types) of the schema types. Following the approach of [17; 19] for semantic query expansion, the semantic expansion of a type, say $EXP(S)$ permits to incorporate any possible restriction which is not present in the original type but is *logically implied* by the type and by the schema.

$EXP(S)$ is based on the iteration of this simple transformation: if a type implies the antecedent of an IC rule then the consequent of that rule can be added. Logical implications between these types (the type to be expanded and the antecedent of a rule) are evaluated by means of *subsumption computation* [4; 3; 2].[1]

At run time, we add to the compiled schema the query Q and activate the process again for Q: $EXP(Q)$, possibly including new *isa* relationships is obtained. If new *isa* relationships are found, it is possible *to move the query down* in the schema hierarchy: this is the main optimization achievement of the proposed method. The main points of our optimization strategy are: (1) $EXP(Q)$ is the *most specialized query* among the equivalent queries. Moreover, during the transformation, we compute and substitute in the query, at each step, the *most specialized classes* satisfying the query. (2) A filtering activity (*constraint removal*) is performed by detecting the *eliminable* factors of a query, that is, the factors logically implied by the query.

2 Complex Objects and IC Rules

The main extension of the description logic proposed in [2] is the generalization of the notion of a database schema as a set of rules which express *inclusion statement* between general types.

Let $\mathbf{D}$ be the countably infinite set of base-values $(d_1, d_2, \ldots)$ and $\mathbf{A}$ be a countable set of *attributes* $(a_1, a_2, \ldots)$. The set $\mathcal{V}$ of all *values* are built by the following rule: $v \rightarrow d \mid \{v_1, \ldots, v_q\} \mid [a_1\colon v_1, \ldots, a_p\colon v_p]$, with $p, q \geq 0$ and $a_1, \ldots, a_p$ distinct in $\mathbf{A}$. Let $\mathbf{B}$ be a countable set of base-type designators $(B, B', \ldots)$ and $\mathbf{C}$ be a countable set of names for *base class* $(C, C' \ldots)$, such that $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ are pairwise disjoint. A *path* p is a sequence of elements $p = e_1.e_2.\ldots.e_n$, with $e_i \in \mathbf{A} \cup \{\forall\} \cup \{\exists\}$. We denote with ϵ the *empty path*. $\mathbf{S}$ denotes the set of all *type* $(S, S', \ldots)$ over given $\mathbf{A}, \mathbf{B}, \mathbf{C}$, obtained according to the following syntax rule:

$$S \rightarrow \top \mid B \mid C \mid [a_1 : S_1, \ldots, a_k : S_k] \mid \forall\{S\} \mid \exists\{S\} \mid S \sqcap S' \mid (p : S)$$

[] denote the usual type constructor of tuple. Two different set constructors $\forall\{\}$, $\exists\{\}$ are introduced. $\forall\{S\}$ corresponds to the usual set constructor and denotes sets whose elements are *all* of type S; $\exists\{S\}$ denotes sets where *at least* one element is of type S. The construct $\sqcap$ holds for *conjunction*. The *path type* $(p\colon S)$ represents a short notation.

[1]The subsumption is similar to the *refinement* or *subtyping* adopted in OODBs [7; 15].

A *database schema* $\mathbf{R}$ is a subset of the cartesian product of a given type system $\mathbf{S}$. An element $R = (S^a, S^c)$ of $\mathbf{R}$ espresses a rule where S^a is the antecedent and S^c the consequent: $S^a \rightarrow S^c$. A rule R is universally quantified over all values $\mathcal{V}$: for all v, if v is of type S^a then v must be of type S^c. Given $\mathbf{R}$, we denote with $\mathbf{R}_\sigma$ the subset of $\mathbf{R}$: $\mathbf{R}_\sigma = \{R \in \mathbf{R} \mid \exists C \in \mathbf{C}, S^a = C\}$ and with σ a total function from $\mathbf{C}$ to $\mathbf{S}$:

$$\sigma(C) = \begin{cases} \top & \text{if } \nexists R \in \mathbf{R} : S^a = C \\ \sqcap_i S_i^c & \forall R_i \in \mathbf{R} : S_i^a = C \end{cases}$$

The subschema $\mathbf{R}_\sigma$ represent the classes and their structures and the function σ associates class names to their descriptions.

Let $\mathcal{I}_\mathbf{B}$ be the (fixed) standard interpretation function from $\mathbf{B}$ to $2^\mathbf{D}$. For a given $\mathcal{V}$, each type S is given a subset of values as its interpretation by the *interpretation function* $\mathcal{I}$ such that: $\mathcal{I}[\top] = \mathcal{V}$, $\mathcal{I}[B] = \mathcal{I}_\mathbf{B}[B]$, $\mathcal{I}[C] \subseteq \mathcal{V}$, $\mathcal{I}[S \sqcap S'] = \mathcal{I}[S] \cap \mathcal{I}[S']$ and

$$\mathcal{I}[\forall\{S\}] = \Big\{\{v_1, \ldots, v_p\} | v_i \in \mathcal{I}[S], 1 \leq i \leq p\Big\}$$
$$\mathcal{I}[\exists\{S\}] = \Big\{\{v_1, \ldots, v_p\} | \exists i, 1 \leq i \leq p, v_i \in \mathcal{I}[S]\Big\}$$
$$\mathcal{I}[[a_1\colon S_1, \ldots, a_p\colon S_p]] = \Big\{[a_1\colon v_1, \ldots, a_q\colon v_q]| \; p \leq q, v_i \in \mathcal{I}[S_i], 1 \leq i \leq p\Big\}$$

For the path types we have

$$\mathcal{I}[(p\colon S)] = \mathcal{I}[(e\colon (p'\colon S))] \text{ if } p = e.p' \text{ where}$$

$$\mathcal{I}[(\epsilon\colon S)] = \mathcal{I}[S],\ \mathcal{I}[(a\colon S)] = \mathcal{I}[[a\colon S]],$$
$$\mathcal{I}[(\forall\colon S)] = \mathcal{I}[\forall\{S\}],\ \mathcal{I}[(\exists\colon S)] = \mathcal{I}[\exists\{S\}]$$

An interpretation $\mathcal{I}$ is a *instance* of $\mathbf{R}$ iff for all $R \in \mathbf{R}$, $\mathcal{I}[S^a] \subseteq \mathcal{I}[S^c]$. Note that, in a instance $\mathcal{I}$: $\mathcal{I}[C] \subseteq \mathcal{I}[\sigma(C)]$, that is the objects of a class satisfy the class description.

2.1 Subsumption and Semantic Expansion

Subsumption w.r.t. $\mathbf{R}$: $S \sqsubseteq_\mathbf{R} S'$ iff $\mathcal{I}[S] \subseteq \mathcal{I}[S']$, for all instance $\mathcal{I}$ of $\mathbf{R}$.
Subsumption w.r.t. $\mathbf{R}_\sigma$: $S \sqsubseteq_\sigma S'$ iff $\mathcal{I}[S] \subseteq \mathcal{I}[S']$, for all instance $\mathcal{I}$ of $\mathbf{R}_\sigma$.

The *semantic expansion* $EXP(S)$ of a type S is a type such that $EXP(S) \simeq_\mathbf{R} S$ and for all $S' \in \mathbf{S}$ such that $S' \simeq_\mathbf{R} S$ we have $EXP(S) \sqsubseteq_\sigma S'$; i.e., $EXP(S)$ is the most refined type among the types $\simeq_\mathbf{R}$-equivalent to the type S as it includes all the possible restrictions implied by the schema $\mathbf{R}$. Thus, it is the lowest with respect to the $\sqsubseteq_\sigma$ relation among all the types $\simeq_\mathbf{R}$-equivalent to S. Note that, the recursive steps generating $EXP(S)$ individuate a class of $\simeq$-equivalent types, where each element is a type $\simeq_\mathbf{R}$-equivalent to the original type S. An

$$\left.\begin{array}{l}\mathbf{R}_\sigma \left\{\begin{array}{lcl}\sigma(\texttt{Manager}) & = & [\texttt{name: String, salary: } 40 \div \infty, \texttt{level: } 5 \div 15] \\ \sigma(\texttt{TManager}) & = & \texttt{Manager} \sqcap [\texttt{level: } 8 \div 12] \\ \sigma(\texttt{Material}) & = & [\texttt{name: String, risk: Int, feature: } \forall\{\texttt{String}\}] \\ \sigma(\texttt{SMaterial}) & = & \texttt{Material} \\ \sigma(\texttt{Storage}) & = & [\texttt{managed-by: Manager, stock: } \forall\{\texttt{Material}\}] \\ \sigma(\texttt{SStorage}) & = & \texttt{Storage}\end{array}\right. \\ \\ \begin{array}{ll}(R_1) & \texttt{Material} \sqcap [\texttt{risk: } 10 \div \infty] \to \texttt{SMaterial} \\ (R_2) & \texttt{Storage} \sqcap (\texttt{stock.}\exists\texttt{.feature.}\exists\texttt{: } \textit{"F1"}) \to [\texttt{managed-by: TManager}] \\ (R_3) & \texttt{Storage} \sqcap [\texttt{stock: } \exists\{\texttt{SMaterial}\}] \to \texttt{SStorage}\end{array}\end{array}\right\} \mathbf{R}$$

Table 1: The Company domain schema

element of this class of equivalence can be evaluated by means of the following function Γ from $\mathbf{S}$ to $\mathbf{S}$:

$$\Gamma(S) = \begin{cases} S \sqcap \sqcap_k (p_k : S_k^c) & \forall R_k, p_k : S \sqsubseteq_\sigma (p_k : S_k^a), \\ & S \not\sqsubseteq_\sigma (p_k : S_k^c) \\ S & \text{otherwise} \end{cases}$$

and we set $\tilde{\Gamma} = \Gamma^{\bar{i}}$, where $\bar{i}$ is the least integer such that $\Gamma^{\bar{i}} = \Gamma^{\bar{i}+1}$. [2]

THEOREM 1 *For all $S \in \mathbf{S}$, $EXP(S)$ can be computed by means of $\tilde{\Gamma}(S)$.*

COROLLARY 1 *For all $S, S' \in \mathbf{S}$ we have $S \sqsubseteq_\mathbf{R} S'$ if and only if $\tilde{\Gamma}(S) \sqsubseteq_\sigma S'$.*

Therefore, computing subsumption in a schema $\mathbf{R}$ can be performed by firstly determining the semantic expansion of a type and, secondly, by computing subsumption in the subschema $\mathbf{R}_\sigma$ [2].

Let us give two simple query optimization examples related to schema of table 1.

Q: "Select storages storing *at least* a material having a risk ≥ 15"

$$Q = \texttt{Storage} \sqcap [\texttt{stock: } \exists\{\texttt{Material} \sqcap [\texttt{risk: } 15 \div \infty]\}]$$

By rules R_1, R_3:

$$\tilde{\Gamma}(Q) = \texttt{SStorage} \sqcap [\texttt{stock: } \exists\{\texttt{SMaterial} \sqcap [\texttt{risk: } 15 \div \infty]\}]$$

In this way, we can optimize the query by obtaining the *most specialized generalization* of the classes involved in the query.

Another rewriting rule proposed in [17; 19] is the *constraint removal*, i.e., removal of implied factors. We formalize constraint removal by subsumption. As an example, consider the query:

Q: "Select storages managed by managers with a level $(6 \div 14)$ and storing at least a material with a feature F1":

$$Q = \underbrace{\texttt{Storage} \sqcap (\texttt{stock.}\exists\texttt{.feature.}\exists\texttt{: } \textit{"F1"})}_{S} \sqcap \underbrace{(\texttt{managed-by.level: } 6 \div 14)}_{S'}$$

By rule R_2 and by $\sigma(\texttt{TManager})$, we have $S \sqsubseteq_\mathbf{R} S'$, as $\tilde{\Gamma}(S) \sqsubseteq_\sigma S'$. Thus, S' can be eliminated from Q avoiding the access to the class `Manager`.

[2]The existence of $\bar{i}$ is guaranteed as the number of rules is finite and a rule cannot be applied more than once with the same path ($S \not\sqsubseteq_\sigma (p: S^c)$).

References

[1] D. Beneventano, S. Bergamaschi, S. Lodi, and C. Sartori. Using subsumption in semantic query optimization. In A. Napoli, editor, *IJCAI Workshop on Object-Based Representation Systems*, Chambery - France, August 1993.

[2] S. Bergamaschi and B. Nebel. Acquisition and validation of complex object database schemata supporting multiple inheritance. *Applied Intelligence: The International Journal of Artificial Intelligence, Neural Networks and Complex Problem Solving Technologies*, 1993. to appear.

[3] S. Bergamaschi and C. Sartori. On taxonomic reasoning in conceptual design. *ACM Transactions on Database Systems*, 17(3):385–422, September 1992.

[4] R.J. Brachman and J.G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2):171–216, 1985.

[5] M. Buchheit, F.M. Donini, and A. Schaerf. Decidable reasoning in terminological knowledge representation systems. *13th. Int. Joint Conf. on Artificial Intelligence*, 1:704–709, 1993.

[6] M. Buchheit, M. A. Jeusfeld, W. Nutt, and M. Staudt. Subsumption between queries to object-oriented database. In *EDBT*, pages 348–353, 1994.

[7] L. Cardelli. A semantics of multiple inheritance. In *Semantics of Data Types - Lecture Notes in Computer Science N. 173*, pages 51–67. Springer-Verlag, 1984.

[8] U. S. Chakravarthy, J. Grant, and J. Minker. Logic-based approach to semantic query optimization. *ACM Transactions on Database Systems*, 15(2):162–207, June 1990.

[9] Edward P.F. Chan. Containment and minimization of positive conjunctive queries in oodb's. In *Principles of Database Systems*, pages 202–11. ACM, 1992.

[10] M.M. Hammer and S. B. Zdonik. Knowledge based query processing. In *6th Int. Conf. on Very Large Databases*, pages 137–147, 1980.

[11] M. Jeusfeld and M. Staudt. Query optimization in deductive object bases. In Freytag, Maier, and Vossen, editors, *Query Processing for Advanced Database System*. Morgan Kaufmann Publishers, Inc., June 1993.

[12] W. Kim. A model of queries for object-oriented database systems. In *Int. Conf. on Very Large Databases*, Amsterdam, Holland, August 1989.

[13] J. J. King. *Query optimization by semantic reasoning*. PhD thesis, Dept. of Computer Science, Stanford University, Palo Alto, 1981.

[14] J. J. King. Quist: a system for semantic query optimization in relational databases. In *7th Int. Conf. on Very Large Databases*, pages 510–517, 1981.

[15] C. Lecluse and P. Richard. Modelling complex structures in object-oriented databases. In *Symp. on Principles of Database Systems*, pages 362–369, Philadelphia, PA, 1989.

[16] H. H. Pang, H. Lu, and B.C. Ooi. An efficient semantic query optimization algorithm. In *Int. Conf. on Data Engineering*, pages 326–335, 1991.

[17] S. Shenoy and M. Ozsoyoglu. A system for semantic query optimization. *ACM-SIGMOD*, pages 181–195, May 1987.

[18] S. Shenoy and M. Ozsoyoglu. Design and implementation of a semantic query optimizer. *IEEE Trans. Knowl. and Data Engineering*, 1(3):344–361, September 1989.

[19] M. Siegel, E. Sciore, and S. Salveter. A method for automatic rule derivation to support semantic query optimization. *ACM Transactions on Database Systems*, 17(4):563–600, December 1992.

Constraints in Complex Object Database Models

D. Beneventano, S. Bergamaschi°, S. Lodi, C. Sartori

Dipartimento di Elettronica, Informatica e Sistemistica
Università di Bologna - CIOC-CNR
°Facoltà di Ingegneria, Università di Modena - CIOC-CNR

1 Introduction

Database design almost invariably includes a specification of a set of rules, the *integrity constraints*, which should guarantee its consistency. Constraints are expressed in various fashions, depending on the data model: e.g. subsets of first order logic, or inclusion dependencies and predicates on row values, or methods in OO environments. Provided that an adequate formalism to express them is available, the following question arises: Is there any way to populate a database which satisfies the constraints supplied by a designer? Means of answering to this question should be embedded in automatic design tools, whose use is recommendable or often required in the difficult task of designing complex database schemas. Many recently proposed tools are based on reasoning activities derived from research in Description Logics, especially because of the advantages of taxonomic reasoning for the preservation of coherence and minimality of a schema [5][2]. The contribution of this research is to propose a computational solution to the problem of schema consistency in Complex Object Data Models (CODM). We extend the expressiveness of CODMs [4] based on class and type taxonomies to capture the semantics of a relevant set of state constraints and present some examples of its application; we then summarize the properties of a tableaux-based solution to the problem of schema consistency (therefore including state constraint consistency) in such extended model. Such a solution is actually a modification of existing algorithms for Description Logics [11][8][9][7].

In order to substantially enhance OODBMSs and CODMs with reasoning features, the next step should be the design of a front end to the DB to validate insertions and updates, with respect to the extended scheme description.

2 Values, Objects, Paths, Types and Schemas

Let $\mathcal{D}$ denote the set of base-values (denoted by $d, d', \ldots$) equal to the union of the sets of integers, strings, boolean, and reals; let $\mathbf{A}$ be a countable set of *attributes* (denoted by $a_1, a_2, \ldots$), and $\mathcal{O}$ a countable set of *object identifiers* (denoted by $o, o', \ldots$) disjoint from $\mathcal{D}$. We define the set $\mathcal{V}(\mathcal{O})$ of all *values over* $\mathcal{O}$ as inductively built by the following rule (assuming $p \geq 0$ and $a_1, \ldots, a_p$ distinct attributes in $\mathbf{A}$):

$$v \rightarrow d \mid o \mid \{v_1, \ldots, v_p\} \mid [a_1 : v_1, \ldots, a_p : v_p]$$

The equality, inequality and order relations $\theta \in \{=, \neq, >, <, \geq, \leq\}$, are defined as usual on $\mathcal{D}$; equality and inequality are extended from $\mathcal{D}$ to all $\mathcal{V}(\mathcal{O})$ in the obvious way. Object identifiers are assigned values by a *total value function* δ from $\mathcal{O}$ to $\mathcal{V}(\mathcal{O})$.

A *path* p is a sequence $e_1.e_2.\ldots.e_n$, where $e_i \in \mathbf{A} \cup \{\Delta, \{\}, \epsilon\}$, $i = 1 \ldots n$, $n \geq 1$. By ϵ we denote the *empty path*, and by $\mathbf{W}$ the set of all paths. Given a set of object identifiers $\mathcal{O}$ and a value function δ, we introduce the function $\mathcal{J} : \mathbf{W} \longrightarrow 2^{\mathcal{V} \times \mathcal{V}}$ defined as follows:

$$\begin{aligned}
\mathcal{J}[\epsilon] &= \Big\{(v, v) \in \mathcal{V} \times \mathcal{V}\Big\} \\
\mathcal{J}[a] &= \Big\{(v_1, v_2) \in \mathcal{V} \times \mathcal{V} \mid v_1 = [\ldots, a : v_2, \ldots]\Big\} \\
\mathcal{J}[\Delta] &= \Big\{(o, v) \in \mathcal{O} \times \mathcal{V} \mid \delta(o) = v\Big\} \\
\mathcal{J}[\{\}] &= \Big\{(v', v'') \in \mathcal{V} \times \mathcal{V} \mid v'' \in v'\Big\} \\
\mathcal{J}[p] &= \mathcal{J}[e_n] \circ \ldots \circ \mathcal{J}[e_2] \circ \mathcal{J}[e_1]
\end{aligned}$$

We assume a countable set $\mathbf{N}$ of type names (denoted by N, N'), including the set $\mathbf{B} = \{\mathtt{Int}, \mathtt{String}, \mathtt{Bool}, \mathtt{Real}\}$ of base-type designators (which will be denoted by B) and the symbols $\top$, $\bot$. $\mathbf{S}(\mathbf{A}, \mathbf{N})$ denotes the set of all *finite type descriptions* (S, S', ...), over given $\mathbf{A}, \mathbf{N}$, obtained according to the following abstract syntax rule:

$$\begin{aligned}
S \rightarrow\ & N \mid S \sqcup S'' \mid S \sqcap S' \mid \neg S \\
& \mid \{S\}_{(min,max)} \mid [a_1 : S_1, \ldots, a_k : S_k] \mid \Delta S \\
& \mid p\theta p' \mid p\theta d \mid p\uparrow
\end{aligned}$$

A *schema* σ over $\mathbf{S}(\mathbf{A}, \mathbf{N})$ is a function $\sigma : \mathbf{N} \setminus (\mathbf{B} \cup \{\top, \bot\}) \rightarrow \mathbf{S}$, with $\sigma = \sigma_P \cup \sigma_D$ and $\sigma_P \cap \sigma_D = \emptyset$.

Let $\mathcal{I}_{\mathbf{B}}$ be the (fixed) standard interpretation function from $\mathbf{B}$ to $2^{\mathcal{D}}$. For a given object assignment δ, each type expression S is mapped to a set of values as its interpretation. The *interpretation function* $\mathcal{I} : \mathbf{S} \rightarrow 2^{\mathcal{V}(\mathcal{O})}$

is recursively defined as follows (we omit the definitions for $\sqcup$, $\sqcap$, $\neg$, $\top$, $\bot$):

$\mathcal{I}[B] = \mathcal{I}_{\mathbf{B}}[B]$
$\mathcal{I}[\{S\}_{(n,m)}] = \{M \subset \mathcal{V}(\mathcal{O}) \mid M \subseteq \mathcal{I}[S], n \leq |M| \leq m\}$
$\mathcal{I}[[a_1 : S_1, \ldots, a_p : S_p]] =$
$\quad \{\mu : \mathbf{A} \to \mathcal{V}(\mathcal{O}) \mid a_i \in \operatorname{dom}(\mu), \mu(a_i) \in \mathcal{I}[S_i]\}$
$\mathcal{I}[\neg S] = \mathcal{V}(\mathcal{O}) \setminus \mathcal{I}[S]$
$\mathcal{I}[\triangle S] = \{o \in \mathcal{O} \mid \delta(o) \in \mathcal{I}[S]\}$
$\mathcal{I}[(p\theta d)] = \{v \in \mathcal{V}(\mathcal{O}) \mid \exists d' \in \mathcal{D}: (v, d') \in \mathcal{J}[p] \wedge (d'\theta d)\}$
$\mathcal{I}[(p_1\theta p_2)] = \{v \in \mathcal{V}(\mathcal{O}) \mid \exists d_1, d_2 \in \mathcal{D}: (v, d_1) \in \mathcal{J}[p_1]$
$\qquad \wedge (v, d_2) \in \mathcal{J}[p_2] \wedge (d_1\theta d_2)\}$
$\mathcal{I}[(p\uparrow)] = \mathcal{V}(\mathcal{O}) \setminus \{v \in \mathcal{V}(\mathcal{O}) \mid \exists v' \in \mathcal{V}(\mathcal{O}): (v, v') \in \mathcal{J}[p]\}$

An interpretation function $\mathcal{I}$ as defined above is a *legal instance* of a schema σ iff the set $\mathcal{O}$ is *finite*, and for all $N \in \mathbf{N}$, $N \in \operatorname{dom}(\sigma_P)$ implies $\mathcal{I}[N] \subseteq \mathcal{I}[\sigma_P(N)]$, and $N \in \operatorname{dom}(\sigma_D)$ implies $\mathcal{I}[N] = \mathcal{I}[\sigma_D(N)]$. Therefore, the interpretation of a primitive type has to be provided by the user, according to the given description, while the interpretation of a derived type is drawn from its definition and from the interpretation of primitive types, thus corresponding to a view in database context. Since the schema is acyclic, the legal instance is uniquely determined by the interpretations of primitive types. The subsumption relation $\leq$ between types and the coherence of a type are defined as usual respectively by $S \leq S' \overset{\text{def}}{\Longleftrightarrow} \forall$ legal instance $\mathcal{I}$ of σ: $\mathcal{I}[S] \subseteq \mathcal{I}[S']$, and $\exists \mathcal{I}: \mathcal{I}[S] \neq \emptyset$. A schema is coherent if and only if all the names of its domain are coherent.

3 Examples and Comments

In order to explain the purpose of our constraint validation method, let us consider the organizational structure of a company. Assume the following: Employees have a name and earn a salary. Managers are employees and have a level composed of a qualification and a parameter. Repositories have a denomination, wich can be either a string or a structure composed by a repository name and an address; a repository stocks a set of at least one and at most five materials. Materials are described by a name and a risk. Departments have a denomination (string), and are managed by a manager. Warehouses have all the properties of departments and repositories.

The above description is expressed in our formalism as follows:

$$\begin{aligned}
\sigma(\texttt{Level}) &= [\texttt{qualification: String},\\
&\quad\ \texttt{parameter: Int}]\\
\sigma(\texttt{Employee}) &= \triangle[\texttt{name: String, salary: Int}]\\
\sigma(\texttt{Manager}) &= \texttt{Employee} \sqcap \triangle[\texttt{level: Level}]\\
\sigma(\texttt{Repository}) &= \triangle[\texttt{denomination: String} \sqcup\\
&\qquad [\texttt{rname: String},\\
&\qquad\ \texttt{address: String}],\\
&\qquad\ \texttt{stock: }\{\texttt{Material}\}_{(1,5)}]\\
\sigma(\texttt{Department}) &= \triangle[\texttt{denomination: String},\\
&\quad\ \texttt{managed-by: Manager}]\\
\sigma(\texttt{Warehouse}) &= \texttt{Department} \sqcap \texttt{Repository}\\
\sigma(\texttt{Material}) &= \triangle[\texttt{name: String, risk: Int}]
\end{aligned}$$

Class and type descriptions use the tuple ([]) and set ({}) constructors, the latter with a cardinality interval. The $\triangle$ operator enforces a distinction between object classes, preceded by it, and value types. With respect to the formalism in [4], the general complement ($\neg$) and the union operator ($\sqcup$), present in many works on complex object data models [1] [10], have been added.

Another relevant capability of this formalism is path support including class and set symbols; therefore navigation through schema classes and types is supported across those types as well, as in

$$\begin{aligned}
&\sigma(\texttt{Technician}) =\\
&\quad \texttt{Employee} \sqcap \triangle[\texttt{works-in: Department}]\\
&\qquad \sqcap (\triangle\texttt{salary} < \triangle\texttt{works-in}.\\
&\qquad \triangle\,\texttt{managed-by}.\ \triangle\ \texttt{salary}).
\end{aligned}$$

where a technician is forced to have a salary lower than the salary of the manager of its department. Paths of this form are necessary for query languages in a OODBMS environment and have been recently introduced also to express constraints [6].

The general form of integrity constraint we represent is an *if then* rule universally quantified over the elements of a class C: "**for all** $x \in C$ **if** x is of type S **then** x must be of type S'" where S and S' are type expressions that can contain *range constraints* and *path equations*. Since the above rule represents an implication, it can be translated into a class description using $\sqcup$, $\neg$. For example, if the class `Shipment` is introduced:

$$\sigma(\texttt{Shipment}) = \triangle[\texttt{urgency: Int, item: Material}]$$

and the following integrity constraint is specified on it: for all shipments it must hold that if the risk of the material is greater than 3 then its urgency must be greater than 10, that is: "**for all** $x \in$ `Shipment` **if** x is of type `Shipment` $\sqcap$ ($\triangle$`item`. $\triangle$ `risk` > 3) **then** x is of type `Shipment` $\sqcap$ ($\triangle$`urgency` > 10)", then the constraint can be embedded in the class description, obtaining the following type description for `Shipment`:

$$\begin{aligned}
\sigma(\texttt{Shipment}) &= \triangle[\texttt{urgency: Int, item: Material}]\\
&\quad \sqcap(\neg(\triangle\texttt{item}.\ \triangle\,\texttt{risk} > 3)) \sqcup\\
&\quad (\triangle\texttt{urgency} > 10))
\end{aligned}$$

4 Checking Coherence

The coherence checking completion rules, devised by DL researchers, are a suitable starting point also to solve the corresponding problem in CODMs. Boolean rules remain obviously unchanged, while rules that operate with feature constructors of DLs have to be extended to deal with $\triangle$ and {} constructor. Essentially, a class $\triangle S$ expresses a function; therefore there has to be at most one variable ound to the type S, representing a value pointed to by n object in the interpretation of $\triangle S$. This happens to e true also for set types: in similarity with the case of umber restrictions in DLs, a set $\{S\}_{(n,m)}$ needs not proluce more than one variable bound to the type S, even if hey have cardinality bounds. Paths are decomposed as eatures by creating one variable for every intermediate ttribute, class or set symbol in the path. Consistency s provably decidable [3], but its complexity is, of course, aised by the presence of general complements.

References

[1] S. Abiteboul and P. Kanellakis. Object identity as a query language primitive. In *SIGMOD*, pages 159–173. ACM Press, 1989.

[2] D. Beneventano, S. Bergamaschi, and C. Sartori. Taxonomic reasoning with cycles in LOGIDATA$^+$. In P. Atzeni, editor, *LOGIDATA$^+$: Deductive Databases with Complex Objects*, page 105:128. Springer-Verlag: LNCS n. 701, Heidelberg - Germany, 1993.

[3] Domenico Beneventano, Sonia Bergamaschi, Stefano Lodi, and Claudio Sartori. Reasoning with constraints in complex object data models. Technical report, CIOC-CNR, Bologna, Italy, March 1994. In preparation.

[4] S. Bergamaschi and B. Nebel. Theoretical foundations of complex object data models. Technical Report 5/91, Roma, January 1992.

[5] S. Bergamaschi and C. Sartori. On taxonomic reasoning in conceptual design. *ACM Transactions on Database Systems*, 17(3):385–422, September 1992.

[6] N. Coburn and G.E. Weddel. Path constraints for graph-based data models: Towards a unified theory of typing constraints, equations and functional dependencies. In *2nd Int. Conf. on Deductive and Object-Oriented Databases*, pages 312–331, Heidelberg, Germany, December 1991. Springer-Verlag.

[7] F.M. Donini, M. Lenzerini, D. Nardi, and W. Nutt. The complexity of concept languages. In J. Allen, R. Fikes, and E. Sandewall, editors, *KR '91 - 2nd Int. Conf on Principles of Knowledge Representation and Reasoning*, pages 151–162, Cambridge - MA, April 1991. Morgan Kaufmann Publishers, Inc.

[8] Bernhard Hollunder and Werner Nutt. Subsumption algorithms for concept languages. DFKI Research Report RR-90-4, Deutsche Forschungszentrum für Künstliche Intelligenz GmbH, Kaiserslautern, Germany, April 1990.

[9] Bernhard Hollunder, Werner Nutt, and M. Schmidt-Schauss. Subsumption algorithms for concept languages. In *Proc. ECAI-90*, Stockholm, Sweden, 1990.

[10] C. Lecluse and P. Richard. The O2 database programming language. In *15th Int. Conf. on Very Large Databases*, pages 411–422, Amsterdam, February 1989.

[11] M. Schmidt-Schauss and G. Smolka. Attributive concept descriptions with unions and complements. *Artificial Intelligence*, 48(1), 1991.

Advantages of using a Terminological system for integrating Relational Databases

Blanco J.M. Illarramendi A. Goñi A. Bermúdez J.
Facultad de Informática, Universidad del País Vasco. Apdo. 649, 20.080 San Sebastián. SPAIN
e-mail: jipileca@si.ehu.es
phone: + 34 43 218000

Abstract

The integration of heterogeneous and autonomous information sources is a requirement for the new type of cooperative information systems. In this paper we show the advantages of using a terminological system for integrating pre-existing relational databases. From the resulting integrated schema point of view, using a terminological system allows for the definition of semantically richer integrated schema. From the integrated schema generation process point of view, the use of a terminological system permits the definition of a more consistent, broad and automatic process. Last, from the processing of queries formulated over the integrated schema point of view, terminological systems provide interesting features for incorporating semantic and caching query optimization techniques.

1 Introduction

Within most organizations there exist many heterogeneous databases, that have been defined independently, which store data that are somehow related. A global and uniform treatment of all these data, maintaining at the same time the autonomy of the component systems, will bring the organizations the opportunity of improving the management of their information.

In the literature several attempts to solve the interoperability problem and in particular the database interoperability can be found. However, a standard solution that gives an answer to all requirements has not been proposed yet. In [SPD92] an overview of different proposals is presented.

In this paper we wish to present our experience building a system that allows the integration of heterogeneous and autonomous relational databases using a terminological system [BIG91], [BIPG92], [BIGP94], [MIB94], [BIG94]. Indeed the main focus of the paper is on presenting the advantages of using a terminological system for the main tasks involved in the integration process: a *translation* task for obtaining a uniform and richer representation of the relational schemata that must be integrated; an *integration* task for creating an integrated schema and a *query processing* task for giving answers to the queries formulated over the integrated schema.

The integration of database systems with terminological systems has already been proposed in other work among which we can point out [BS92], [ACHK93], [BST+93] and [SGN93]. In [BS92] the use of taxonomic reasoning techniques to support the conceptual design of schemata is proposed. They maintain that the designed schemata will be more consistent. In [ACHK93] an existing terminological system (LOOM) is used as a model to describe database schemata and, a system that uses LOOM to provide efficient access to a relational database is described. However, the schemata integration task is somehow limited; they do not allow relations between objects that belong to different schemata. Our proposal is richer in this sense. In [BST+93] it is argued that the exploitation of pre-existing databases using a terminological system (CLASSIC) allows the generation of new information sources. Nevertheless, they assume that the integrated schema is defined previously and so the integration task is reduced to the mapping process among the local and the integrated schemata. Last, in [SGN93] CANDIDE, a DBMS based on description logics, is used as a tool to automate a significant part of the schemata integration process. This is the work that has more points in common with our proposal, however, the differences can be summarized in the following aspects: 1) [SGN93] assume that all the schemata to be integrated are defined using CANDIDE, so a previous translation step is not defined. In our case we deal with heterogeneous relational databases; 2) [SGN93] center their solution around the specification of correspondences among attributes, in our case correspondences among concepts play the main role; and 3) our integration philosophy is more interactive.

The main advantages that the use of a terminological system provides for the integration process and that are explained in the following sections can be summarized in the following points: 1) the possibility of defining semantically rich integrated schema; 2) a wider range of translation and integration types; 3) the automatic verification and derivation of new relationships among classes; 4) automatic detection of inconsistent queries and also reformulation of them; and last, 5) the definition of cached concepts and automatic discovering of cached queries. In the remainder of this paper we present the mentioned advantages.

2 Translation

The first requirement to do an integration of heterogeneous databases is to define them in a uniform way, that is, using the same data model, called a canonical model. It is assumed that this canonical model should have an expressive power higher than that offered by the relational model in order to support an interesting integration process; hence most of the works in the area use semantic or object-oriented data models. In our case we use a terminological system. The main advantages that its use provides in the translation task can be summarized in the following points: 1) definition of semantically richer schemata and 2) definition of complex translation types.

2.1 Definition of semantically richer schemata

This can be achieved using on the one hand, rich structural mechanisms provided by a terminological system such as generalization/specialization that permit the definition of conceptual hierarchies. Moreover, terminological systems apart from the specialization of concepts permit the specialization of roles. Therefore, it is possible to define hierarchies among roles too. For example, the relation *client* defined in a relational schema as

client(c#, name, address, pay-interval)

that contains data about all the clients of an enterprise could be translated into three different concepts namely, *client*, *slowpayer* and *good* related those last two with the first one by an *is-a* relationship.

In order to be able to obtain these hierarchies, properties about dependencies: inclusion, exclusion and functional; information about null values or information about domain values of attributes can be used. For example, in the relation *client* the domain values that *clients* qualified as *slowpayer* take for the attribute *pay-interval* must be greater than 30, and for the *good clients* less than or equal to 30.

Furthermore, the use of a terminological system facilitates the process of building these hierarchies because, due to the classification provided by them, the *is-a* relationships are handled automatically. This means that the PRI(Person Responsible of the Integration) does not need to know the exact place in the hierarchy where a new concept should be introduced. The terminological system will automatically discover the correct place.

Last, the feature provided by terminological systems of attaching descriptions to concepts in terms of necessary or necessary and sufficient properties enrich also from a semantic point of view the resulting schema.

2.2 Definition of complex translation types

This advantage is also related to the fact that concepts are associated with intensional descriptions. Intensional descriptions allows one to define translations in terms of semantic properties that are not supported by other approaches. For example, the concept *active-employee* of a hierarchy could be defined as

active-employee := employee and atleast(2,participates)

where *employee* makes reference to a relational table and *atleast(2,participates)* expresses a property that must be verified by the tuples in order to be considered them as *active-employees*. Complex terms can be defined over concepts and roles in the hierarchy. Notice that this issue allows a wide range of translation types.

3 Integration

Once the terminologies have been obtained from the component databases, the next step consists in integrating the different terminologies in order to obtain an integrated schema.

In general, the integration task requires first of all, to define correspondences among data elements (concepts and roles) of the terminologies that must be integrated and then the application of some integration rules. In the literature several approaches that deal with the system integration task have been presented [BEM92]. In the following we summarize the advantages that the use of a terminological system provides for this task: 1) the possibility of automatically discovering errors in the definition of correspondences between data elements expressed by the PRI; 2) the automatic inference of new correspondences between data elements not explicitly defined by the PRI; and 3) the possibility of defining correspondences between data elements using intensional descriptions.

3.1 Verification of correspondences

In the case of dealing with a terminological system, correspondences between data elements of the terminologies that must be integrated can be verified using the intensional descriptions associated with the concepts among which the correspondences are established. For example, if the following two concepts are described

normal := client and atleast(1,pay-interval) and all(pay-interval, le(30))

government := client and atleast(1, pay-interval) and all(payment, ge(60))

and the PRI expresses that *normal equivalent government* because he or she misunderstood the semantics associated with the concepts, the terminological system will inform him or her that it is not correct because the descriptions are incompatible.

3.2 Automatic derivation

The classification mechanism can play an important role in the integration task deriving new relationships between data elements not defined explicitly by the PRI. For example, during the integration task of the following two terminologies *t1* and *t2*, if the PRI asserts that *t1_client equivalent t2_client*[1] (figure 1) as a result of applying the corresponding integration rule (explained in

[1]This correspondence is correct in terms of the descriptions associated with the concepts *t1_client* and *t2_client*

detail in [BIG94]) only one concept *client* will appear in the resulting integrated schema.

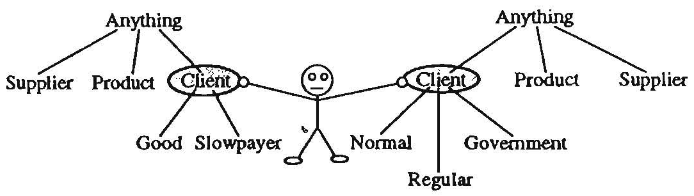

Figure 1: Asserting an equivalence

Moreover, after a redefinition process, the terminological system will automatically discover that *good* and *normal* are equivalent concepts, and that *slowpayer* subsumes *regular* and *Government* (see figure 2).

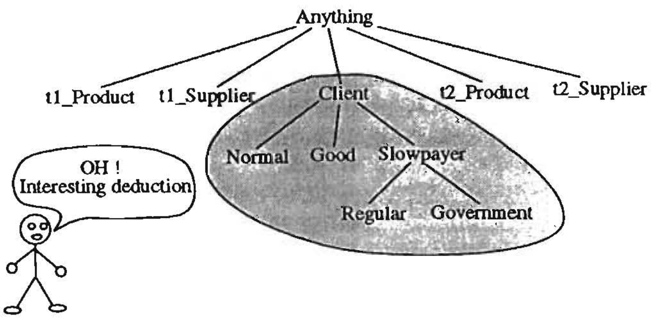

Figure 2: Resulting integrated schema

3.3 Correspondences based on descriptions

In the majority of work in the integration area the correspondences are defined using the notion of RWS, that is based on the extensions of the data elements that are being related. However, using a terminological system, more complex correspondences can be expressed using semantic properties. This is possible due to the facility provided of adding intensional descriptions to the concepts definitions. For example, the PRI could define the correspondence *engineer* ≡ *employee and all(salary,ge(30000))* to express that *engineers* of one terminology are employees (according to their definition in other terminology) with a salary greater than 30000. It is not possible to define this type of correspondences by using directly the Real World State (RWS) notion utilized in other work. Thus the specification of concepts using intensional descriptions permits a richer integration process.

4 Query Processing

Once an integrated schema has been obtained, it provides the users with an integrated and global view of the data stored in pre-existing databases that will be used by them to formulate queries over. To find the answers to these queries in an efficient way is the goal of the query processing step.

In general, the query processing is carried out with two different kinds of processors: the Global Query processor and the Local Query processor. The subgoals of the former one are to make a global query optimization; to decompose a query into subqueries that will run over different databases and to generate an optimal plan to build the answer. The subgoals of the last one are to make local optimizations; to find the answers for the subqueries; and last, to send the answers to the Global Query process when is needed.

The main advantages that the use of a terminological system provides for the Global Query processor (notice that Local Query processors are developed for the local database systems) are summarized in the following points:

- The possibility of giving intensional answers.
- A semantic optimization of queries using the classification mechanism. In the database area, semantic query optimization methods exploit domains knowledge such as that expressed by integrity constraints, hierarchies, etc. to detect inconsistent queries or to transform a user formulated query into another one with the same answer, that is semantically equivalent, but that can be processed more efficiently. These semantic optimization methods are external to the database systems and are defined as a special purpose mechanism. Using a terminological system, it is possible to do semantic query optimization using the reasoning capabilities of these systems. The classification mechanism of a terminological system allows the detection of inconsistent queries and the reformulation of them.
- Support for defining and identifying cached data. After a process of identifying the data that are worth caching, they can be grouped under a concept and later introduced into a terminology by using the capability of defining rules provided by terminological systems[2]. For example, suppose that we are interested in caching *active-employee* described as

  active-employee := employee and atleast(2,participates)

  it could be declared as cached defining the rule

  active-employee => cached

  It is obvious that this would not be useful if one could not ask if a concept is cached or not. But the subsumption mechanism of the terminological system provides this capability. In fact, if the concept *cached* subsumes the one about which we are asking, then that concept is cached. For example, after the declaration of the previous rule, if the following query

  getall employee and atleast(3,participates)

  is formulated, then the system will recognize it as cached, because

[2]With the use of the rules, non-definitional information can be added to the terminologies.

subsumes(cached,employee and atleast(3,participates)).

Last, notice that the user does not need to know the existence of the generated cached concepts to formulate his or her queries because the Global Query processor will detect the relationship between these concepts and the user queries.

5 Conclusions

In the area of information systems the problem of integrating different application environments has attracted substantial attention. Of special interest is the integration of database systems with knowledge based systems in order to take advantage of the additional features provided by latter, without giving up the services provided by the former. The practical benefits of this type of integration apply to many different contexts, such as access to pre-existing databases when working with knowledge bases, allowing databases to provide persistent object support for knowledge-based applications, and the coexistence of different autonomous databases under a global integrated schema by using a knowledge based system. In this paper we have concentrated in this last point. We have shown the advantages of using a terminological system for the integration of pre-existing relational databases.

In summary we can say that from the integration point of view, terminological systems allow for consistent integration process; and from the query processing point of view they provide mechanisms to support query optimization techniques such as semantic query optimization and or caching optimization techniques.

Acknowledgments We would like to thank Alex Borgida and the members of the AIMS proyect group fot the many useful discussions and suggestions.

References

[ACHK93] Y. Arens, C.Y. Chee, C. Hsu, and C.A. Knoblock. Retrieving and integrating data from multiple information sources. *International Journal on Intelligent and Cooperative Information Systems*, 2(2):127–158, 1993.

[BEM92] O.A. Bukhres, A.K. Elmagarmid, and J.G. Mullen. Object-oriented multidatabases: Systems and research overview. In *Proc. of the Int. Conf. Information and Knowledge Management CIKM-92 Baltimore USA*, 1992.

[BIG91] J.M. Blanco, A. Illarramendi, and A. Goñi. A uniform approach to design a federated information base using BACK. In *Proc. of the Terminological Logic Users Workshop. Berlin*, 1991.

[BIG94] J.M. Blanco, A. Illarramendi, and A. Goñi. Building a federated database system: an approach using a knowledge based system. Submitted to the Int. Journal on Intelligent and Cooperative Information Systems. In revision., 1994.

[BIGP94] J.M. Blanco, A. Illarramendi, A. Goñi, and J. M. Pérez. Using a terminological system to integrate relational databases. In *Information Systems Design and Hypermedia. CEPADUES-EDITIONS*, 1994.

[BIPG92] J. M. Blanco, A. Illarramendi, J. M. Pérez, and A. Goñi. Making a federated database system active. In *Database and Expert Systems Applications*. Springer-Verlag, 1992.

[BS92] S. Bergamaschi and C. Sartori. On Taxonomic Reasoning in Conceptual Design. *ACM Transactions on Database Systems*, 17(3):385–421, 1992.

[BST+93] R.J. Brachman, P.G.. Selfridge, L.G. Terveen, B. Altman, A. Borgida, F. Halpner, T. Kirk, A. Lazar, D.L. McGuinnes, and L.A. Resnick. Integrated support for data archaeology. *International Journal on Intelligent and Cooperative Information Systems*, 2(2):159–185, 1993.

[MIB94] E. Mena, A. Illarramendi, and J. M. Blanco. Magic: An interface for generating mapping information between object-based and relational systems. In *Information Systems Design and Hypermedia*. Cepadues-Editions, 1994.

[SGN93] A.P. Sheth, S.K. Gala, and S.B. Navathe. On automatic reasoning for schema integration. *International Journal of Intelligent and Cooperative Information Systems*, 2(1):23–50, 1993.

[SPD92] S. Spaccapietra, C. Parent, and Y. Dupont. Model independent assertions for integration of heterogeneous schemas. *VLDB*, 1:81–126, 1992.

Persistent Maintenance of Object Descriptions using BACK

Albrecht Schmiedel
Technische Universität Berlin
atms@cs.tu-berlin.de

1 Background

Traditionally, systems implementing description logics (DL) such as CLASSIC, LOOM and BACK are in-core systems[1], and not well-adapted for storing and processing large amounts of data. They read definitions and descriptions from ASCII files, build up internal data structures, and do classification and consistency checking. As a result, they are fairly slow; if the KB has to be reprocessed each time the system is brought up, the number of manageable and accessable objects is rather limited. There are two limitations to be dealt with: (1) *space*: the number of instances that one can deal with is restricted to what can be handled in main memory, and (2) *time*: the number of manageable instances is also limited by the time it takes to process them, particularly if the results of classification and recognition are not saved between invocations of the system.

Both limitations can be overcome by providing some kind of persistency. Firstly, for handling a large number of instances, there must be transparent access to instances residing on secondary storage, and secondly the results of classification and recognition (the actual services of the terminological reasoner) must be made persistent as well in order to avoid redoing them each time an instance is accessed.

One solution for both aspects would probably be to implement BACK on top of a persistent storage manager (such as Exodus); it would transparently map data structures created by BACK to secondary storage, translate back and forth between pointers and object ids ('pointer swizzling'), and page in and out relevant portions of memory as needed. To my knowledge, this approach has not yet been taken with a terminological reasoner. However, a terminological system with a customized persistency mechanism (though not for a very expressive language) has been described in [Mays et al.,1991].

The other approach to scaling up the size of potential KBs involves some kind of coupling to external databases. This direction has been pursued in the AIMS project ([Bagnasco et al.,1991], [Illarramendi et al.,1991]), and recently also by [Borgida and Brachman,1993],[Devanbu,1993]. The primary motivation here is to access large, pre-existing databases transparently via a rich domain model expressed in the description logic. In practice, this is achieved by translating concepts into database queries, and thereby loading only those instances that a user is specifically interested in. This certainly lifts the space limitation mentioned above, and it also alleviates the limits imposed by processing times in those cases where the classification inference can be "delegated" to the database machinery by issuing an appropriate set of queries (cf. [Borgida and Brachman,1993]).

These two approaches are by no means mutually exclusive. In fact, a genuinely persistent DL system acting as a server to applications and as a client to a variety of databases and other data sources is an appealing scenario. The conceptual modelling capabilities make a DL system an ideal candidate as a front-end for heterogeneous data sources (cf. [Illarramendi et al.,1991]).

2 An Experiment with Semantic Indexing

The work we report on here involves neither a fully persistent DL system nor transparent access to databases; rather, it is an experiment aimed at alleviating the time and space limitations mentioned above by extending a pure in-core DL system with facilities for building and maintaining a persistent index into a (potentially large) set of externally stored individual descriptions. We believe that this technique, which we call *semantic indexing*, would also be useful as part of an intrinsically persistent system that accesses external data.

Our semantic index is constructed by maintaining a relation between a *set of indexing concepts* and a *set of indexed individuals*. Indexing concepts are a subset of all concepts defined in a terminology, chosen by the user. Indexed individuals are the ones to be included in the index (there may be others as well). We implemented this mapping using two persistent hash tables: For each indexing concept the set of immediate individuals as well as its cardinality can be looked up, and inversely for each indexed individual the set of most specific indexing concepts.

The index manager is a program built on top of the

[1]An exception to this is K-REP [Mays et al.,1991].

BACK++ system[2]. It provides the functionality for modifying the persistent index (adding/removing indexing concepts and indexed instances), and for supporting queries using the index.

The index is built incrementally. At no time do all the indexed individuals have to be loaded simultaneously. New individuals are added by loading their complete descriptions. The system determines the set of indexing concepts for each new individual, and updates the cardinality information associated with indexing concepts. After the index has been updated, the individual description can be unloaded. Adding new indexing concepts may involve reloading descriptions of those individuals which are 'candidates', i.e. not known to instantiate disjoint indexing concepts nor known to instantiate subsumees of the new one. The index manager maintains the necessary book-keeping information concerning the status of indexing concepts and individuals (loaded, modified, new, etc.).

The index can support queries w.r.t. a much larger set of individuals than can possibly be loaded into the system. Queries are processed in three phases. Initially, only the indexing concepts along with the cardinality information are loaded. In Phase I the query is classified and the user receives immediate feedback concerning the cardinality of the result set in terms of upper and lower bounds. The bounds are computed using subsumption and disjointness relations of the query w.r.t. indexing concepts for which exact cardinalities are known. Of course, the quality of the cardinality feedback depends very much on how close the query is related to existing indexing concepts.

In Phase II the actual extension of indexing concepts cached in the index is additionally used for computing upper and lower bounds on the the cardinality of the query. This generally results in much better cardinality estimates, at the cost of having to access the index on disc. In case the query is equivalent to a combination of indexing concepts, the exact cardinality can be computed. If this is not the case, and the exact extension of a query is desired, in Phase III the descriptions of the 'candidate individuals' (see above) have to be accessed and tested against the query. The user can now choose to declare the query as a new indexing concept for future immediate access.

We tested this approach by classifying the data of about 8000 medical examinations (and patients) against about 400 indexing concepts. Each examination had about 50 observations (represented as non-indexed individuals), so that a total of about 400,000 individuals were loaded and classified to build the persistent index. BACK++ took several hours for this. When using the index, cardinality constraints for queries can be obtained without delay, and of course the retrieval of sets of individuals equivalent to a combination of indexing concepts is nearly immediate. However, if candidates need to be processed, response times are less than optimal since the full descriptions are loaded and classified rather than only those parts relevant for the query at hand.

[2]The C++ version of BACK built by NSL, Paris.

3 Concluding Remarks

We have developed a semantic indexing mechanism that is crucially dependent on reasoning with descriptions as provided by terminological systems such as BACK. The indexing elements are arbitrarily complex descriptions logically related by subsumption and disjointness.

Compared with standard value-based indexes, this results in the following characteristics:
(1) A semantic index is inherently multidimensional since any combination of properties cast into a DL concept (i.e. an arbitrary query) can serve as an indexing element.
(2) As a structured concept the indexing elements are not just attribute values, but can be based on complex descriptions of related individuals.
(3) A semantic index as a whole is highly adaptable to patterns of usage. Indexing concepts can be added or removed at will, making it very dense and precise w.r.t to 'interesting sets' of individuals, or very sparse in other, less 'interesting' areas.
(4) Since the index is actually a set of partial descriptions for the indexed instances, lots of information (such as cardinality estimates) can be drawn from the index alone without accessing (possibly remote) individual descriptions at all.

References

[Bagnasco et al., 1991] Bagnasco, C., Spampinato, L., Vischi, C., The EUROPE BRÜCKE Language. Deliverable for ESPRIT P5120 AIMS, November 1991.

[Borgida and Brachman, 1993] Borgida, A., and Brachman, R.J., Loading Data into Description Reasoners. Proceedings of ACM SIGMOD '93, Washington DC, 1993

[Devanbu, 1993] Devanbu, P.T., Translating Description Logics to Information Server Queries. To appear in: Proceedings of Second Conference on Information and Knowledge Management (CIKM '93), Washington DC, 1993.

[Hoppe et al., 1993] Hoppe, Th., Kindermann, C., Quantz, J.J., Schmiedel, A., and Fischer, M., BACK V5 Tutorial and Manual. KIT Report 100, Department of Computer Science, Technische Universität Berlin, Berlin, Germany, March 1993.

[Illarramendi et al., 1991] Illarramendi, A., Blanco, J.M., and Goni, A., A Uniform Approach To Design a Federated System Using BACK. Proceedings Terminological Logic Users Workshop, KIT Report 95, Technische Universität Berlin, October 1991

[Mays et al., 1991] Mays, E., Lanka, S., Dionne, B., and Weida, T., A Persistent Store for Large Shared Knowledge Bases. IEEE Transaction on Knowledge and Data Engineering, Vol. 3, No. 1, March 1991

Position Paper for Description Logic Workshop

Lin Padgham
Department of Computer and Information Science
Linköping University, S-581 83 Linköping, Sweden
email: lin@ida.liu.se

1 Introduction

I am basically interested in description logics because I am interested in classification as a very general purpose commonsense reaoning mechanism, which humans use, and which can be effectively exploited in computer systems, to make them more "intelligent". I became interested in description logics when I was involved in research in defeasible inheritance reasoning. As I looked for applications for testing defeasible inheritance I found that classification and default reasoning seemed often to be required together. I thus became interested in integrating defaults into description logics. My published papers in this area are [5; 4; 3].

I have also more recently been working on the addition of composite objects to description logics, with the view that the part-of relation is also a structuring mechanism (with some similarities to is-a) which allows for general purpose representation and reasoning strategies. I have a paper at KR'94, with my PhD student Patrick Lambrix, on this topic [2].

Other members of my research group have done some preliminary work on addition of temporal constructs to description logics. I have been peripherally involved in this work [1].

In summary, my interest in description logics can be said to be in the area of conceptual extensions to current description logics. I am also interested in real applications of description logics [4] and their use together with database systems.

2 Defaults

My work on integrating defaults into description logics, went initially in two directions. The first was a comprehensive medical diagnosis application where I worked together with my then PhD student, Tingting Zhang. This application represented default information about diseases, and used both default inheritance, and classification in the reasoning engine [4]. The use of defaults for this application was clearly necessary as almost all of the knowledge base consisted of default information. The combination of the two reasoning methods, appears encouraging in the sense that the application was quite successful, making the same diagnoses as real doctors on non-controversial cases. On controversial cases the system behaved similarly to doctors, but we had no way of knowing which doctor's diagnosis to regard as correct.

In connection with this work I also worked on a representation of the Tbox when defaults are involved, and algorithms for Tbox classification as well as realisation. Unlike standard description logics, different algorithms seemed to be required for classification in the Tbox and its analog, realisation, in the Abox [4].

The default inheritance and the classification algorithms were run sequentially in the above application, rather than in an integrated fashion. The system relied on interaction with the user to recover from possible wrong default conclusions drawn by the default inheritance reasoner. This was satisfactory for this application but is not an adequate long term solution. As a first step in developing an algorithm capable of correct integrated default inheritance and classification I did some work with Bernhard Nebel on extending my previous work, (including an approximately linear algorithm), on default inheritance, to a simple description logic with combined classification and default inheritance [3].

I am currently supervising a PhD student (Niclas Wahllöf), who is attempting to integrate defaults into the CLASSIC system. The aim is to find computationally acceptable ways of integrating defaults into CLASSIC in a theoretically disciplined manner.

3 Composite Objects and Part-Of Relation

I have for a number of years been concerned with composite objects in my system oriented research in intelligent information systems. I have more recently been working on incorporating reasoning about composite objects into description logics. It seems that the ability to reason about wholes and parts, is a very general purpose human reasoning mechanism, which would be worthwhile to formalise, and which should be usable in a wide variety of applications. The kind of reasoning which we typically do involves such things as the ability to break down composite objects into their parts at varying levels of detail, to abstract parts which are the same from different composite objects, and to conceptually place together parts to make wholes. Inheritance of attributes between parts and wholes (in both directions) is also common.

I have been working together with Patrick Lambrix on specifying a description logic theory and language which supports inference of parts from the existence of wholes, and allows description of composite objects in terms of definitional parts (necessary and sufficient), and the required relationships between those parts. On the basis of these definitions, composite objects can be inferred to exist if all of the parts exist and they are in the correct relationship to each other. (This work is presented at KR-94 [2]).

In order to support inference about the existence of composite objects, as well as other kinds of inference regarding parts and wholes, we have defined a module hierarchy, based on the usability of an object in building a larger composite object. Like the is-a hierarchy, when new objects are inserted, links may need to be added to and deleted from existing objects, in order to maintain a compact representation. We have defined a notion of "most-composite-module" which is analogous to most-specific-subsumer, and defines which modules in the hierarchy an object should be linked to. This hierarchy can then be traversed in the search for possible composite objects.

In the general case there may be multiple ways of combining parts to form wholes. We define three possible levels to which compositional inferencing may reasonably proceed. A full compositional extension is an Abox where no further compositional inference can be made based on the definitions in the Tbox and the existence of Abox parts/individuals. For any given Abox and Tbox, there may be multiple compositional extensions. Credulous compositional extensions are those compositional extensions which are preferred according to some very general preferences (preference for forming more specific and more complex composite objects). A skeptical compositional conclusion is defined to be the intersection of the credulous compositional extensions, and represents the inferences about which there is no ambiguity once the general preferences for the credulous extensions are applied. A skeptical compositional conclusion is not necessarily an extension, in that it may be possible to make further compositional inference. In such a case it would be ambiguous as to which compositional inference should be made.

References

[1] Lambrix, P., Rönnquist, R., 'Terminological Logic Involving Time and Evolution: A Preliminary Report', *Proceedings of the Seventh International Symposium on Methodologies for Intelligent Systems - ISMIS 93*, pp 162-171, 1993.

[2] [PL94] Padgham, L., Lambrix, P., 'A Framework for Part-Of Hierarchies in Terminological Logics', *Principles of Knowledge Representation and Reasoning: Proceedings of the Fourth International Conference - KR 94*, 1994.

[3] Padgham, L., Nebel, B., 'Combining Classification and Nonmonotonic Inheritance Reasoning: A First Step', *Proceedings of the Seventh International Symposium on Methodologies for Intelligent Systems - ISMIS 93*, pp 132-141, 1993.

[4] Padgham, L., Zhang, T., 'A Terminological Logic with Defaults: A Definition and an Application', *Proceedings of the International Conference on Artificial Intelligence - IJCAI 93*, pp 662-668, 1993.

[5] [ZP90] Zhang, T., Padgham, L., 'A Diagnosis System Using Classification in an Inheritance Net', *Proceedings of the International Symposium on Methodologies for Intelligent Systems*, Knoxville, USA, Oct 24-27 1990.

Knowledge Formal Specifications for Formal Verification: a Proposal based on the Integration of Terminological Logics with other Logical Formalisms

Marie-Christine ROUSSET
L.R.I, U-A CNRS 410- Universite Paris-Sud
91405 ORSAY CEDEX- FRANCE
e-mail: mcr@lri.lri.fr
phone: (33-1) 69 41 61 97; Fax: (33-1) 69 41 65 86

1 Introduction

In this paper, we focus on the problem of formally specifying knowledge bases in order to formally verify them. The knowledge bases related to real-world applications are complex because they regroup large pieces of knowledge of different types. Two levels of knowledge are generally distinguished: the domain knowledge and the reasoning knowledge. As the domain knowledge is the basis for reasoning, it is especially crucial to verify that this knowledge does not contain any anomaly before reasoning on it: a sound and sophisticated reasoning has no interest if it is applied to wrong data. Our approach for specifying domain knowledge consists in first identifying the different types of knowledge that have to cohabit and then of choosing adequate logical formalisms to specify each of them. In this way, we take advantage of existing and well-known logical formalisms but we have to take special care with their integration. As our aim is knowledge verification, we do not want to limit the logical formalisms to their representational aspect, but we want to use the associated inferential services in order to support some verifications. In section 2, we first point out three different and complementary aspects that can be explicited from a domain knowledge, and the logical formalisms chosen to specify them. In section 3, we focus on terminological logics, and we show how they can be used to formally specify descriptive knowledge. In section 4, we show how taking into account these different aspects leads to the definition of the global consistency of the whole knowledge base. We show how to integrate some existing methods of consistency checking of a rule base and of a terminology.

2 Different logical formalisms for specifying different types of knowledge

The major lesson learned from the first generation expert systems is that a knowledge base has to contain different kinds of knowledge which can have different roles for the reasoning task. This obvious statement has led to second generation expert systems ([STEELS85]), which have as common denominator the distinction between shallow and deep knowledge. Further distinctions can be made, highlighting structural, behavioral, functional, causal knowledge. Our approach consists in regrouping these different types of knowledge according to the main existing logical formalisms: relation-centered, object-centered, function-centered (the word "function" is taken here in the logical sense). We propose then to distinguish three kinds of knowledge that we have respectively called deductive, descriptive and equational. These three kinds of knowledge are relevant in many real-world applications, and there already exists appropriate logical formalisms for them.

Deductive knowledge is the core of most existing expert systems, where it is encoded in a specific rule-based language associated to a specific inference engine. Deductive knowledge can cover different types of knowledge expressing behavior rules, deductive links between some properties of some domain concepts, or heuristic rules of thumb. These different types of knowledge have to be exploited differently by the reasoning task. However, as pieces of domain knowledge, they can be formally specified in a sublanguage of first order logic with good computational properties, which ensure that the computation of the deductive closure is feasible and efficient. This is the case for rule bases composed of logical implications on literals without functional symbols.

Descriptive knowledge states the multi-faceted structure of the objects of the domain. Different aspects of structure have to be taken into account: the organization of the different concepts as a hierarchy of specialization, the structural description of composite components, their functional description, and their topological description are just some of the many aspects that contribute to the whole descriptive knowledge. Some of these aspects can be encoded in frame-based or object-oriented languages. Their formal specification is another problem which requires formal properties that frame-based or object-oriented languages lack. We think that terminological logics are good candidates for formally specifying descriptive knowledge because of their formal semantics supporting clearly identified inferential services.

Equational knowledge states equality (or inequality) constraints on some properties of domain concepts. Such a knowledge can be specified as equations on functional terms, on which rewriting techniques allow computations. In this paper, we simply mention this type of knowledge without further detailing it, because our current work is focussed on the descriptive knowledge and its integration with deductive knowledge.

3 Specification of descriptive knowledge in terminological logics

Terminological logics are obviously appropriate to specify one of the aspects of structuration: the organization of different concepts into a hierarchy of specialization. The point is to take into account other aspects of descriptive knowledge such as structural description of composite components, their functional description, their topological description, etc. The approach we advocate consists in translating each of the different aspects of structuration by means of generic roles: has-component, has-function, connected-to, etc. Such roles associated to "range" restrictions allow the explicitation of different links of structuration between domain concepts. For instance, "any element of class A has a component of class B, and is connected to an element of class C" can be specified in the declaration of the concept A by the subterm:
ATLEAST(1, has-component and RANGE(B)) AND
ATLEAST(1, connected-to and RANGE(C))
The relationship between the different structuration links can be expressed by means of inference schemas that have to be integrated into the inferential services offered by the terminological systems. The important point is that, as they are generic, this can be done once and for all, before any specific domain is considered. For instance, the three following inference schemas have to be implemented for the composition link.

Schema 1: Transitivity:
Natural syntax:

A has a component of class B,
B has a component of class C
$\vdash$ A has a component of class C

Corresponding terminological syntax:

A $\preceq$ ATLEAST(1, has-component and RANGE(B))
B $\preceq$ ATLEAST(1, has-component and RANGE(C))
$\vdash$ A $\preceq$ ATLEAST(1, has-component and RANGE(C))

Schema 2:
Natural syntax:

A is a sub-class of B
B has a component of class C
$\vdash$ A has a component of class C

Corresponding terminological syntax:

A $\preceq$ B
B $\preceq$ ATLEAST(1, has-component and RANGE(C))
$\vdash$ A $\preceq$ ATLEAST(1, has-component and RANGE(C))

Schema 3:
Natural syntax:

A has a component of class B
B is a sub-class of C
$\vdash$ A has a component of class C

Corresponding terminological syntax:

A $\preceq$ ATLEAST(1, has-component and RANGE(B))
B $\preceq$ C
$\vdash$ A $\preceq$ ATLEAST(1, has-component and RANGE(C))

The two last schemas do not need to be implemented: they are directly taken into account by the subsumption algorithms (providing they are complete). The implementation of the first one is also superfluous in terminological systems that allow the definition of transitive closure for roles. More details on our approach for accounting for composition links in terminological logics are given in [BOUALI92] and [HORS93b].

4 Global consistency of domain knowledge

In the previous sections, we have highlighted different types of knowledge, deductive, descriptive and equational knowledge. For a complex domain, these different types of knowledge are interrelated. Their interaction affects the global inferential potentialities of the domain knowledge. From a knowledge verification point of view, and especially for consistency checking, two points must be outlined.

First, the distinction between the different types of knowledge can be taken as an advantage, because it can lead to specific and clearly identified consistency checking methods. In section 4.1, we briefly recall the definition and the method we proposed ([ROUSSET88], [ROUSSET93b] for rule base consistency. In section 4.2, our recent work ([ROUSSET93a]) about consistency and compatibility of concepts specified in terminological logics is summarized.

Second, independently consistency checking of the different types of knowledge is not sufficient to guarantee the global consistency of the whole domain knowledge. Taking into account their interaction is necessary.

We illustrate this point on the integration of a rule base and a terminology related to the same domain. On one hand, the set of predicates used in the rule base represents a part of the basic domain vocabulary. On the other hand, the set of concepts of a terminology defines another part of the basic domain vocabulary, by highlighting the structuring links between domain concepts. Some elements are common, that is, some concepts and roles defined in the terminology are used as predicates in the literals of the rule base. The fact that they represent the same entities has to be taken into account and has important consequences on the global domain knowledge consistency. As a matter of fact, the knowledge expressed in the terminology can be seen as the structuration of the literals used in the rule base.

In section 4.3, we summarize how our method for rule base consistency checking can be extended by taking into account the descriptive knowledge in order to obtain a method for checking the global consistency of a rule base and a terminology taken as a whole.

4.1 Rule base consistency checking

The definition of rule base consistency issued from [ROUSSET88] relies on the explicit declaration of integrity constraints by means of incompatibility rules of the form "L1, L2,...,Ln → False".

With respect to a set IR of incompatibility rules:
Literals are IR-incompatible iff they make fire an incompatibility rule a set of literals is IR-consistent iff it doesn't contain IR-incompatible literals;
A rule base is IR-consistent iff from any IR-consistent set of initial literals (i.e, which do not appear in any rule conclusion), it gives a deductively closed set of literals that is IR-consistent.

In [ROUSSET88] and [ROUSSET93b], we have shown how ATMS-like algorithms ([DE KLEER86]) can be used to get a complete method for rule base consistency checking. ATMS-like algorithms allow the computation of the label of False, which is a minimal disjunctive form of all the initial literals causing the deduction of False from the set of assertions composed of both the rules of the checked rule base and the integrity constraints.

4.2 Descriptive knowledge consistency checking

The formal semantics of terminological logics provides simple and rigorous definitions that meet the intuition of the notions of consistency and compatibility applied to concepts.

> A concept C is consistent iff there exists an interpretation I such as I(C) is not empty.
> C1, C2, ..., Cn are compatible iff there exists an interpretation I such as the intersection of I(C1),I(C2), ..., I(Cn) is not empty.
> A terminology T is consistent iff all the concepts defined in T are consistent.

Our approach for consistency checking of a terminology is detailed in [ROUSSET93a]. It follows a "knowledge engineering" point of view, aiming at pointing out and explaining the causes of the detected inconsistencies, especially when they are not obvious. As a matter of fact, from the point of view of the user, some relations of subsumptions or disjointness between concepts are obvious, some others are more "hidden" and then difficult to figure out and to understand. The obvious cases can be considered as self-explanatory, the others have to be explained. In the framework of the descriptive language we consider, which contains the most classical terminological constructs (negation being restricted to primitive concepts), we have stated two cases of obvious disjointness:
(i) C and NOT(C), where C is a primitive concept
(ii) ATLEAST (k , R) and ATMOST(k-1 , R)
Regarding obvious subsumptions relations, we have considered they correspond to structural subsumptions.
For complex concepts, having the form of conjunction of concepts, consistency checking can be reduced to checking the disjointness of a subset of concepts. The basic idea of our method for checking the disjointness of concepts is to try to reach an obvious case of disjointness, possibly resulting from some steps of subsumptions, according to the following proposition:

> Two concepts C1 and C2 are disjoint iff either they are obviously disjoint or there exists two obviously disjoint concepts D1 and D2, such that C1 $\preceq$ D1 and C2 $\preceq$ D2.

The important point is that we get the subsumption relations which are not obvious supported by some rules. These rules make explicit some complex subsumptions relations that are not pointed out by structural and superficial subsumption checking.

Examples of some of these rules:

> atleast(k1,R and range(C1))
> and atleast(k2, R and range(C2))
> → atleast(k1+k2 , R)
> (if C1 and C2 are disjoint)
>
> atleast(k,R and range(C1))
> and all(R and range(C2) , C)
> → atleast(k , R and range(C))
> (if C1 $\preceq$ C2)
>
> atleast(k1,R) and atmost(k2,R and range(C))
> → atleast(k1-k2,R and range(not(C))
> (k1 $\geq$ k2)
>
> all(R,C1) and atmost(k,R and range(C2))
> → atmost(k,R)
> (if C1 $\preceq$ C2)
>
> all(R,C1) and all(R,C2)
> → atmost(0,R)
> (if C1 and C2 are disjoint)

Given a complex conjunction of concepts, these rules can be exploited in a backward-chaining way, in order to construct a trace explanation of the complex subsumption steps resulting to a case of obvious disjointness case. Such an approach can be associated to an existing description system, either to explain an inconsistency, which has been previously detected by the system, or to palliate the incompleteness of the associated subsumption algorithm, in order to find inconsistencies which have not been detected by the system.

4.3 Consistency checking of the descriptive and deductive knowledge, taken as a whole.

If it has been checked and proven consistent, a terminology can be considered as a reliable knowledge which, possibly associated to a set of integrity constraints, can then be used as a reference for rule base consistency checking. It can be used to prove that some literals of the rule base are incompatible. This leads to the extension of the incompatibility definition of literals built on predicates belonging both to the vocabulary of the rule base and of the terminology.

> Literals p1(X), p2(X), ..., pn(X) are T-incompatible, with respect to a terminology T,

iff p1, p2, ..., pn are disjoint concepts in the terminology T.
A set of literals is T-consistent, with respect to a terminology T, iff it doesn't contain T-incompatible literals.

This leads to the following definition of global consistency which is an extension of the rule base consistency definition integrating the notion of structural compatibility with respect to a terminology.

A Rule Base is globally consistent, with respect to a set of integrity constraints IR and to a terminology T, iff from any IR-consistent and T-consistent set of initial literals, it gives a deductively closed set of literals that is IR-consistent and T-consistent.

The calculation of the no-good sets (i.e, the label of False) and the method of incompatibility checking of concepts can be merged in order to obtain a global constructive method of consistency checking. The incompatibility checking of concepts can be used both to simplify the no-good sets and to create new no-good sets. More details are given in [HORS93a].

5 Conclusion and perspectives

In this paper, we have laid the foundations of a proposal for formal knowledge specifications aimed at making formal verifications possible. Our approach consists of taking advantage of existing logical formalisms appropriate to specify different types of knowledge. We have presented our first results, which deal with the integration of deductive and descriptive aspects, from the point of view of consistency checking. Several extensions will have to be considered: The equational knowledge has to be taken into account and integrated with the two other aspects. Equations on functional terms can be seen as constraints on some literals used in a rule base related to the same domain. The integration of the three aspects (deductive, descriptive, equational) has to be considered in a larger perpective going beyond consistency checking. The challenge is to clearly define the global inferential services which can be offered by putting together three specification formalisms each having their own specific inferential services. We really think that there would be great benefit in making the effort of precisely and rigorously specifying knowledge, very early in the knowledge-based systems life cycle. Our work is a basic block in line with this general viewpoint.

References

[BOUALI92] F. BOUALI. Les systemes terminologiques et la relation partie-de. Technical report, Paris-Sud University, 1992.

[DE KLEER86] J. DE KLEER. An assumption-based truth-maintenance system. *Artificial Intelligence*, 28:127–224, 1986.

[HORS93a] M.C ROUSSET; P. HORS. A formal framework for structured and deductive knowledge consistency checking. *Proceedings of the IJCAI workshop on Validation of Knowledge-Based Systems*, 1993.

[HORS93b] P. HORS. Prise en compte du lien de composition. Technical report, E.D.F, june 1993.

[ROUSSET88] M.C ROUSSET. On the consistency of knowledge bases: the covadis system. *Proceedings of European Conference on Artificial Intelligence*, 1988.

[ROUSSET93a] P. HORS; M.C ROUSSET. Consistency of structured knowledge: a formal framework based on description logics. *Proceedings of EUROVAV 93*, 1993.

[ROUSSET93b] S. LOISEAU; M.C ROUSSET. Formal verification of knowledge bases focused on consistency: two experiments based on atms techniques. *International Journal of Expert Systems: Research and Applications*, 6(3), 1993.

[STEELS85] L. STEELS. Second generation expert systems. *Future Generation of Computer Systems*, 1(4), 1985.

Generalized Feature Graphs: A Kernel for Description Logic Implementations

Gunnar Teege
Institut für Informatik, TU München
80290 München, Germany
email: teege@informatik.tu-muenchen.de

Abstract

If feature graphs are extended in a simple way they provide a basis for implementing arbitrary description logic formalisms. We describe the extension and the resulting language and we briefly show how to use it as a kernel for description logics.

1 Introduction

A common characteristics of the feature graphs introduced by Rounds and Kasper [Rounds and Kasper,1986] and all extensions thereof is the fact that the graphs must be rooted. The root node automatically represents the object depicted by the feature graph (the *depictee*). Hence, feature graphs always describe the depictee with the help of other objects which are functionally dependent on the depictee.

This mechanism was sufficient in the original application of feature graphs for representing grammatical information in unification grammars. Later, feature graphs found their way to knowledge representation and have been integrated with description logic formalisms [Nebel and Smolka,1990]. However, description logics provide non-functional roles as additional means for characterizing the depictee in a description. Hence, the feature graphs can only be used for representing a part of a description. They must be supplemented or replaced by other representation formalisms, such as constraint systems [Hollunder and Nutt,1990].

On the other hand, feature graphs are a well understood, compact representation formalism. They can easily be visualized, and standard algorithms for graph manipulation are available for implementing operations. Therefore we propose to generalize feature graphs in a way that they can be used for a much broader part of description logics. In this form they provide a kernel for implementing description logic systems. Mechanisms which provide more expressiveness can be layered on top of this kernel.

Here we describe the extension for the simplest possible kind of feature graphs and characterize the expressiveness of the resulting generalized feature graphs in the domain of description logics. A similar extension is possible for more elaborate feature graph formalisms, such as typed feature graphs [Carpenter,1992], or feature graphs with negation [Dawar and Vijay-Shanker,1990]. A more detailed treatment of the extension is given in [Teege,1994a]. There we also define a linear notation for the generalized feature graphs. Due to space limitations we omit this notation here.

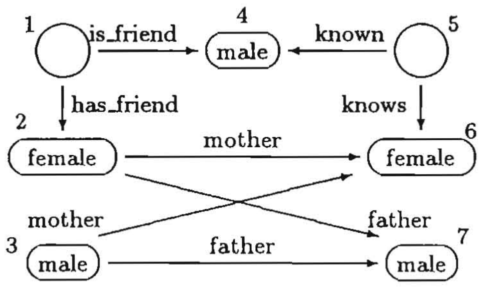

Figure 1: A Generalized Feature Graph

2 Generalized Feature Graphs

The simplest possible feature graphs are rooted directed graphs with labeled edges and the restriction that the labels of all edges originating in a single node must be pairwise different ("deterministic directed graph"). We generalize this formalism as follows. We drop the restriction of rootedness and that of connectedness for the graph. The depictee must be explicitly specified in the form of a tuple of nodes in the graph. We call the resulting description language $\mathcal{RICE}$-$\mathcal{TL}$. It was originally developed as part of the knowledge representation system $\mathcal{RICE}$ [Teege,1991].

An example of an extended feature graph is depicted in Figure 1. In the example we use atomic node labels for the sake of clarity. However, they can immediately be replaced by corresponding features leading to additional isolated nodes. An example of a description is the node tuple $(3,2)$. If the features are interpreted as suggested by their names this tuple roughly corresponds to the description "a son and his sister who has a boy friend known by her mother".

It is easy to see that $\mathcal{RICE}$-$\mathcal{TL}$ is equivalent to the following kind of description specification in monadic logic. The graph is equivalent to a formula

$$\exists X : \bigwedge_i x_{i1} = f_i(x_{i2})$$

where X is a vector of variables, the x_{ij} are variables occurring in X and the f_i are features. The vector X contains exactly one variable for every graph node. For every edge there is a corresponding equation in the conjunction. A description is equivalent to an arbitrary tuple of variables in X.

Thus, in $\mathcal{RICE}$-$\mathcal{TL}$ a description denotes a tuple of objects selected from a set of objects with arbitrary mutual dependencies specified by unary functions.

Additionally, $\mathcal{RICE}$-$\mathcal{TL}$ supports descriptions of complex functional dependencies which are constructed from unary functions. A description of a complex functional dependency is given by every pair (N_1, N_2) of node tuples, where each node in N_2 can be reached from at least one node in N_1 via a directed path in the graph. We call tuple pairs of this kind *complex features*, and we call the tuple N_1 the *domain tuple* and N_2 the *value tuple* of the complex feature. A complex feature is the description of a functional dependency between two object tuples which can be reduced to unary functional dependencies between tuple components. Thus, complex features are a straightforward generalization of the (atomic) features from single objects to object tuples. An example of a complex feature in the graph in Figure 1 is the pair with domain tuple (2, 3) and value tuple (7, 6).

A sound formal treatment of $\mathcal{RICE}$-$\mathcal{TL}$ is achieved by using category theory. The generalized feature graphs form a category and the same holds true for node tuples and complex features. The subsumption test corresponds to a morphism test, and meet and join operations correspond to product and coproduct operations in the categories. Standard algorithms existing for the graph category operations can be used for implementing these description operations. However, it should be noted that the subsumption test is of non-polynomial complexity in $\mathcal{RICE}$-$\mathcal{TL}$. A full categorical treatment of $\mathcal{RICE}$-$\mathcal{TL}$ will be given in [Teege,1994b].

As usual, the category structure on the descriptions induces a lattice structure where product and coproduct play the role of the lattice operations. In this lattice it is possible to define the subtraction operation [Teege,1994c] for $\mathcal{RICE}$-$\mathcal{TL}$ and implement it with the help of graph operations.

3 Expressiveness of $\mathcal{RICE}$-$\mathcal{TL}$

We now give a short impression of the expressiveness of $\mathcal{RICE}$-$\mathcal{TL}$ compared to usual feature graphs and to usual description logics.

The main extension with respect to usual feature graphs is the possibility to specify descriptions of feature *values*. Normally, a feature, such as "color" can only be used for building descriptions of things *having* a specific color, such as "blue thing". In $\mathcal{RICE}$-$\mathcal{TL}$ we can also build descriptions of things *being* the color of specific things, such as "color of a car". This case corresponds to a graph where the depictee is the destination node of an edge.

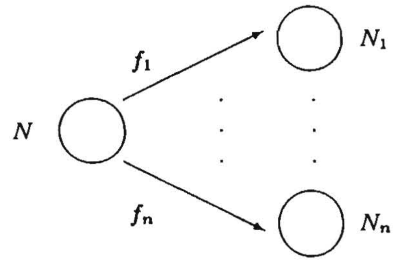

Figure 2: A Generalized Feature Graph for Representing a Role

Descriptions of feature values can be used for modeling arbitrary n-ary roles (relations). As usual, we replace an n-ary role by n features $f_1, \ldots, f_n$. In the corresponding graph, which is depicted in Figure 2, n edges labeled with these features lead from a single node N to n different nodes N_i. The node N represents the set of relation instances. Every node N_i represents the set of possible fillers of the i-th role component. The n-tuple $T = (N_1, \ldots, N_n)$ of all nodes N_i represents all tuples of objects related by the role, hence T is a good choice for a description of the role.

A generalized feature graph may contain arbitrary structures involving roles represented with the help of features. Hence, it is possible to construct complex structural descriptions where objects are related by several different roles. This case is similar to the structural descriptions and role value maps already defined in KL-ONE [Brachman and Schmolze,1985]. An example can be found in Figure 1. Node 4 is linked via the two relation nodes 1 and 5 to the rest of the graph.

However, structural descriptions of roles have usually been defined in a way as to specify conditions for *sets of objects* related to a single object. This cannot be expressed in $\mathcal{RICE}$-$\mathcal{TL}$. As we have seen in Section 2 $\mathcal{RICE}$-$\mathcal{TL}$ allows only existential quantification, no universal quantification. In particular, $\mathcal{RICE}$-$\mathcal{TL}$ only supports exists-restrictions for roles. All-restrictions, such as defined in most description logics, are beyond the scope of $\mathcal{RICE}$-$\mathcal{TL}$. The same holds true for number-restrictions which involve constraints on object sets as well.

4 $\mathcal{RICE}$-$\mathcal{TL}$ as a Kernel for Description Logics

We have seen that $\mathcal{RICE}$-$\mathcal{TL}$ on its own does not cover the expressiveness of usual description logics. However, the language fully supports a restricted kind of descriptions, i.e., those which are formulated with the help of existentially quantified objects and unary functional dependencies between the objects. For this kind of descriptions $\mathcal{RICE}$-$\mathcal{TL}$ easily supports arbitrary complex descriptions and it provides standard implementations for a wide range of operations and constructors. Hence, $\mathcal{RICE}$-$\mathcal{TL}$ is useful as a basis for description logic implementations. It can be regarded as being a "kernel" on top of which more elaborate mechanisms for description formulation can be built as additional layers.

A possible extension layer supports the formulation of constraints on object sets, such as all-restrictions or number-restrictions. The main idea is to use explicit descriptions of object sets. These descriptions may then be constrained in a similar way in which descriptions of single objects or object tuples are constrained by the graph structure in $\mathcal{RICE}$-$\mathcal{TL}$. One approach to object set representations on top of $\mathcal{RICE}$-$\mathcal{TL}$ are sets of object or object tuples mapped by a (complex) feature to a common image.

Another example of a layer on top of $\mathcal{RICE}$-$\mathcal{TL}$ is a layer supporting concept definitions (also called terminological axioms) or, more generally, equivalence declarations on descriptions for expressing facts. A layer of this kind has been described in [Teege,1991].

Finally, $\mathcal{RICE}$-$\mathcal{TL}$ is similar to a central part of the conceptual graph mechanism developed by Sowa [Sowa,1984]. An n-tuple of graph nodes corresponds to an n-adic abstraction using a conceptual graph with no additional information associated with the nodes. Thus, $\mathcal{RICE}$-$\mathcal{TL}$ can also be used as a kernel for conceptual graph implementations. The additional mechanisms of conceptual graphs can be layered on top of $\mathcal{RICE}$-$\mathcal{TL}$ in the same way as the other extensions mentioned previously.

Compared to other implementation techniques for Description Logics, such as constraint systems, graphs provide a more direct representation of the description structure. Hence implementations of structural operations can use the explicit links instead of searching and matching for finding connected parts. This is similar to the use of connection graphs in theorem proving [Kowalski,1975]. However, links in connection graphs are an additional indexing facility which does not change the semantics of the connected formulas. The links in extended feature graphs are instead an integral part of the term-forming mechanism. The semantics of a description heavily depends on the links in the corresponding graph.

References

[Brachman and Schmolze, 1985]
Brachman, R. J., Schmolze, J. G.: An Overview of the KL-ONE Knowledge Representation System. Cognitive Science **9**, 171–216 (1985)

[Carpenter, 1992]
Carpenter, B.: The Logic of Typed Feature Structures. Cambridge University Press, 1992

[Dawar and Vijay-Shanker, 1990]
Dawar, A., Vijay-Shanker, K.: An Interpretation of Negation in Feature Structure Descriptions. Computational Linguistics **16**:1, 11–21 (Mar. 1990)

[Hollunder and Nutt, 1990]
Hollunder, B., Nutt, W.: Subsumption Algorithms for Concept Languages. Deutsches Forschungszentrum für Künstliche Intelligenz DFKI, Research Report RR-90-04, 1990

[Kowalski, 1975]
Kowalski, R.: A Proof Procedure Using Connection Graphs. Journal of the ACM **22**:4, 572–595 (Oct. 1975)

[Nebel and Smolka, 1990]
Nebel, B., Smolka, G.: Representation and Reasoning with Attributive Descriptions. In: Bläsius, K.-H., Hedtstück, U., Rollinger, C.-R. (eds.): Sorts and Types in Artificial Intelligence. Lecture Notes in Artificial Intelligence 418, Berlin: Springer, 1990, pp. 112–139

[Rounds and Kasper, 1986]
Rounds, W., Kasper, R.: A Complete Logical Calculus for Record Structures Representing Linguistic Information. In: Proceedings of the 1st IEEE Symposium on Logic in Computer Science, 1986, pp. 38–43

[Sowa, 1984]
Sowa, J. F.: Conceptual Structures. Addison-Wesley, Reading, Mass., 1984

[Teege, 1991]
Teege, G.: Ein System zur Repräsentation von deklarativem Gebietswissen für intelligente Tutorsysteme. Technische Universität München, PhD thesis, 1991

[Teege, 1994a]
Teege, G.: Abstract Descriptions in the RICE Knowledge Representation Formalism. Inst. für Informatik, Technische Universität München, Munich, Germany, Technical Report I9407, Feb. 1994

[Teege, 1994b]
Teege, G.: A Categorical Treatment of the Description Logic RICE-TL. Inst. für Informatik, Technische Universität München, Munich, Germany, Technical Report, 1994, (in preparation)

[Teege, 1994c]
Teege, G.: Making the Difference: A Subtraction Operation for Description Logics. In: Doyle, J., Sandewall, E., Torasso, P. (eds.): Principles of Knowledge Representation and Reasoning: Proc. of the 4th International Conference (KR94), San Francisco, CA: Morgan Kaufmann, 1994

International Workshop on Description Logics

Organizers: Franz Baader, Maurizio Lenzerini, Werner Nutt, Peter F. Patel-Schneider

D-94-10
Document