International Workshop on Description Logics

Rome, June 2-3, 1995

Alex Borgida
Department of Computer Science
Rutgers University, New Brunswick, NJ, U.S.A.
borgida@cs.rutgers.edu

Maurizio Lenzerini
Dipartimento di Informatica e Sistermistica
Università di Roma, "La Sapienza", Italy
lenzerini@dis.uniroma1.it

Daniele Nardi
Dipartimento di Informatica e Sistermistica
Università di Roma, "La Sapienza", Italy
nardi@dis.uniroma1.it

Bernhard Nebel
University of Ulm
and DFKI, Saarbrücken
nebel@informatik.uni-ulm.de

Preface

This volume contains the collection of papers that were presented at the 1995 International Workshop on Description Logics, that was held on June 2 and 3, 1995, at the Centro Congressi of Università di Roma "La Sapienza", Italy.

The Workshop was organized in 8 sessions, with 3 sessions including contributions on theoretical issues about Description Logics:

- Extensions to Description Logics,
- Integration with other formalisms,
- Foundations,

and 3 sessions on the development of knowledge representation systems based on Description Logics:

- Connections to Databases,
- Applications,
- Systems.

The remaining two sessions were devoted to the demonstrations of applications and prototypes, and to a final summary of the Workshop and perspectives on the field.

The Workshop was organized with the contribution of Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", and partially supported by a grant from the Consiglio Nazionale delle Ricerche. Special thanks go to Franz Baader for the financial contribution to the student support, and to Teresa Cremona, Giuseppe De Giacomo, Francesco M. Donini, Riccardo Rosati, and Andrea Schaerf for their contribution to the organization.

Contents

Applications

Systems

Hierarchical Plans in a Description Logic of Time and Action*

Alessandro Artale
Ladseb-CNR
I-35020 Padova PD, Italy
artale@ladseb.pd.cnr.it

Enrico Franconi
IRST
I-38050 Povo TN, Italy
franconi@irst.itc.it

Abstract

A formal language for representing and reasoning about time, actions and plans in a uniform way is presented. We employ an action representation in the style of Allen, where an action is represented by describing what is true while the action is occurring. In this sense, an action is defined by means of temporal constraints on the world states, which pertain to the action itself, and on other possible qualifications for the action occurring over time, while a plan is seen just as a complex action. Therefore, there is no difference between actions and plans in this framework. A distinction between action types and individual actions is supported by the formalism. In this paper, we show how to extend the language with a decomposition operator in a way of distinguishing the different actions composing a plan and not just the different temporal qualifications for the plan itself. Thus, a plan is a hierarchical structure made of its distinct component subparts.

The formal representation language used in this paper is a member of the family of description logics, and it is provided with a well founded syntax, semantics and calculus. The classification and recognition tasks, together with the basic subsumption procedure, form the basis for action management. An action description can be automatically classified into a taxonomy; an action instance can be recognized to take place at a certain moment from the observation of what is happening in the world during a time interval.

1 Introduction

The structured logic of time and action considered in this paper [Artale and Franconi,1993; Artale and Franconi,1994a; Artale *et al.*,1994; Artale and Franconi,1994b; Artale and Franconi,1995] permits dealing with time, actions and plans in a uniform way. As opposed to the most common approaches to modeling actions as state change – e.g., the formal models based on *situation calculus* [McCarthy and Hayes,1969], the STRIPS planning system [Lifschitz,1987] – where actions are instantaneous and defined as functions from one state to another, we prefer to explicitly introduce the notion of time by admitting that actions take time, like in [Allen,1991]. We argue that this view of an action is closer to naturally occurring processes. Different actions can be concurrent or may overlap in time; effects of overlapping actions can be different from the sum of their individual effects; effects may not follow the action but more complex temporal relations may hold.

Starting from a formal language able to express temporally related objects – inspired by the work of [Schmiedel,1990] – actions are represented through temporal constraints on the world states, which pertain to the action itself, like in [Allen,1991]. With respect to [Allen,1991], our formalism has a clear distinction between the language for expressing action types (the conceptual level) and the language for expressing individual actions (the assertional level). Informally, plans are built by temporally relating action types in a compositional way using the temporal constructors available in the language; similar ideas were pursued by [Weida and Litman,1992]. In this way, since the temporal relationships are proper operators of the basic language, the distinction between actions and plans disappears. As a matter of fact, we do not need distinct languages for objects and states representation, for time representation, for actions representation, and for plans representation.

While actions describe how the world is affected by their occurrence, plans are described as a collection of action types constrained by temporal relations. In this way, a plan can be graphically represented as a temporal constraint network, where nodes denote action types. At this level of representation, plans can be seen as *complex actions*: since actions composing a plan can be expanded, plans and actions are not structurally different. This distinction is further elaborated in this paper, where each action composing a plan is considered as a

*This work has been partially supported by the Italian National Research Council (CNR), projects "Sistemi Informatici e Calcolo Parallelo", "Pianificazione Automatica", "Robotica" and "Ontologic and Linguistic Tools for Conceptual Modeling".

C, D	$\rightarrow$	$A \mid$	(atomic concept)
		$\top \mid$	(top)
		$C \sqcap D \mid$	(conjunction)
		$C \sqcup D \mid$	(disjunction)
		$p \downarrow q \mid$	(agreement)
		$p : C$	(selection)
		$C@X \mid$	(temporal qualifier)
		$C[Y]@X \mid$	(temporal subst. qualifier)
		$\Diamond(X\ Y \cdots)\ \mathcal{R}_1 \mathcal{R}_2 \cdots \bullet C$	(temporal quantifier)
p, q	$\rightarrow$	$f \mid$	(atomic feature)
		$\star g \mid$	(atomic parametric feature)
		$p \circ q$	(feature chain)
$\mathcal{R}$	$\rightarrow$	$(X\ (R)\ Y)$	(temporal constraint)
R, S	$\rightarrow$	$R \ , \ S \mid$	(temporal disjunction)
		$\mathsf{s} \mid \mathsf{mi} \mid \mathsf{f} \mid \ldots$	(Allen temporal relations)
X, Y	$\rightarrow$	$\sharp \mid \mathrm{x} \mid \mathrm{y} \mid \ldots$	(temporal variables)

Figure 1: Syntax rules for the temporal concept language

step referring to a different individual action, and an appropriate function relates a plan to its steps.

The distinction made by the language between the conceptual and assertional aspect of the knowledge allows us to describe actions both at an abstract level (action type) and instance level (individual action). In this environment we exploit the *subsumption* calculus as the main inference tool for managing collections of action types. Given a set of observations of individual actions in the world the system is able to recognize which type of action has taken place at a certain time interval; this task is known as *recognition*. Action types are organized in a subsumption-based plan taxonomy, which can play the role of a plan library to be used for plan retrieval and individual plan recognition tasks [Kautz,1991]. In this way, we have refined the concept of what is currrently called *plan recognition*, by splitting it into the different tasks of *plan description classification* and *specific plan recognition with respect to a plan description*. In [Artale and Franconi,1994a] we proved the decidability and proposed an actual algorithm for the subsumption and recognition reasoning tasks in the basic language.

An interesting feature of this representational framework is its extensibility to cover different standard problems from the AI literature on actions. Homogeneous concepts can be introduced, for representing states whose properties holding at an interval do also hold at all subintervals [Artale *et al.*,1994]. We propose an approach to the *frame problem*, i.e., the problem to compute what is unchanged by an action. By introducing an *inertial* operator in the language we express the persistence of the truth of a proposition if there is no evidence of its falsity; with this approach the famous *Yale Shooting Problem* can be easily solved [Artale and Franconi,1994b]. We consider here the possibility to relate an action with more elementary actions, by *decomposing* it in partially ordered steps; this kind of representation is found in hierarchical planners.

2 The Formal Language

We briefly describe the basic description logic able to represent classes of individuals and their temporal relations – for a full discussion of the language please refer to [Artale and Franconi,1994a; Artale and Franconi,1995]. 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. 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).

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. 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 $\sharp$. Explicit temporal variables allow for complex temporal patterns to be specified. [Bettini,1993] proposes a variable-free temporal extension of a description logic, which can not express temporal patterns like $(y\ \mathsf{starts}\ x)(z\ \mathsf{finishes}\ x)(y\ \mathsf{meets}\ z)$. We believe that the introduction of variables in the language does not affect the principle of description logics which are considered as variable-free languages; in fact, variables are limited in the temporal realm.

We are going now to informally present the *meaning* of the terms of the language. 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 of individual actions of that type occurring at some interval.

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, 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 informal meaning of a concept with a temporal existential quantification can be understood with the following example in the action domain.

$$\texttt{Basic-Stack} \doteq \Diamond(x\ y)(x\ \mathsf{m}\ \sharp)(\sharp\ \mathsf{m}\ y).\ \big(\star\texttt{BLOCK} : (\texttt{OnTable}@x \sqcap \texttt{OnBlock}@y)\big)$$

`Basic-Stack` denotes, according to the definition (a terminological axiom), any action occurring at some inter-

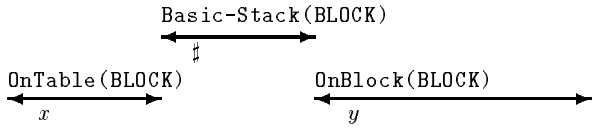


Figure 2: Temporal dependencies in the definition of the `Basic-Stack` action.

val involving a ⋆`BLOCK` that was once `OnTable` and then `OnBlock`. The ♯ 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 temporal constraints $(x\ \mathsf{m}\ \sharp)$ and $(\sharp\ \mathsf{m}\ y)$ state that the interval denoted by x should meet the interval denoted by ♯ – the occurrence interval of the action type `Basic-Stack` – and that ♯ should meet y. Figure 2 shows the temporal dependencies of the intervals in which the concept `Basic-Stack` holds. The parametric feature ⋆`BLOCK` plays the role of *formal* parameter of the action, mapping any individual action of type `Basic-Stack` to the block to be stacked, independently from time. Please note that, whereas the existence and identity of the ⋆`BLOCK` of the action is time invariant, it can be qualified differently in different intervals of time, e.g. the ⋆`BLOCK` is necessarily `OnTable` only during the interval denoted by x. On the other hand, the (non-parametric) feature `OnTable` may change its value over time – thus, representing a changing state.

The assertion `Basic-Stack`(i, a) says that a is an individual action of type `Basic-Stack` occurred at the interval i. Moreover, the same assertion implies that a is related to a ⋆`BLOCK` , say b, which is of type `OnTable` at some interval j, meeting i, and of type `OnBlock` at another interval l, met by i.

$$\begin{aligned}&\texttt{Basic-Stack}(i,a) \Longrightarrow\\ &\quad \exists b.\ \star\texttt{BLOCK}(a,b) \wedge\\ &\qquad \exists j,l.\ \big(\texttt{OnTable}(j,b) \wedge \texttt{OnBlock}(l,b) \wedge\\ &\qquad\quad \mathsf{m}(j,i) \wedge \mathsf{m}(i,l)\big)\end{aligned}$$

Now we show how from a series of observations in the world we can make action recognition – i.e. the task we have already called *specific plan recognition with respect to a plan description*. This is an inference service which computes if an *individual action* is an instance of an *action type* at a certain interval. Given the following state of affairs, describing a world where blocks can be on the table and/or on each other and where a generic individual action a is taking place at time interval i_a having the block *block-a* as its parameter:

$$\begin{aligned}&\star\texttt{BLOCK}(a, \textit{block-a}),\\ &\mathsf{m}(i_1, i_a),\ \texttt{OnTable}(i_1, \textit{block-a}),\\ &\mathsf{mi}(i_2, i_a),\ \texttt{OnBlock}(i_2, \textit{block-a})\end{aligned}$$

then, the system recognizes that in the context of a knowledge base Σ, composed by the above theory and the definition of the `Basic-Stack` concept, the individual action a is an instance of the concept `Basic-Stack` at the time interval i_a, i.e. $\Sigma \models \texttt{Basic-Stack}(i_a, a)$.

3 Decomposition of Actions

In this section we consider plans as hierarchical structures whose constituent actions could be seen as its decomposing steps. Functional relations decompose an individual plan to its steps, allowing us to take into account plans where the component actions do not share the same actual parameters. Suppose, for example, that we want represent the plan composed by a sequence of two Stack actions. This plan could be represented as:

$$\begin{aligned}&\texttt{Double-Stack} \doteq\\ &\quad \Diamond(x\ y)(x\ \mathsf{d}\ \sharp)(y\ \mathsf{d}\ \sharp)(x\ \mathsf{m}\ y).\\ &\qquad (\texttt{Basic-Stack}@x \sqcap \texttt{Basic-Stack}@y)\end{aligned}$$

According to this plan description, each individual plan – say for example a – of type `Double-Stack` has to be an instance of two `Basic-Stack` actions occurring at two "meeting" intervals, too.

$$\begin{aligned}&\texttt{Double-Stack}(i,a) \Longrightarrow\\ &\quad \exists j,l.\ \big(\texttt{Basic-Stack}(j,a) \wedge\ \texttt{Basic-Stack}(l,a) \wedge\\ &\qquad \mathsf{d}(j,i) \wedge \mathsf{d}(l,i) \wedge \mathsf{m}(j,l)\big)\end{aligned}$$

This representation does not take into account the case where the actual parameters involved in the two `Basic-Stack` actions are different. In fact, there is only one action involved in the former definition: a. The individual action a is qualified as a `Double-Stack` plan at the current interval, it is a `Basic-Stack` action at the interval j, and it is a `Basic-Stack` action again at the interval l. The crucial point is that the parameters involved in the `Stack` actions are described by means of *parametric features* – functions that remain invariant during time – which in this case apply to the *same* individual `Basic-Stack` action – a. So, the parameter of the action a must be the same at the intervals j and l, and the case of sequential actions of the same type but with different parameters can not be expressed. In the case of the `Basic-Stack` actions, the actual ⋆`BLOCK` parameter value should be the same at both j and l intervals, disallowing the possibility of expressing the plan composed by the sequence of two `Basic-Stack` actions manipulating two different blocks.

The described behavior is the consequence of having a fully substitutional framework for plan definitions. Since plans are seen just as complex actions, the *parts* composing a plan are just different qualifications on the same individual entity. A solution to this problem is to associate distinct individual actions for each action step occuring in a given plan, in such a way that parametric features, being applied on distinct action instances, can introduce different parameter values. This is realized by relating the plan to the component action types by means of different functions, using the *decomposition* operator. We add the following syntactic rules, to be used for plan definitions:

$$\begin{array}{rcll} C, D & \rightarrow & C@@X & \text{(decomposition)} \\ p, q & \rightarrow & p@@X \downarrow q@@Y & \text{(temporal agreement)} \end{array}$$

where $C@@X$ means that C is a step in a plan and the temporal agreement specifies equality constraints between paths of features defined in different steps of a plan (see, e.g. [Weida and Litman,1994]).

More formally, we add the following semantic equations to the semantics of the basic description logic [Artale and Franconi,1994a]:

$$(C@@X)^{\mathcal{I},\mathcal{S}}_{\mathcal{V},t,\mathcal{H}} = \{a \in dom(X^{\mathcal{S}}) \mid X^{\mathcal{S}}(a) \in C^{\mathcal{I},\mathcal{S}}_{\mathcal{V},\mathcal{V}(X),\mathcal{H}}\}$$

$$(p@@X \downarrow q@@Y)^{\mathcal{I},\mathcal{S}}_{\mathcal{V},t,\mathcal{H}} = \{a \in dom(X^{\mathcal{S}} \circ p^{\mathcal{I},\mathcal{S}}_{\mathcal{V},\mathcal{V}(X),\mathcal{H}}) \cap dom(Y^{\mathcal{S}} \circ q^{\mathcal{I},\mathcal{S}}_{\mathcal{V},\mathcal{V}(Y),\mathcal{H}}) \mid X^{\mathcal{S}} \circ p^{\mathcal{I},\mathcal{S}}_{\mathcal{V},\mathcal{V}(X),\mathcal{H}}(a) = Y^{\mathcal{S}} \circ q^{\mathcal{I},\mathcal{S}}_{\mathcal{V},\mathcal{V}(Y),\mathcal{H}}(a)\}$$

where $\cdot^{\mathcal{S}}$ – called the *Step Interpretation* – is a function associating a newly generated function to each interval introduced by a decomposition operator.

The representation of the `Double-Stack` plan in the extended language is:

$$\begin{array}{l} \texttt{Double-Stack} \doteq \\ \quad \Diamond(x\ y)\ (x\ \texttt{d}\ \sharp)(y\ \texttt{d}\ \sharp)(x\ \texttt{m}\ y). \\ \qquad (\texttt{Basic-Stack}@@x \sqcap \texttt{Basic-Stack}@@y) \end{array}$$

Since the nodes x and y of the conceptual temporal constraint network introduce distinct individual actions, we can now distinguish the different instances of the two `Basic-Stack` actions occurring in distinct intervals:

$$\begin{array}{l} \texttt{Double-Stack}(i,a) \Longrightarrow \\ \quad \exists b,c,j,l.\ (\texttt{X}(a,b) \wedge \texttt{Basic-Stack}(j,b) \wedge \\ \qquad \texttt{Y}(a,c) \wedge \texttt{Basic-Stack}(l,c) \wedge \\ \qquad \texttt{d}(j,i) \wedge \texttt{d}(l,i) \wedge \texttt{m}(j,l)) \end{array}$$

where `X` and `Y` are newly generated functions, mapping the individual plan a to its individual steps b and c. It is now possible the accomplishment of a plan moving two different blocks during two `Basic-Stack` events in sequence.

The case of *simultaneous* occurrences of the same action type with different parameters can be expressed in the extended language. The plan where two possibly different blocks are moved by two `Basic-Stack` actions at the same time is:

$$\begin{array}{l} \texttt{Sim-Double-Stack} \doteq \\ \quad \Diamond(x\ y)(x\ \texttt{d}\ \sharp)(y\ \texttt{d}\ \sharp)(x\ =\ y). \\ \qquad (\texttt{Basic-Stack}@@x \sqcap \texttt{Basic-Stack}@@y) \end{array}$$

We want now to address the case where different individual actions share common parameters or where a plan is made of the repetition, in different intervals, of the same individual action. We exploit the *temporal agreement* construct – $p@@X \downarrow q@@Y$, which specifies equality constraints between features introduced in different intervals. In the following example, the `Same-Block-Double-Stack` plan is constituted of two `Basic-Stack` actions moving the same $\star$`BLOCK`:

$$\begin{array}{l} \texttt{Same-Block-Double-Stack} \doteq \\ \quad \Diamond(x\ y)\ (x\ \texttt{d}\ \sharp)(y\ \texttt{d}\ \sharp)(x\ \texttt{m}\ y). \\ \qquad (\texttt{Basic-Stack}@@x \sqcap \texttt{Basic-Stack}@@y \sqcap \\ \qquad \star\texttt{BLOCK}@@x \downarrow \star\texttt{BLOCK}@@y) \end{array}$$

`Same-Double-Stack` is a plan where a `Stack` action occurs twice in distinct intervals, with the same fillers for the actual parameters:

$$\begin{array}{l} \texttt{Same-Double-Stack} \doteq \\ \quad \Diamond(x\ y)\ (x\ \texttt{d}\ \sharp)(y\ \texttt{d}\ \sharp)(x\ \texttt{m}\ y). \\ \qquad (\texttt{Basic-Stack}@@x \sqcap \texttt{Basic-Stack}@@y \sqcap \\ \qquad @@x \downarrow @@y) \end{array}$$

The construct $@@X \downarrow @@Y$ – a particular case of temporal agreement – imposes an equality constraint between the functions associated to the nodes `X` and `Y` , so that the two `Basic-Stack` actions are instantiated by the same individual object; thus, the actual parameters associated to the `Basic-Stack` actions are equal.

Finally we propose an example where the actions composing the plan have different types. Consider, in the cooking domain, the following example:

$$\begin{array}{l} \texttt{Heat-Noodles} \doteq \\ \quad \Diamond(x\ y)\ (x\ \texttt{d}\ \sharp)(y\ \texttt{d}\ \sharp)(x\ \texttt{b},\texttt{m}\ y). \\ \qquad (\texttt{Make-Noodles}@@x \sqcap \texttt{Heat}@@y \sqcap \\ \qquad \star\texttt{Agent}@@x \downarrow \star\texttt{Agent}@@y) \end{array}$$

The definition specifies that the plan has two steps, `Make-Noodles` and `Heat`, instantiated by distinct individual actions, while the temporal agreement forces the $\star$`Agent`s of the two steps to be the same.

References

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

[Artale and Franconi, 1993] Alessandro Artale and Enrico Franconi. A unified framework for representing time, actions and plans. In F. D. Anger, H. W. Guesgen, and J. van Benthem, editors, *Workshop Notes of the IJCAI-93 Workshop on Temporal and Spatial Reasoning*, pages 193–217, Chambery, France, August 1993. Also in the Workshop Notes of the IJCAI-93 Workshop on Object-Based Representation System, pages 32–44; a shorter version appears in the Proceedings of the Italian Planning Workshop 1993 (IPW'93), pages 167–172, Roma Italy, September 1993.

[Artale and Franconi, 1994a] Alessandro Artale and Enrico Franconi. A computational account for a description logic of time and action. In *Proc. of the 4 th International Conference on Principles of Knowledge Representation and Reasoning (KR-94)*, pages 3–14, Bonn, Germany, May 1994.

[Artale and Franconi, 1994b] Alessandro Artale and Enrico Franconi. Persistent properties in a description logic of time and action. In *Working Notes of the*

*AI*IA Temporal Reasoning Workshop 1994*, pages 9–12, Parma, Italy, September 1994.

[Artale and Franconi, 1995] Alessandro Artale and Enrico Franconi. Time, actions and plans representation in a description logic. *International Journal of Intelligent Systems*, 1995. To appear.

[Artale *et al.*, 1994] Alessandro Artale, Claudio Bettini, and Enrico Franconi. Homogeneous concepts in a temporal description logic. In *Proceedings of the International Workshop on Description Logics*, pages 36–41. DFKI, Saarbrücken, Bonn, Germany, May 1994.

[Bettini, 1993] Claudio Bettini. A family of temporal terminological logics. In *Proc. of the third Congress of the Italian Association for Artificial Intelligence, AIIA-93*, LNAI. Springer-Verlag, 1993.

[Kautz, 1991] Henry A. Kautz. A formal theory of plan recognition and its implementation. In James F. Allen, Henry A. Kautz, Richard N. Pelavin, and Josh D. Tenenberg, editors, *Reasoning about Plans*, chapter 2, pages 69–126. Morgan Kaufmann, 1991.

[Lifschitz, 1987] Vladimir Lifschitz. On the semantics of STRIPS. In *The 1986 Workshop on Reasoning about Actions and Plans*, pages 1–10. Morgan Kaufman, 1987.

[McCarthy and Hayes, 1969] J. McCarthy and P. J. Hayes. Some philosophical problems from the standpoint of Artificial Intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 4, pages 463–502, Edinburgh, UK, 1969. Edinburgh University Press.

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

[Weida and Litman, 1992] Robert Weida and Diane Litman. 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 (KR-92)*, pages 282–293, Cambridge, MA, October 1992.

[Weida and Litman, 1994] Robert Weida and Diane Litman. Subsumption and recognition of heterogeneous constraint networks. In *Proceedings of CAIA-94*, 1994.

Terminological Logics with Modal Operators

Franz Baader
Lehr- und Forschungsgebiet Theoretische Informatik, RWTH Aachen,
Ahornstraße 55, 52074 Aachen, Germany

Armin Laux
German Research Center for Artificial Intelligence (DFKI GmbH)
Stuhlsatzenhausweg 3, 66123 Saarbrücken, Germany

Abstract

Terminological knowledge representation formalisms can be used to represent objective, time-independent facts about an application domain. Notions like belief, intentions, and time which are essential for the representation of multi-agent environments can only be expressed in a very limited way. For such notions, modal logics with possible worlds semantics provides a formally well-founded and well-investigated basis. This paper presents a framework for integrating modal operators into terminological knowledge representation languages. These operators can be used both inside of concept expressions and in front of terminological and assertional axioms. We show that satisfiability in the extended language is decidable, provided that all modal operators are interpreted in the basic logic K, and that the increasing domain assumption is used.

1 Introduction

Terminological knowledge representation languages in the style of KL-ONE [Brachman and Schmolze,1985] have been developed as a structured formalism to describe the relevant concepts of a problem domain and the interactions between these concepts. Various terminological systems have been designed and implemented that are based on the ideas underlying KL-ONE (see [Woods and Schmolze,1992] for an overview). Representing knowledge of an application domain with such a kind of system amounts to introducing the terminology of this domain via concept definitions, and then describing (an abstraction of) the relevant part of the "world" by listing the facts that hold in this part of the world. In a traditional terminological system, such a description is rigid in the sense that it does not allow for the representation of notions like time, or beliefs of different agents. In systems modeling aspects of intelligent agents, however, intentions, beliefs, and time-dependent facts play an important role. Modal logics with possible worlds semantics is a formally well-founded and well-investigated framework for the representation of such notions. The present paper is concerned with integrating modal operators (for time, belief, etc.) into a terminological formalism. The first task is to find an appropriate semantics for the combined language. In addition, if such a language should be used in a system, one must design algorithms for the important inference problems (such as consistency of knowledge bases) for the language.

Several approaches have been proposed for the combination of terminological formalisms with notions like time or beliefs. A very simple possibility to represent beliefs of agents is realized in the partition hierarchy SB-PART [Kobsa,1989], which is an extension of the SB-ONE system. In this approach, each agent may have its own set of terminological axioms ($TBox$), and these TBoxes can be ordered hierarchically. However, this extension lacks a formal semantics and it does not allow for representing properties of belief, such as introspection, or interactions between beliefs of different agents. A more formal approach is used in M-KRYPTON [Saffiotti and Sebastiani,1988], where a sub-language of the KRYPTON representation language is extended by modal operators B_i, which can be used to represent the beliefs of agent i. Properties of beliefs are taken into consideration by using the well-known modal logic KD45. Due to the undecidable base language, however, [Saffiotti and Sebastiani,1988] just introduces a formal semantics, without giving any inference algorithms for the extended language. In [Schild,1991], it has been shown that terminological systems already have a strong connection to modal logic. In fact, the concept language $\mathcal{ALC}$ is nothing but a syntactic variant of the propositional multi-modal logic $\mathsf{K}_{(m)}$. Building upon this observation, [Schild,1993] augments $\mathcal{ALC}$ by tense operators. The two approaches that come next to the one we shall introduce below are described in [Laux,1994a; Laux,1994b] and in [Ohlbach and Baader,1993]. Both extend $\mathcal{ALC}$ by modal operators, but with different emphasis. The differences between these approaches and ours are clarified in the next section.

2 Classification

When extending a terminological knowledge representation language by modalities for belief, time, etc. one has

various degrees of freedom. Before describing the specific choices made in this article, we shall informally explain the different alternatives.

For simplicity, assume that we are interested in time and belief operators only. Thus, in addition to the objects we have time points and belief worlds. This means that the domain of an interpretation is the Cartesian product $\vec{D} = D_{object} \times D_{time} \times D_{belief}$ of the set of objects, the set of time points, and the set of belief worlds. Concepts are no longer just sets of objects; their interpretation also depends on the actual belief world and time point. Thus, they can be seen as subsets of $\vec{D}$, and not just as subsets of D_{object}. Roles operate on objects, whereas modalities for time (like *future* or *tomorrow*) operate on time points, and modalities for belief (like *bel-John*) operate on belief worlds. As for concepts, however, the interpretation of roles and modalities depends on all dimensions. Thus, a role *loves* is interpreted as a function from $\vec{D}$ into $2^{D_{object}}$, which relates any individual in D_{object} (say John) with a set of individuals (the individuals John loves), but this set depends on the actual time point and belief world. Modalities like *future* are treated analogously. Modal operators can now be used both inside of concept expressions and in front of concept definitions and assertions. For example, we can describe the set of individuals that love a woman that—according to John's belief—is pretty by the concept expression $\exists\, \mathit{loves}.(\mathit{Woman} \sqcap [\mathit{bel\text{-}John}]\mathit{Pretty})$, and we can express that—according to John's belief—a happy husband is one married to a woman whom he (John) believes to be pretty by the terminological axiom

$$[\mathit{bel\text{-}John}](\mathit{Happy\text{-}husband} = \exists\, \mathit{married\text{-}to}.(\mathit{Woman} \sqcap [\mathit{bel\text{-}John}]\mathit{Pretty})).$$

The assertion $[\mathit{bel\text{-}John}]\langle \mathit{future}\rangle(\mathit{Peter\ married\text{-}to\ Mary})$ says that John believes that, at some point in the future, Peter will be married to Mary.

With the usual interpretation of the Boolean operators, of value and exists restrictions on roles, and of box and diamond operators for the modalities, this yields a multi-dimensional version of the multi-modal logic K_m. As described until now, this logic is a strict sub-language of the one introduced in [Ohlbach and Baader,1993]. The restriction lies in the fact that, unlike in [Ohlbach and Baader,1993], we do not consider roles and modalities that have a complex structure (such as $[\mathit{wants}]\mathit{own}$, where the modality *wants* is used to modify the role *own*). There are several reasons why this approach is not yet satisfactory. First, the object and the other dimensions are treated analogously. This means, for example, that the interpretation of the modality *future* depends not only on the actual time point, but also on the current object and the belief world. Whereas the dependence from the belief world may seem to be quite reasonable, it is rather counterintuitive that the future time points reached from time t_0 are different, depending on whether we are interested in the individual Sue or Mary. Thus, it seems to be more appropriate to treat the object dimension in a special way: whereas the interpretation of roles should depend on the actual time point etc., the interpretation of modalities should not depend on the object under consideration.

The need for a special treatment of the object dimension can also be motivated by considering the semantics of concept definitions (and assertions). In [Ohlbach and Baader,1993], concept definitions are required to hold for all objects, time points, and belief worlds. This is a straightforward generalization of the treatment of definitions in terminological languages, where a definition $C = D$ must hold for *all* objects, i.e., in a model of $C = D$ all objects o must satisfy that o belongs to the interpretation of C iff it belongs to the interpretation of D. For the other dimensions, however, this differs from the usual definition of models in modal logics, where a formula is only required to hold in one world.

Another problem is that not only the roles, but also all the other modalities are just interpreted in the basic logic K, i.e., they are not required to satisfy specific axioms for belief or time. In the present paper, we shall not take into account this last aspect, but we shall treat the object dimension in a special way, thus eliminating the problems mentioned above. In [Laux,1994a; Laux,1994b] both aspects are considered. However, modal operators are not allowed to occur inside of concept expressions, which considerably simplifies the algorithmic treatment of the formalism. The difference to [Ohlbach and Baader,1993] is, on the one hand, the special treatment of the object dimension. In addition, [Ohlbach and Baader,1993] does not consider assertions, and even though concept definitions are introduced, they are not handled by the satisfiability algorithm. On the other hand, [Ohlbach and Baader,1993] allows for very complex roles and modalities, which are not considered here.

3 Syntax and Semantics of $\mathcal{ALC}_{\mathcal{M}}$

First, we present the syntax of our multi-dimensional modal extension of the concept language $\mathcal{ALC}$. As for $\mathcal{ALC}$, we assume a set of concept names, a set of role names, and a set of object names to be given. Beside the object dimension (which will be treated differently from the other dimensions), we assume that there are $\nu \geq 1$ additional dimensions (such as time points, epistemic alternatives, or intensional states). In each dimension, there can be several modalities, which can be used in box and diamond operators. For example, in the dimension *time points* we could have *future* and *tomorrow*, and in the dimension *belief worlds* we could have *belief-John* and *belief-Mary*. If o is a modality of dimension i we write $\mathit{dim}(o) = i$. In this case, $[o]$ and $\langle o\rangle$ are *modal operators* of dimension i.

Definition 3.1 (Syntax) Concepts *of $\mathcal{ALC}_{\mathcal{M}}$ are inductively defined as follows. Each concept name is a concept, and $\top$ and $\bot$ are concepts. If C and D are concepts, R is a role name, and o is a modality then* $C \sqcap D$ (concept conjunction), $C \sqcup D$ (concept disjunction), $\neg C$ (concept negation), $\forall R.C$ (value restriction),

$\exists R.C$ (exists restriction), $[o]C$ (box operator), *and* $\langle o\rangle C$ (diamond operator) *are concepts.*

Terminological axioms *of* $\mathcal{ALC}_\mathcal{M}$ *are of the form* $m\,(C = D)$ *where* C *and* D *are concepts of* $\mathcal{ALC}_\mathcal{M}$ *and* m *is a (possibly empty) sequence of modal operators.* Assertional axioms *of* $\mathcal{ALC}_\mathcal{M}$ *are of the form* $m\,(xRy)$ *or* $m\,(x : C)$ *where* x *and* y *are object names,* R *is a role name,* C *is a concept, and* m *is a (possibly empty) sequence of modal operators. An* $\mathcal{ALC}_\mathcal{M}$-formula *is either a terminological or an assertional axiom.*

Traditional terminological systems impose severe restrictions on the admissible sets of terminological axioms: *(1)* The concepts on the left-hand sides of axioms must be concept names, *(2)* concept names occur at most once as left-hand side of an axiom (unique definitions), and *(3)* there are no cyclic definitions. The effect of these restrictions is that terminological axioms are just macro definitions (introducing names for large descriptions), which can simply be expanded before starting the reasoning process. Unrestricted terminological axioms are a lot harder to handle algorithmically (see, e.g., [Buchheit *et al.*,1993]), but they are very useful for expressing constraints on concepts that are required to hold in the application domain. In the presence of modal operators, the requirement of having unique definitions is not appropriate anyway. For example, Peter may have a definition of *Happy-husband* that is quite different from John's definition. Thus, it is desirable to have different definitions $m_1\,(A = C)$ and $m_2\,(A = D)$ of the same concept name A for different modal sequences m_1 and m_2. Even though m_1 and m_2 are different, there can be interactions between these definitions. For example, m_1 could be of the form $\langle o\rangle$ and m_2 of the form $[o]$. Thus, it is not a priori clear how the requirement of "unique definitions" can be adapted to the case of terminological axioms with modal prefix. To avoid these problems, we consider the more general case where arbitrary axioms are allowed.

Let us now turn to the semantics of $\mathcal{ALC}_\mathcal{M}$. The modal operators will be interpreted by a Kripke-style possible worlds semantics. Thus, for each dimension i we need a set of possible worlds D_i. Modalities of dimension i correspond to accessibility relations on D_i, which may, however, depend on the other dimensions as well. Concepts and roles are interpreted in an object domain, but this interpretation also depends on the modal dimensions.

Definition 3.2 (Semantics) *A* Kripke structure $K = (\mathcal{W}, \Gamma, K_I)$ *consists of a set* $\mathcal{W}$ *of possible worlds, a set of accessibility relations* Γ*, and a* K*-interpretation* K_I *over* $\mathcal{W}$*, which are given as follows. First, the set of possible worlds* $\mathcal{W}$ *is the Cartesian product of non-empty domains* $D_1, \ldots, D_\nu$*, one for each dimension. Second,* Γ *contains for each modality* o *of dimension* i *an accessibility relation* γ_o*, which is a function* $\gamma_o : \mathcal{W} \to 2^{D_i}$*. We also write* $((d_1, \ldots, d_i, \ldots, d_\nu), (d_1, \ldots, d'_i, \ldots, d_\nu)) \in \gamma_o$ *for* $d'_i \in \gamma_o(d_1, \ldots, d_i, \ldots, d_\nu)$*. Finally, the* K*-interpretation* K_I *consists of a domain* Δ^{K_I} *and an interpretation function* $\cdot^{K_I}$*. The domain is the union of non-empty domains* $\Delta^{K_I}(w)$ *for all worlds* $w \in \mathcal{W}$*. The interpretation function associates (i) with each object name* x *an element* $x^{K_I} \in \Delta^{K_I}$*, (ii) with each concept name* A *and world* $w \in \mathcal{W}$ *a set* $(A, w)^{K_I} \subseteq \Delta^{K_I}(w)$*, (iii) with the top concept and the bottom concept the sets* $(\top, w)^{K_I} = \Delta^{K_I}(w)$ *and* $(\bot, w)^{K_I} = \emptyset$ *(for each world* w*), and (iv) with each role name* R *and world* $w \in \mathcal{W}$ *a binary relation* $(R, w)^{K_I} \subseteq \Delta^{K_I}(w) \times \Delta^{K_I}(w)$*.*

Note that the interpretation of object names does not depend on the particular world (i.e., we are using so-called "rigid designators"), whereas the interpretation of concept and role names depends on the world. For a given world w, the interpretation of a concept A (resp. of a role R) in w is a subset of (resp. a binary relation on) the domain $\Delta^{K_I}(w)$ associated with w. The interpretation of concept names and roles is expanded to the concept forming operators as follows: If C and D are concepts, R is a role, and w is a world, then

$$
\begin{aligned}
(C \sqcap D, w)^{K_I} &= (C, w)^{K_I} \cap (D, w)^{K_I} \\
(C \sqcup D, w)^{K_I} &= (C, w)^{K_I} \cup (D, w)^{K_I} \\
(\neg C, w)^{K_I} &= \Delta^{K_I}(w) \setminus (C, w)^{K_I} \\
(\forall\, R.C, w)^{K_I} &= \{\delta \in \Delta^{K_I}(w) \mid \delta' \in (C, w)^{K_I} \\
&\qquad \text{for each } \delta' \text{ with } (\delta, \delta') \in (R, w)^{K_I}\} \\
(\exists\, R.C, w)^{K_I} &= \{\delta \in \Delta^{K_I}(w) \mid \delta' \in (C, w)^{K_I} \\
&\qquad \text{for some } \delta' \text{ with } (\delta, \delta') \in (R, w)^{K_I}\} \\
([o]\, C, w)^{K_I} &= \{\delta \in \Delta^{K_I}(w) \mid \delta \in (C, w')^{K_I} \\
&\qquad \text{for each } w' \text{ with } (w, w') \in \gamma_o\} \\
(\langle o\rangle\, C, w)^{K_I} &= \{\delta \in \Delta^{K_I}(w) \mid \delta \in (C, w')^{K_I} \\
&\qquad \text{for some } w' \text{ with } (w, w') \in \gamma_o\}
\end{aligned}
$$

Observe that, for each concept C and world w, we have $(C, w)^{K_I} \subseteq \Delta^{K_I}(w)$. Two $\mathcal{ALC}_\mathcal{M}$ concepts C and D are called *equivalent* iff for all Kripke structures $K = (\mathcal{W}, \Gamma, K_I)$ and all worlds $w \in \mathcal{W}$ we have $(C, w)^{K_I} = (D, w)^{K_I}$.

Now we can define under which conditions an $\mathcal{ALC}_\mathcal{M}$-formula F is *satisfied in a Kripke structure* $K = (\mathcal{W}, \Gamma, K_I)$ *and a world* $w \in \mathcal{W}$, written as $K, w \models F$, by induction on the length of the modal prefix:

$$
\begin{aligned}
K, w \models C = D \quad &\text{iff} \quad (C, w)^{K_I} = (D, w)^{K_I}, \\
K, w \models x : C \quad &\text{iff} \quad x^{K_I} \in (C, w)^{K_I}, \\
K, w \models xRy \quad &\text{iff} \quad (x^{K_I}, y^{K_I}) \in (R, w)^{K_I}, \\
K, w \models [o]\, G \quad &\text{iff} \quad K, w' \models G \\
&\qquad \text{for each world } w' \text{ with } (w, w') \in \gamma_o, \\
K, w \models \langle o\rangle\, G \quad &\text{iff} \quad K, w' \models G \\
&\qquad \text{for some world } w' \text{ with } (w, w') \in \gamma_o.
\end{aligned}
$$

Here G is an $\mathcal{ALC}_\mathcal{M}$-formula, C, D are concepts, x, y are object names, R is a role name, and o is a modality. A set $\{F_1, \ldots, F_n\}$ of $\mathcal{ALC}_\mathcal{M}$-formulas is *satisfiable* iff there exists a Kripke structure $K = (\mathcal{W}, \Gamma, K_I)$ and a world $w_0 \in \mathcal{W}$ such that $K, w_0 \models F_i$ for $i = 1, \ldots, n$. In this case we write $K \models F_1, \ldots, F_n$.

Even though we have introduced a domain $\Delta^{K_I}(w)$ for each world w, we have not yet said anything about the relationship between the domains of different worlds. In the simplest approach, the domains $\Delta^{K_I}(w_1)$ and $\Delta^{K_I}(w_2)$ of each pair w_1, w_2 of worlds are independent of each other. This approach is known as *varying domain assumption*. In most cases, however, it is more

reasonable to assume certain relationships between the domains of different worlds. The most commonly used approach is the so-called *increasing domain assumption*, where $\Delta^{K_I}(w) \subseteq \Delta^{K_I}(w')$ if the world w' is accessible from the world w, i.e., there exists a modality o such that $(w, w') \in \gamma_o$. The advantage of this approach is that domain elements that have been introduced in w can also be referred to in all worlds that are accessible from w, i.e., domain elements do not "vanish" when we move from one world to another. As a special case, the *constant domain assumption* is sometimes used, where the domains $\Delta^{K_I}(w_1)$ and $\Delta^{K_I}(w_2)$ are identical for all worlds w_1 and w_2. Finally, the *decreasing domain assumption* can be used to express that new domain elements cannot arise when moving from one world to another one.

As an example that demonstrates the consequences of changing the requirements on the relationship between domains of worlds, consider the $\mathcal{ALC}_\mathcal{M}$-formulas $x : (\langle o \rangle\ C)$ and $\langle o \rangle\ (x : C)$, where x is an object name, o is a modality, and C is a concept. For a Kripke structure $K = (\mathcal{W}, \Gamma, K_I)$ and a world $w \in \mathcal{W}$ we have (i) $K, w \models x : \langle o \rangle\ C$ iff $x^{K_I} \in \Delta^{K_I}(w)$ and there exists a world w' such that $(w, w') \in \gamma_o$ and $x^{K_I} \in (C, w')^{K_I}$, and (ii) $K, w \models \langle o \rangle\, (x : C)$ iff there exists a world w' such that $(w, w') \in \gamma_o$, $x^{K_I} \in \Delta^{K_I}(w')$, and $x^{K_I} \in (C, w')^{K_I}$. Thus, the main difference is that in the first case x^{K_I} is required to be in $\Delta^{K_I}(w)$, whereas this is not necessary in the second case. The reason is that, in the first case, x must belong to the interpretation of a concept in world w. In the second case, x is just required to be in the interpretation of a concept in the successor world.

If we assume just increasing domains, it is possible that $x^{K_I} \in \Delta^{K_I}(w')$, but $x^{K_I} \notin \Delta^{K_I}(w)$. Hence it may be the case that $K, w \models \langle o \rangle (x : C)$, but $K, w \not\models x : (\langle o \rangle C)$. If we assume constant domains, however, it holds that $\Delta^{K_I}(w) = \Delta^{K_I}(w')$, and thus $K, w \models x : (\langle o \rangle\ C)$ iff $K, w \models \langle o \rangle\ (x : C)$.

In the full paper [Baader and Laux,1994] we discuss the algorithmic problems that are caused by assuming constant domains. It turned out that testing satisfiability with respect to the constant domain assumption requires more than a straightforward modification of the satisfiability algorithm for increasing domains (which will be presented in the next section). In fact, until now we did not succeed in finding an appropriate modification.

In this paper we shall restrict our attention to increasing domains. Furthermore, we assume that all terminological axioms are of the form $m\,(C = \top)$, where C is a concept and m is a (possibly empty) sequence of modal operators. As in the case of $\mathcal{ALC}$ without modalities, it is easy to verify that this can be done without loss of generality.

Lemma 3.3 *Let $K = (\mathcal{W}, \Gamma, K_I)$ be a Kripke structure, w be a world in $\mathcal{W}$, m be a (possibly empty) sequence of modal operators, and C, D be concepts. Then $K, w \models m\,(C = D)$ iff $K, w \models m\left((C \sqcap D) \sqcup (\neg C \sqcap \neg D) = \top\right)$.*

4 Testing Satisfiability of $\mathcal{ALC}_\mathcal{M}$-formulas

We present an algorithm for testing satisfiability of a finite set $\{F_1, \ldots, F_n\}$ of $\mathcal{ALC}_\mathcal{M}$-formulas.[1] To keep notation simple we assume (without loss of generality) that concepts are in *negation normal form*, i.e., negation signs occur immediately in front of concept names only. Our calculus for testing satisfiability of $\mathcal{ALC}_\mathcal{M}$-formulas is based on the notions of labeled $\mathcal{ALC}_\mathcal{M}$-formulas and of world constraint systems. A *labeled $\mathcal{ALC}_\mathcal{M}$-formula* consists of an $\mathcal{ALC}_\mathcal{M}$-formula F together with a label l, written as $F \,||\, l$. The label l is a syntactic representation of a world in which F is required to hold. A *world constraint* is either a labeled $\mathcal{ALC}_\mathcal{M}$-formula or a term $l \bowtie_o l'$, where l, l' are labels and $\bowtie_o$ is a syntactic representation of the accessibility relation of modality o. A *world constraint system* is a finite, non-empty set of world constraints.

A Kripke structure $K = (\mathcal{W}, \Gamma, K_I)$ satisfies a world constraint system W iff there is a mapping α that maps labels in W to worlds in $\mathcal{W}$ such that (i) $K, \alpha(l) \models F$ for each world constraint $F \,||\, l$ in W, and (ii) $(\alpha(l), \alpha(l')) \in \gamma_o$ for each world constraint $l \bowtie_o l'$ in W. A world constraint system W is *satisfiable* iff there exists a Kripke structure satisfying W. In order to test satisfiability of a set $\{F_1, \ldots, F_n\}$ of $\mathcal{ALC}_\mathcal{M}$-formulas we translate this set into the world constraint system $W_0 = \{x_0 : \top \,||\, l_0, F_1 \,||\, l_0, \ldots, F_n \,||\, l_0\}$, where x_0 is a new object name not occurring in $\{F_1, \ldots, F_n\}$, and l_0 is an arbitrary label (which is intended to represent the real world). We say the world constraint system W_0 is *induced by* $\{F_1, \ldots, F_n\}$. It is easy to verify that $\{F_1, \ldots, F_n\}$ is satisfiable iff W_0 is satisfiable. The world constraint $x_0 : \top \,||\, l_0$ can obviously be satisfied by any Kripke structure. This constraint is necessary to guarantee that the domains $\Delta^{K_I}(w)$ of the canonical Kripke structure constructed in the proof of completeness are non-empty (see the full paper [Baader and Laux,1994]).

The $\mathcal{ALC}_\mathcal{M}$-satisfiability algorithm takes as input a world constraint system W_0 that is induced by a finite set of $\mathcal{ALC}_\mathcal{M}$-formulas. It successively adds new world constraints to W_0 by applying several propagation rules, which will be defined later. A world constraint system that is induced by a finite set of $\mathcal{ALC}_\mathcal{M}$-formulas, or that is obtained by a finite sequence of applications of propagation rules to an induced system, will be called *derived system*.

In the following, we use the letters x, y, z to denote object names, l to denote labels, A, B to denote concept names, C, D to denote concepts, and R to denote role names. If necessary, these letters will have an appropriate subscript. Before introducing the propagation rules in a formal way (in Figure 1), let us first describe the underlying ideas on an intuitive level. The rules that handle the usual $\mathcal{ALC}$ concept forming operators are well-known and rather straightforward (see,

[1] It is easy to see that all the other interesting inference problems (like the subsumption or the instance problem) can be reduced to this problem.

e.g., [Baader and Hollunder,1991]). In order to illustrate the rules that handle modalities and world constraints of the form $C = \top \,||\, l$, suppose that the $\mathcal{ALC}_{\mathcal{M}}$-formula $\langle o \rangle\ (B = \top)$ is given, where o is a modality of some dimension. In order to test satisfiability of this $\mathcal{ALC}_{\mathcal{M}}$-formula, we start with the induced world constraint system $W_0 = \{x_0 : \top \,||\, l_0, \langle o \rangle\ (B = \top) \,||\, l_0\}$. By definition, W_0 is satisfiable iff there is a Kripke structure $K = (\mathcal{W}, \Gamma, K_I)$, a mapping α, and a world $w_0 = \alpha(l_o) \in \mathcal{W}$ such that $x_0^{K_I} \in \Delta^{K_I}(w_0)$ and $K, w_0 \models \langle o \rangle\ (B = \top)$. Since $K, w_0 \models \langle o \rangle\ (B = \top)$ iff $K, w_1 \models B = \top$ for some world w_1 with $(w_0, w_1) \in \gamma_o$, the $\rightarrow_\Diamond$ rule adds the world constraints $l_0 \bowtie_o l_1$ and $B = \top \,||\, l_1$ to W_0, where l_1 is a new label. This yields the new world constraint system $W_1 = W_0 \cup \{l_0 \bowtie_o l_1, B = \top \,||\, l_1\}$. Because of the semantics of $\mathcal{ALC}_{\mathcal{M}}$-formulas, we furthermore know that $K, w_1 \models B = \top$ iff $\delta \in (B, w_1)^{K_I}$ for all $\delta \in \Delta^{K_I}(w_1)$. By the increasing domain assumption, $x_0^{K_I} \in \Delta^{K_I}(w_0)$ implies $x_0^{K_I} \in \Delta^{K_I}(w_1)$. Summing up, we must guarantee that $x_0^{K_I} \in (B, w_1)^{K_I}$ and therefore must add the world constraint $x_0 : B \,||\, l_1$ to W.

More generally, we say that an object name x is *relevant for label l* (in a world constraint system W) iff there is a label l' occurring in W such that (*i*) W contains a world constraint of the form $x : C \,||\, l'$, $xRy \,||\, l'$, or $yRx \,||\, l'$, and (*ii*) l is *accessible from l'*, i.e., either l is l' or there are world constraints $l' \bowtie_{o_1} l_1, \ldots, l_{n-1} \bowtie_{o_n} l$ in W for some modalities $o_1, \ldots, o_n$. Now, if x is relevant for l and there is a world constraint $C = \top \,||\, l$ in W for some concept C, then the $\rightarrow_=$ rule adds $x : C \,||\, l$ to W (unless this world constraint is already contained in W). In our example, this rule applies to W_1, and it yields the world constraint system $W_2 = W_1 \cup \{x_0 : B \,||\, l_1\}$. To W_2 no more propagation rules are applicable, and—as shown in [Baader and Laux,1994]—we can use this system to construct a Kripke structure that satisfies the $\mathcal{ALC}_{\mathcal{M}}$-formula $\langle o \rangle\ (B = \top)$. A world constraint system to which no more propagation rules are applicable will be called *complete*.

Termination of the propagation rule applications can only be guaranteed if applicability of the usual rule for handling exists restrictions is restricted in an appropriate way. This is due to the presence of axioms of the form $C = \top$. To illustrate this problem, consider the world constraint system $W = \{x : A \,||\, l, \exists R.C = \top \,||\, l\}$. Since x is relevant for l, the $\rightarrow_=$ rule adds $x : \exists R.C \,||\, l$. Now, the usual propagation rule $\rightarrow_\exists$ that treats exists restrictions would add $xRy \,||\, l$ and $y : C \,||\, l$ to W, where y is a new object. However, y is again relevant for l, and thus we must add $y : \exists R.C \,||\, l$. The $\rightarrow_\exists$ rule would thus be applicable to $y : \exists R.C \,||\, l$, generating new world constraints $yRz \,||\, l$ and $z : C \,||\, l$, etc.

In order to avoid such infinite chains of rule application, we introduce the notion of blocked objects.[2] Intuitively, an object x is blocked w.r.t. label l if we need not introduce a new object in order to be sure that the exists restrictions on x can be satisfied. Consider, for instance, the world constraint system $W = \{x : \exists R.C \,||\, l, x : D \,||\, l, xRy \,||\, l, y : \exists R.C \,||\, l\}$. In this case, it is sufficient to apply the $\rightarrow_\exists$ rule just to x. In fact, since all constraints for y are also constraints for x, any contradiction that could be obtained by applying this propagation rule to y can already be obtained by applying it to x. The idea is thus to say that y is blocked by x with respect to a label l if $\{C \mid y : C \,||\, l \in W\} \subseteq \{D \mid x : D \,||\, l \in W\}$. In the above example, y would thus be blocked by x, and the $\rightarrow_\exists$ rule would only be applied to x. In general, this notion of blocking is too strong, though. In fact, consider the system W' that is obtained from W by deleting the constraint $x : D \,||\, l$. In this system, x would be blocked by y and vice versa. Such cyclic blocking is clearly not appropriate since contradictions that are possibly hidden in C would never be detected.

In order to avoid cyclic blocking, we assume that the (countably infinite) set of all object names is given by an enumeration $y_1, y_2, y_3, \ldots$ We write $y < x$ if y comes before x in this enumeration. This ordering is used as follows. Whenever a new object y is introduced by applying the $\rightarrow_\exists$ rule to a world constraint system W, y is chosen such that all objects in W are smaller than y w.r.t. this ordering. In addition, only smaller objects can block a given object.

Definition 4.1 *An object y is* blocked *by an object x w.r.t. label l in a world constraint system W iff $\{C \mid y : C \,||\, l \in W\} \subseteq \{D \mid x : D \,||\, l \in W\}$ and $x < y$.*

Now, the $\rightarrow_\exists$ rule is applicable to a world constraint $x : \exists R.C \,||\, l$ in a world constraint system W only if x is not blocked by some object z w.r.t. l in W. A formal description of the propagation rules is given in Figure 1. Given a set $\{F_1, \ldots, F_n\}$ of $\mathcal{ALC}_{\mathcal{M}}$-formulas the *$\mathcal{ALC}_{\mathcal{M}}$-satisfiability algorithm* proceeds as follows. Starting with the world constraint system W_0 that is induced by $\{F_1, \ldots, F_n\}$, propagation rules are applied as long as possible.

The transformation rules are *sound* in the sense that, if W is a satisfiable world constraint system, each applicable propagation rule can be applied in such a way that the obtained derived system is satisfiable (see [Baader and Laux,1994] for a proof). For the "don't-know" nondeterministic $\rightarrow_\sqcup$ rule there are two alternative successor systems, and soundness means that one of them is satisfiable if the original system is satisfiable.[3] For the other rules (which are deterministic), soundness just means that application of the rule transforms a satisfiable system into a new satisfiable system. Furthermore, given an arbitrary induced world constraint system W_0, only a finite number of propagation rules can successively be applied, starting with W_0 (see also [Baader and Laux,1994] for a proof). This *termination property* means that, after a finite number of propagation rule applications to

[2] This idea was already used in [Buchheit *et al.*,1993; Baader *et al.*,1994], with slightly differing definitions of blocked objects.

[3] Note that the choice of an applicable rule is "don't-care" non-deterministic, i.e., we need not try different orders of rule applications.

$W \to_{\Diamond} \{l \bowtie_o l', \varphi' \,||\, l'\} \cup W$

if $\varphi \,||\, l$ is in W, where φ is $\langle o \rangle\, F$ (resp. $x : \langle o \rangle\, C$), φ' is F (resp. $x : C$), there is no label l'' in W such that the world constraints $l \bowtie_o l''$ and $\varphi' \,||\, l''$ are in W, and l' is a new label.

$W \to_{\Box} \{\varphi' \,||\, l'\} \cup W$

if $\varphi \,||\, l$ and $l \bowtie_o l'$ are in W, where φ is $[o]\, F$ (resp. $x : [o]\, C$), φ' is F (resp. $x : C$), and $\varphi' \,||\, l'$ is not in W.

$W \to_{\sqcap} \{x : C_1 \,||\, l, x : C_2 \,||\, l\} \cup W$

if $x : C_1 \sqcap C_2 \,||\, l$ is in W and W does not contain both world constraints $x : C_1 \,||\, l$ and $x : C_2 \,||\, l$.

$W \to_{\sqcup} \{x : D \,||\, l\} \cup W$

if $x : C_1 \sqcup C_2 \,||\, l$ is in W, neither $x : C_1 \,||\, l$ nor $x : C_2 \,||\, l$ is in W, and D is either C_1 or C_2.

$W \to_{\exists} \{xRy \,||\, l, y : C \,||\, l\} \cup W$

if $x : \exists R.C \,||\, l$ is in W, x is not blocked in W, and y is a new object such that $y > z$ for all objects z occurring in W.

$W \to_{\forall} \{y : C \,||\, l\} \cup W$

if $x : \forall R.C \,||\, l$ and $xRy \,||\, l$ are in W and W does not contain the world constraint $y : C \,||\, l$.

$W \to_{=} \{x : C \,||\, l\} \cup W$

if x is relevant for l, $C = \top \,||\, l$ is in W, and $x : C \,||\, l$ is not in W.

Figure 1: Propagation rules of the $\mathcal{ALC}_{\mathcal{M}}$-satisfiability algorithm.

W_0 we obtain a complete world constraint system (i.e., a system to which no more rules apply), say W'. If W' is satisfiable we can conclude that W_0 is satisfiable (since W_0 is a subset of W'). Otherwise, if W' is unsatisfiable, we can possibly derive another complete world constraint system from W_0 by another choice for the nondeterministic $\to_{\sqcup}$ rule. If all the (finitely many) choices lead to an unsatisfiable complete system then soundness of the rules implies that the original system W_0 was unsatisfiable.

Thus, it remains to be explained how satisfiability of a complete world constraint system can be decided. For this purpose, we say that a world constraint system W contains an *obvious contradiction* (or *clash* for short) if it contains either a pair of labeled $\mathcal{ALC}_{\mathcal{M}}$-formulas of the form $x : A \,||\, l$ and $x : \neg A \,||\, l$ or a labeled $\mathcal{ALC}_{\mathcal{M}}$-formula $x : \bot \,||\, l$ (for some object x, concept name A, and label l). Obviously, a world constraint system containing a clash is unsatisfiable. On the other hand, if a system is clash-free and complete then it is satisfiable (see [Baader and Laux,1994] for a proof of this property, which shows *completeness* of the propagation rules). Summing up, we obtain the following theorem.

Theorem 4.2 *Satisfiability of a finite set of $\mathcal{ALC}_{\mathcal{M}}$-formulas is decidable if we assume increasing domains.*

5 Conclusion

The framework for integrating modal operators into terminological knowledge representation languages presented in this paper should be seen as the starting point for developing more elaborate hybrid languages of this type. Extensions in at least two directions will be necessary.

First, for the adequate representation of notions like belief and time, the basic modal logic K is not sufficient. Instead, one must consider modalities that satisfy appropriate modal axioms. Second, the multi-dimensionality of our language has not really been made use of. In fact, it is easy to see that with respect to satisfiability there is no difference between the ν-dimensional and the corresponding 1-dimensional case (see [Baader and Laux,1994] for details). We have introduced a multi-dimensional framework since it is more flexible. In an extended language, different dimensions could satisfy different modal axioms (e.g., KD45 in the belief dimension, and at least S4 in the time dimension).[4] In addition, one might want to specify certain interactions between different dimensions such as independence of one dimension from certain other dimensions.

The reason for considering a simplified framework without any of these extensions in the present paper is that in this context it is possible to design a rather intuitive calculus for satisfiability. Also, the proof of soundness, termination and completeness of this calculus is still relatively short and comprehensible. For this reason, we claim that this calculus can serve as a basis for satisfiability algorithms for more complex languages. Another topic of future research will be investigating the constant domain assumption and its algorithmic ramifications. The answer to the question whether constant domain assumption or increasing domain assumption is more appropriate from the semantic point

[4] In the propositional case, the combination of different modal logics obtained this way corresponds to what Gabbay calls "dove-tailing of propositional modal logics" [Gabbay,1994].

of view strongly depends on the intended interpretation of the modalities (belief, knowledge, time, etc.).

Acknowledgment: An extended version of this article will appear in the proceedings of the conference IJCAI'95. This work has been supported by the German Ministry for Research and Technology (BMFT) under research contract ITW 8903 0, and the EC Working Group CCL, EP6028.

References

[Baader and Hollunder, 1991] F. Baader and B. Hollunder. A terminological knowledge representation system with complete inference algorithms. In *Proc. of PDK'91*, 1991.

[Baader and Laux, 1994] F. Baader and A. Laux. Terminological logics with modal operators. Research Report RR-94-33, DFKI Saarbrücken, 1994.

[Baader *et al.*, 1994] F. Baader, M. Buchheit, and B. Hollunder. Cardinality restrictions on concepts. In *Proc. of KI'94*, 1994.

[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.

[Buchheit *et al.*, 1993] M. Buchheit, F. M. Donini, and A. Schaerf. Decidable reasoning in terminological knowledge representation systems. In *Proc. of IJCAI'93*, 1993.

[Gabbay, 1994] D. M. Gabbay. LDS – Labelled Deductive Systems, Volume 1 — Foundations. Technical Report MPI-I-94-223, Max-Planck-Institut für Informatik, Saarbrücken, 1994.

[Kobsa, 1989] A. Kobsa. The SB-ONE knowledge representation workbench. In *Preprints of the Workshop on Formal Aspects of Semantic Networks*, 1989.

[Laux, 1994a] A. Laux. Beliefs in multi-agent worlds: A terminological logics approach. In *Proc. of ECAI'94*, 1994.

[Laux, 1994b] A. Laux. Integrating a modal logic of knowledge into terminological logics. In *Proc. of IBERAMIA'94*, 1994.

[Ohlbach and Baader, 1993] H.-J. Ohlbach and F. Baader A multi-dimensional terminological knowledge representation language. In *Proc. of IJCAI'93*, 1993.

[Saffiotti and Sebastiani, 1988] A. Saffiotti and F. Sebastiani. Dialogue modelling in M-KRYPTON, a hybrid language for multiple believers. In *Proc. of IEEE*, 1988.

[Schild, 1991] K. Schild. A correspondence theory for terminological logics: Preliminary Report. In *Proc. of IJCAI'91*, 1991.

[Schild, 1993] K. Schild. Combining terminological logics with tense logic. In *Proc. of EPIA'93*, 1993.

[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*. 1992.

Epistemic $\mathcal{ALC}$-knowledge bases

Francesco M. Donini, Daniele Nardi, Riccardo Rosati*
Dipartimento di Informatica e Sistemistica,
Università di Roma "La Sapienza",
Via Salaria 113, I-00198 Roma, Italy,
email:{donini,nardi,rosati}@dis.uniroma1.it

Statement of interest
We consider the epistemic concept language proposed in [5], which has been used to formalize procedural rules and weak definitions using a very restricted form of epistemic sentences. Our statement is that there are a number of other uses of the epistemic operator that require different forms of epistemic sentences in the knowledge base. To justify this statement we briefly recall the epistemic language $\mathcal{ALCK}$, then address the formalization of defaults, and provide examples of assertions including epistemic operators in the construct for universal quantification on roles and in the scope of existential quantification on roles.

1 Introduction

Concept languages (also called terminological logics, description languages) have been studied in the past years to provide a formal characterization of frame-based system. However, while the fragment of first-order logic which characterizes the most popular constructs of these languages has been clearly identified (see for example [25]), there is not yet consensus on the practical aspects of frame systems that cannot be formalized in a first-order setting. In fact, frame-based systems, as well as systems based on concept languages [3; 13], admit both forms of non-monotonic reasoning, such as defaults and closed world reasoning, and procedural features, e.g. rules. These issues have been addressed in the recent literature (see for example [1; 5; 16; 17; 22]), but the proposals typically capture one of the above mentioned aspects. In this paper we propose a concept language with an epistemic operator interpreted in terms of minimal knowledge [8; 6; 11], and we show that it provides an adequate treatment of the following features: defaults, procedural rules, weak definitions and role closures, as they are provided in the most popular systems based on concept languages. The intuitive reason that makes our approach suitable for a precise characterization of the behaviour of implemented systems is that the minimization of knowledge on possible world structures carries the idea of restricting reasoning on individuals that are known to the knowledge base (i.e. individuals that have an explicit name). In fact, most implementations are object centered, which enables them to perform efficient reasoning on the properties of individuals. However, the resulting behaviour is not only justified by implementation considerations, but carries an intuitive and natural restriction of the reasoning that the system is required to perform. Since in our setting both forms of reasoning are allowed, the expressivity of the formalism is substantially enriched, giving the knowledge base designer the option to choose between standard logical reasoning and reasoning which is restricted to the individuals known to the system. We provide an example of this enriched expressivity by extending the idea of closure to concepts.

The work extends the approach taken in [5] in two respects. From the point of view of the formalism we consider a family of logics, i.e. the ground nonmonotonic modal logics, that is parameterized with respect to the possible world structures used to interpret the modal operator. From the point of view of the language we are able to address defaults, and to show that they can not be adequalely represented in the framework proposed in [5].

The paper is organized as follows. We briefly recall the epistemic language $\mathcal{ALCK}$, then discuss different interpretations of the modal operator and address the representation of defaults in the epistemic concept language. We finally address other uses of the epistemic operator both capturing the behaviour of the most popular knowledge representation systems based on concept languages, and proposing new features.

2 An Epistemic Concept Language

In this section we briefly introduce an epistemic concept language, which is an extension of the concept language $\mathcal{ALC}$ with an epistemic operator. Generally speaking, we use $\mathbf{K}C$ to denote the set of individuals *known* to be instances of the concept C in every model for the knowledge base. The syntax of $\mathcal{ALCK}$ admits the epistemic operator before any concept and role expression of the language $\mathcal{ALC}$. The semantics of $\mathcal{ALCK}$ is an adaptation to the framework

*This paper has been submitted for presentation at the Fourth Congress of the Italian Association for Artificial Intelligence, AI*IA-95.

of concept languages of the one proposed in [10; 11; 19], which is based on the Common Domain Assumption (i.e. every interpretation is defined over a fixed domain, called Δ) and on the Rigid Term Assumption (i.e. for every interpretation the mapping from the individuals into the domain elements, called γ, is fixed).

The syntactic definition of $\mathcal{ALCK}$ is as follows (C, D denote concepts, R denotes a role, A denotes a primitive concept and P a primitive role):

$$
\begin{array}{rcll}
C, D & \longrightarrow & A \mid & \text{(primitive concept)} \\
 & & \top \mid & \text{(top)} \\
 & & \bot \mid & \text{(bottom)} \\
 & & C \sqcap D \mid & \text{(intersection)} \\
 & & C \sqcup D \mid & \text{(union)} \\
 & & \neg C \mid & \text{(complement)} \\
 & & \forall R.C \mid & \text{(universal quantification)} \\
 & & \exists R.C \mid & \text{(existential quantification)} \\
 & & \mathbf{K}C & \text{(epistemic concept)} \\
R & \longrightarrow & P \mid & \text{(primitive role)} \\
 & & \mathbf{K}P & \text{(epistemic role)}.
\end{array}
$$

An *epistemic interpretation* is a triple $(\mathcal{I}, \mathcal{W}, \mathcal{R})$ where $\mathcal{I}$ is a possible-world, $\mathcal{W}$ a set of possible-worlds, and $\mathcal{R}$ is a binary relation on $\mathcal{W}$, such that the following equations are satisfied:

$$
\begin{array}{rcl}
\top^{\mathcal{I},\mathcal{W},\mathcal{R}} & = & \Delta \\
\bot^{\mathcal{I},\mathcal{W},\mathcal{R}} & = & \emptyset \\
A^{\mathcal{I},\mathcal{W},\mathcal{R}} & = & A^{\mathcal{I}} \subseteq \Delta \\
P^{\mathcal{I},\mathcal{W},\mathcal{R}} & = & P^{\mathcal{I}} \subseteq \Delta \times \Delta \\
(C \sqcap D)^{\mathcal{I},\mathcal{W},\mathcal{R}} & = & C^{\mathcal{I},\mathcal{W},\mathcal{R}} \cap D^{\mathcal{I},\mathcal{W},\mathcal{R}} \\
(C \sqcup D)^{\mathcal{I},\mathcal{W},\mathcal{R}} & = & C^{\mathcal{I},\mathcal{W},\mathcal{R}} \cup D^{\mathcal{I},\mathcal{W},\mathcal{R}} \\
(\neg C)^{\mathcal{I},\mathcal{W},\mathcal{R}} & = & \Delta \setminus C^{\mathcal{I},\mathcal{W},\mathcal{R}} \\
(\forall R.C)^{\mathcal{I},\mathcal{W},\mathcal{R}} & = & \{d_1 \in \Delta \mid \forall d_2. \\
 & & (d_1, d_2) \in R^{\mathcal{I},\mathcal{W},\mathcal{R}} \rightarrow d_2 \in C^{\mathcal{I},\mathcal{W},\mathcal{R}}\} \\
(\exists R.C)^{\mathcal{I},\mathcal{W},\mathcal{R}} & = & \{d_1 \in \Delta \mid \exists d_2. \\
 & & (d_1, d_2) \in R^{\mathcal{I},\mathcal{W},\mathcal{R}} \wedge \ d_2 \in C^{\mathcal{I},\mathcal{W},\mathcal{R}}\} \\
(\mathbf{K}C)^{\mathcal{I},\mathcal{W},\mathcal{R}} & = & \bigcap_{(\mathcal{I},\mathcal{J}) \in \mathcal{R}} (C^{\mathcal{J},\mathcal{W},\mathcal{R}}) \\
(\mathbf{K}P)^{\mathcal{I},\mathcal{W},\mathcal{R}} & = & \bigcap_{(\mathcal{I},\mathcal{J}) \in \mathcal{R}} (P^{\mathcal{J},\mathcal{W},\mathcal{R}}).
\end{array}
$$

An $\mathcal{ALCK}$-knowledge base Ψ is a pair $\Psi = \langle \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{T}$, called the *TBox*, is a set of inclusion statements of the form $C \sqsubseteq D$, with $C, D \in \mathcal{ALCK}$, and $\mathcal{A}$ (the *ABox*) of is a set of membership assertions, of one of the forms $C(a), R(a, b)$ with $C, R \in \mathcal{ALCK}$ and a, b are names of individuals. The truth of inclusion statements and membership assertions in an epistemic interpretation is defined as set inclusion and set membership, respectively. The general form of the semantics is based on a preference criterion on epistemic interpretations; a full definition is given in [15]. Here we recall the special case where $\mathcal{R} = \mathcal{W} \times \mathcal{W}$, corresponding to consider the so-called universal S5 models, considered in [5]. In such a case an *epistemic model* for Ψ is a triple $(\mathcal{I}, \mathcal{W}, \mathcal{W} \times \mathcal{W})$, where $\mathcal{I} \in \mathcal{W}$ and $\mathcal{W}$ is any maximal set of worlds such that for each $\mathcal{J} \in \mathcal{W}$, every sentence (inclusion or membership assertion) of Ψ is true in $(\mathcal{J}, \mathcal{W}, \mathcal{W} \times \mathcal{W})$.

An $\mathcal{ALCK}$-knowledge base Ψ is said to be *satisfiable* if there exists an epistemic model for Ψ, *unsatisfiable* otherwise. Ψ logically implies an assertion σ, written $\Psi \models \sigma$, if σ is true in every epistemic model for Ψ.

3 Defaults as epistemic sentences

In this section we discuss whether defaults are representable as epistemic sentences in $\mathcal{ALCK}$. We start by observing that the interpretation of the modal operator in $\mathcal{ALCK}$ corresponds to the first-order extension of the propositional *ground* nonmonotonic logic $S5_G$. We will show that this particular logic does not admit any modular translation for default theories, while such a translation is possible in the case of ground logics built from modal systems different from $S5$.

We first briefly recall propositional ground modal logics [20; 23; 9; 6] and default logic [18], then show some properties of this family of logics with respect to their ability to capture rules and defaults, and finally turn our attention to concept languages by defining a modal concept language which admits a modular translation for default theories.

Given a normal modal logic $\mathcal{S}$, a consistent set of formulas T is an $\mathcal{S}_G$-*expansion* for a set $I \subseteq \mathcal{L}_K$ if

$$
T = Cn_{\mathcal{S}}(I \cup \{\neg K\varphi \mid \varphi \in \mathcal{L} \setminus T\}). \tag{1}
$$

where $Cn_{\mathcal{S}}$ is the consequence operator of modal logic $\mathcal{S}$, $\mathcal{L}$ is the propositional language and $\mathcal{L}_K$ is the propositional language augmented by the modal operator K. The resulting (nonmonotonic) consequence operator is defined as the intersection of all $\mathcal{S}_G$-expansions for I. Therefore, for every normal modal logic $\mathcal{S}$, the corresponding *ground* nonmonotonic modal logic $\mathcal{S}_G$ is obtained by means of the above fixpoint equation.

We recall that a propositional default theory is a pair (D, W), such that D is a set of defaults, i.e. inference rules of the form $\frac{\alpha : M\beta_1, \ldots, M\beta_n}{\gamma}$, where $\alpha, \beta_i, \gamma \in \mathcal{L}$, W is a theory in $\mathcal{L}$ and $M\beta$ is interpreted as: "it is consistent to assume β". A *justification-free default* is a default where the justification part is empty, i.e. of the form $\frac{\alpha :}{\gamma}$. As we shall see later, this kind of defaults correspond to trigger rules, whose formalization in the language $\mathcal{ALCK}$ has been studied in [5].

We are interested in the translation of a default theory into a modal theory. Therefore we give the following definitions (taken from [7]).

Definition 3.1 *A* faithful *translation from default logic to a ground nonmonotonic logic $\mathcal{S}_G$ is a mapping tr which transforms each default theory $\mathcal{D}$ into a modal theory $tr(\mathcal{D})$ such that the objective parts of the $\mathcal{S}_G$-expansions of $tr(\mathcal{D})$ are exactly the default extensions of $\mathcal{D}$.*

As pointed out by Gottlob [7], not every translation that is faithful is useful in practice. In particular, we would like to be able to turn each default into a modal sentence, independently of other defaults. Such translations are called modular.

Definition 3.2 *A translation tr from default logic to a ground nonmonotonic logic $\mathcal{S}$ is* modular *iff for each default set D and each $W \subseteq \mathcal{L}$ it holds that $tr(D, W) = tr(D, \emptyset) \cup W$.*

We shall use the modular translation *emb* introduced in [24].

$$emb(d) = K\alpha \wedge K\neg K\neg\beta_1 \wedge \ldots \wedge K\neg K\neg\beta_n \supset \gamma$$

$$emb(D, W) = W \cup \{emb(d) \mid d \in D\}$$

where d is a default. We now show some properties of ground logics with respect to their ability to capture defaults.

A first interesting result concerns the existence of modular translations for *justification-free* defaults. In particular, we have that $emb(D, W)$ provides the desired result for *any* ground nonmonotonic logic.

Theorem 3.3 *There exists a faithful modular translation from justification-free default theories to any ground nonmonotonic logic.*

Proof. Let (D, W) be a default theory such that D is a collection of justification-free defaults. Then (D, W) has exactly one default extension S. The theory $emb(D, W)$ has $ST(S)$, i.e. the unique stable theory T such that $T \cap \mathcal{L} = Cn(S)$, where Cn is the propositional consequence operator, as its only K_G-expansion[1] (see Theorem 3.5 below). Moreover, it can be shown that every $S5$-model $\mathcal{M}$ for $emb(D, W)$ is such that $Th(\mathcal{M}) \cap \mathcal{L} \subseteq S$, which implies that $ST(S)$ is the only $S5_G$-expansion for $emb(D, W)$. Thus, for every logic $\mathcal{S}$ such that $K \subseteq \mathcal{S} \subseteq S5$, theory $emb(D, W)$ admits exactly one $\mathcal{S}_G$-expansion $ST(S)$. Therefore for such logics *emb* is a faithful translation for justification-free defaults. □

With respect to defaults we start with a negative result on the logic $S5_G$. Notice that this result also holds for logic $\mathcal{ALCK}$, which corresponds to a first-order extension of $S5_G$.

Theorem 3.4 *There exists no faithful modular translation from default logic to $S5_G$.*

Proof. Consider the default theory (D, W_0) such that $D = \{\frac{a}{a}\}$, $W_0 = \emptyset$, and suppose tr is a faithful modular translation from DL to $S5_G$. This implies that $tr(D, W_0)$ has only one $S5_G$-expansion $T = ST(a)$. Therefore, in every (monotonic) $S5$-model $\mathcal{M}$ for $tr(D, W_0)$ it holds that $\mathcal{M} \models a$. Now, given $W_1 = \{\neg a\}$, by the hypothesis of modularity of tr it follows that $tr(D, W_1) = tr(D, W_0) \cup W_1$. Consequently, $tr(D, W_1)$ is an $S5$-inconsistent theory, and hence it has no $S5_G$-expansions, while on the other hand the default theory (D, W_1) has the default extension $Cn(\{\neg a\})$, thus contradicting the hypothesis of faithfulness of tr. □

The impossibility of a faithful modular translation from default theories to $S5_G$ originates from the monotonicity of this particular logic with respect to objective formulae, in the sense that for each $I, \varphi \in \mathcal{L}_K$ and for each $\psi \in \mathcal{L}$, if $I \models_{S5_G} \psi$ then $I \cup \varphi \models_{S5_G} \psi$. Therefore in $S5_G$ only modal formulae can change their validity when new information is added. Since no other ground logic shares this characteristic with $S5_G$, this negative behaviour seems to be restricted to the logic $S5_G$ only. On the other hand, for a wide class of ground logics we obtain the following positive result (which is analogous to that obtained for McDermott and Doyle's logics in [24]).

Theorem 3.5 *Given a modal logic $\mathcal{S}$ such that $K \subseteq \mathcal{S} \subseteq S4F$, there exists a faithful modular translation from default logic to the ground nonmonotonic logic $\mathcal{S}_G$.*

Sketch of the proof. (For the detailed proof see [15]) First, it is shown that if S is a default extension for the default theory (D, W), then $ST(S)$ is an $\mathcal{S}_G$-expansion for $emb(D, W)$ for any modal logic $\mathcal{S}$ such that $K \subseteq \mathcal{S} \subseteq S5$. Then, we prove that if $ST(S)$ is a $S4F_G$-expansion for $emb(D, W)$, then S is a default extension for (D, W). This is obtained by exploiting a correspondence between minimal expansions in McDermott and Doyle logics and ground expansions. The two above results imply that, for each modal logic $\mathcal{S}$ such that $K \subseteq \mathcal{S} \subseteq S4F$, *emb* is a faithful translation for $\mathcal{S}$; since *emb* is modular, this concludes the proof. □

We now turn our attention to the problem of expressing defaults in concept languages. We denote the family of concept languages $\mathcal{ALCK}(\mathcal{S}_G)$ as the epistemic concept language where the modal operator is interpreted as in ground propositional logic $\mathcal{S}_G$, by choosing the properties of the accessibility relation. In the setting of concept languages we have to deal with the issue of giving a semantics to open defaults, i.e. defaults in which free variables occur. The semantics of open defaults is still debated [12; 2]. We assume the semantics proposed by Baader and Hollunder [1], which restricts the application of defaults only to the individuals explicitly mentioned in the ABox. Notice that this semantics can be viewed as the natural extension of the semantics of rules given in [5], where rules apply only to the known individuals in the knowledge base.

The following theorem shows that, under this semantics, default theories are expressible in a large subset of the family $\mathcal{ALCK}(\mathcal{S}_G)$.

Theorem 3.6 *Given a modal logic $\mathcal{S}$ such that $K \subseteq \mathcal{S} \subseteq S4F$, there exists a faithful modular translation from default logic to the concept language $\mathcal{ALCK}(\mathcal{S}_G)$.*

Proof. We show that the following modular translation τ is faithful:

[1] K_G denotes the ground logic obtained from the modal logic K, not to be confused with the symbol used for the epistemic operator.

$$d = \frac{\alpha : M\beta_1 \ldots M\beta_n}{\gamma}$$

$(2)\tau(d) = K(O \wedge \alpha) \wedge K\neg K\neg\beta_1 \wedge \ldots \wedge K\neg K\neg\beta_n \supset \gamma$

where O is a concept whose extension comprises all the individuals explicitly mentioned in the ABox, namely we assume that for each individual a there exists an assertion $O(a)$ in the ABox. We can see formula $K(O \wedge \alpha) \wedge K\neg K\neg\beta_1 \wedge \ldots \wedge K\neg K\neg\beta_n \supset \gamma$ as the set of its instances on all individuals (both known and unknown), i.e. as two sets of closed formulae, one on known individuals and the other one on unknown individuals, which we call respectively KD and UD. These formulae correspond to propositional defaults according to the translation emb. Now, Theorem 3.5 guarantees that the set of formulae KD exactly corresponds to the instances of the default rule on known individuals. On the other hand, none of the propositional defaults corresponding to the set UD can be applied, because of the presence of the conjunct O in the prerequisite: consequently, τ is a faithful translation for default theories under the semantics of Baader and Hollunder. □

Notice that the formalization of procedural rules given by [5], is the same obtained by applying emb to justification-free defaults. It can be shown that, with respect to $\mathcal{ALCK}(S5_G)$, the modification of the semantics for the modal operator does not affect the behaviour of knowledge bases which employ the epistemic operator only in the formalization of weak definitions and rules as introduced in [5]. Now we consider logic $\mathcal{ALCK}(S4F_G)$. The modified semantics does not change the interpretation of epistemic queries (this is a result valid for each concept language $\mathcal{ALCK}(\mathcal{S}_G)$). Therefore we can extend to $\mathcal{ALCK}(S4F_G)$ the properties shown in [5] for $\mathcal{ALCK}(S5_G)$, in particular the ability to express forms procedural rules. Moreover, the last theorem states that $\mathcal{ALCK} - S4F_G$ allows for the formalization of defaults.

We can define a decision procedure for $\mathcal{ALCK}(S4F_G)$-knowledge bases containing defaults, and allowing epistemic queries. Such a procedure employs the calculus for $\mathcal{ALCK}(S5_G)$ defined in [4] and separately treats defaults, for example using the method of Schwind and Risch [21] on the instances of the defaults on all the known individuals. Consequently, instance checking for this kind of $\mathcal{ALCK}(S4F_G)$-knowledge bases is a decidable problem.

4 Epistemic knowledge bases

The use of the modal operator to represent features offered by current systems based on concept languages has been addressed in [5], where two settings are proposed:

1. the knowledge base is constituted by a set of non-epistemic assertions and the epistemic operator is allowed in the query language only. In this setting it has been shown that one can express a form of closed-world reasoning, as well as integrity constraints in the style of [19];
2. the knowledge base is constituted by a pair $(\mathcal{R}, \mathcal{A})$, where $\mathcal{A}$ is a set of non-epistemic assertions as before and $\mathcal{R}$ is a set of epistemic assertions of the form $\mathbf{K}C \sqsubseteq D$, where C, D are $\mathcal{ALC}$-concepts. In this setting it is possible to formalize procedural rules in the style of CLASSIC [3], and weak definitions that arise by treating the definition $A \doteq D$ as the two weaker inclusions $\{\mathbf{K}A \sqsubseteq D, \mathbf{K}D \sqsubseteq A\}$.

In this section we consider knowledge bases with more general forms of epistemic sentences. First of all, as we have shown in the previous section, defaults can be expressed as a particular class of epistemic sentences 3, interpreted by refining the semantics of the modal operator.

In the following we first show that role closure can be nicely formalized in our setting by using epistemic roles, then we discuss a new form of closure obtained by using epistemic concept expressions.

Closure on roles is available both in CLASSIC(see [3]) and in LOOM (see [13]). The idea is to restrict universal role quantifications to the known individuals filling the role in the knowledge base. The following example has been developed in the system CLASSIC. Let us consider the following knowledge base:

$$\texttt{vegetarian} \doteq \forall\texttt{EATS.plant}$$
$$\texttt{EATS(Bob, Celery)}$$
$$\texttt{plant(Celery)}$$

If one applies the closure operator to the role `EATS`, CLASSIC infers `vegetarian(Bob)`. We can formalize the above sequence of operations by introducing the epistemic operator in the knowledge base. In particular, the definition $\texttt{vegetarian} \doteq \forall\texttt{EATS.plant}$ plus the closure of `EATS` could be represented by $\texttt{vegetarian} \doteq \forall\mathbf{K}\texttt{EATS.plant}$. The difference with CLASSIC is that a subsequent assertion `EATS(Bob,Meat)` would cause a system warning in CLASSIC, while in our epistemic setting would simply mean that `vegetarian(Bob)` no longer holds.

Another interesting example points out the procedural nature of the closure operator. Let us consider the following knowledge base, which includes two rules expressed as epistemic sentences, and the closure operator is again expressed in epistemic terms:

$\mathbf{K}(\forall\mathbf{K}R.A) \sqsubseteq (FILLS\, Q\, c)$
$\mathbf{K}(\forall\mathbf{K}Q.B) \sqsubseteq (FILLS\, R\, c)$
$R(a,b), Q(a,b), A(b), B(b)$

This knowledge base admits two epistemic models, one in which $Q(a,c)$ holds by effect of the first rule, and another one in which $R(a,c)$ holds because the second rule is applied. The situation is reproducible in CLASSIC, which selects one extension or the other, based on the order in which the closure operations are executed. In other words, if R is closed first, than the result of the application of the first rule inhibits the application of the second one and vice versa.

Finally, we turn our attention to other uses of the epistemic operator that become possible in our setting. In

particular, we can state complete knowledge about a concept. Consider for example a university, whose professors have offices inside buldings. Let Σ be the following knowledge base:

university(ULS)	building(MainBld)
professor(George)	has-office(George, room1)
professor(Bob)	has-office(Bob, room2)
is-in(room1, MainBld)	is-in(room2, MainBld)

and consider the assertion $\mathcal{A}$ = "the university has a rector, who is a professor". One may express the assertion as $\exists$has-rector.professor(ULS). Consider the question to the knowledge base "Do you know where the rector's office is?", which can be expressed as the following query Q:

$$\mathbf{K}\exists\texttt{has-rector}.\exists\texttt{has-office}.\exists\texttt{is-in}.\mathbf{K}\texttt{building}(\texttt{ULS})$$

Of course the answer is No, since the rector can be any (unknown) professor whose office we don't know about. This is captured by the fact that

$$\Sigma \cup \{\exists\texttt{has-rector.professor}(\texttt{ULS})\} \models \neg Q$$

because there is one epistemic model, where in different worlds there can be a different (known or unknown) rector, with the office in any building.

Suppose now that we know who the professors are, that is, we have complete knowledge about professors of the university. In this case, the assertion $\mathcal{A}$ should be expressed as $\exists$has-rector.**K**professor(ULS)[2]. Now we have that

$$\Sigma \cup \{\exists\texttt{has-rector.}\mathbf{K}\texttt{professor}(\texttt{ULS})\} \models Q$$

which correctly captures the fact that the knowledge base knows where the rector's office is, namely, in the main building. In fact, there are two epistemic models of $\Sigma \cup \{\exists\texttt{has-rector.}\mathbf{K}\texttt{professor}(\texttt{ULS})\}$. In one model, George is the rector, while in the other Bob is. In both models, every world contains the fact that the rector's office is in the main building.

Notice that the kind of closure on roles that we have just described is not available in implemented systems, although it seems to be both useful and natural. We expect that other uses of the epistemic operator can be found and we are currently investigating this possibility.

5 Conclusions

In this paper we have proposed a nonmonotonic modal formalism based on the idea of minimization of knowledge, that captures the intuition of restricting certain forms of reasoning to the individuals known to the knowledge base. We have shown that the formalism allows for the representation of defaults, by adopting the semantics of the modal operator given by ground non monotonic modal logics. Moreover, we have shown that a number of non-first order features that are available in knowledge representation systems based on concept languages can be nicely formalized by allowing various forms of modal statements in the knowledge bases. The proposed formalism fills in the gap between theory and practice by providing forms of reasoning that both have a natural interpretation and correspond to some of the features implemented in the systems. We plan to develop our work by investigating the reasoning methods that are more appropriate for the different classes of epistemic sentences that are included in the knowledge base.

[2] In fact, the last expression captures the assertion "the university has a rector, which is *one of* the (known) professors".

References

[1] F. Baader and R. Hollunder. Embedding defaults into terminological knowledge representation formalisms. In *Proceedings of KR-92*, pages 306–317, Morgan Kaufmann, 1992.

[2] F. Baader and K. Schlechta. A semantics for open normal defaults via a modified preferential approach. DFKI research report RR-93-13, 1993.

[3] R. J. Brachman, D. McGuinness, P. F. Patel-Schneider, L. A. Resnick. Living with CLASSIC: When and How to Use a KL-ONE-Like Language. In *Principles of Semantic Networks*, J. Sowa ed., Morgan Kaufmann, 1990.

[4] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, Werner Nutt, Andrea Schaerf. Adding epistemic operators to concept languages. In *Proceedings of KR-92*, pages 342–353, Morgan Kaufmann, 1992.

[5] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, Werner Nutt, Andrea Schaerf. Queries, Rules and Definitions as Epistemic Sentences in Concept Languages. In *Theoretical Foundations of Knowledge Representation and Reasoning*, Gerhard Lakemeyer and Bernhard Nebel eds., LNAI 810, Springer-Verlag, 1994.

[6] F. M. Donini, D. Nardi and R. Rosati. Ground Nonmonotonic Modal Logics for Knowledge Representation. To appear in *Proceedings of WOCFAI-95*.

[7] Georg Gottlob. The Power of Beliefs - or - Translating Default Logic into Standard Autoepistemic Logic. In *Proc. of the 13th Int. Joint Conf. on Artificial Intelligence IJCAI-93*, Chambery, France, 1993.

[8] J. Halpern and Y. Moses. Towards a theory of knowledge and ignorance: preliminary report. In K. Apt editor, *Logics and models of concurrent systems*, pages 459–476, Springer-Verlag, 1985.

[9] M. Kaminski. Embedding a default system into nonmonotonic logic. *Fundamenta Informaticae*, 14:345–354, 1991.

[10] Hector J. Levesque. Foundations of a functional approach to knowledge representation. *Artificial Intelligence Journal*, 23:155–212, 1984.

[11] Vladimir Lifschitz. Nonmonotonic databases and epistemic queries. In *Proc. of the 12th Int. Joint*

Conf. on Artificial Intelligence IJCAI-91, Sydney, 1991.

[12] Vladimir Lifschitz. On open defaults. In *Proc. of the Symposium on Computational Logics*, Brüssel, Belgium, 1990.

[13] Robert MacGregor. A deductive pattern matcher. In *Proc. of the 7th Nat. Conf. on Artificial Intelligence (AAAI-88)*, pages 403–408, 1988.

[14] V.W. Marek and M. Truszczyński. Nonmonotonic logic. Context-dependent reasoning. Springer-Verlag, 1993.

[15] D. Nardi and R. Rosati. Modal logics of minimal knowledge. Technical Report, Dip. Informatica e Sistemistica, Univ. "La Sapienza", Roma, 1995. Available by anonymous ftp at dis.uniroma1.it, in /pub/nardi/NaRo95.ps.

[16] Lin Padgham and Tingting Zhang. A terminological logic with defaults: a definition and an application. In *Proc. of the 13th Int. Joint Conf. on Artificial Intelligence IJCAI-93*, pages 662–668, Chambery, France, 1993.

[17] J. Quantz and V. Royer. A preference semantics for defaults in terminological logics. In *Proceedings of KR-92*, pages 294–305, Morgan Kaufmann, 1992.

[18] R. Reiter, A Logic for Default Reasoning. *Artificial Intelligence Journal*, 13:81-132, 1980.

[19] R. Reiter. On asking what a database knows. In J. W. Lloyd, editor, *Symposium on Computational Logics*, pages 96–113. Springer-Verlag, ESPRIT Basic Research Action Series, 1990.

[20] G. Schwarz. Bounding introspection in nonmonotonic logics. In *Proceedings of the 3rd international conference on principles of knowledge representation and reasoning (KR-92)*, pages 581–590, Morgan Kaufmann, 1992.

[21] C. Schwind and V. Risch. A tableau-based characterisation for default logic. In *Proceedings of the First European Conference on Symbolic and Quantitative Approaches for Uncertainty*, pages 310–317, Marseilles, France, 1991.

[22] Umberto Straccia. Default inheritance reasoning in hybrid KL-ONE-style logics. In *Proc. of the 13th Int. Joint Conf. on Artificial Intelligence IJCAI-93*, pages 676–681, Chambery, France, 1993.

[23] M. Tiomkin and M. Kaminski. Nonmonotonic default modal logics. In *Proceedings of the Third Conference on Theoretical Aspects of Reasoning about Knowledge (TARK-90)*, pages 73–84, 1990.

[24] M. Truszczyński. Modal interpretations of default logic. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence IJCAI-91*, Sydney, 1991.

[25] William A. Woods and James G. Schmolze. The KL-ONE Family. *Semantic Networks in Artificial Intelligence*. Published as a special issue of *Computers & Mathematics with Applications*, Volume 23, Number 2–9, 1992, pages 133–178.

Hierarchical Correspondance between Physical Situations and Action Models

Véronique Royer, ONERA DES/SIA
29 avenue de la division Leclerc, BP 72, F-92322 Châtillon
email: royer@onera.fr

Abstract

The framework of the paper is the recognition of ongoing actions of groups of agents in a scene watched by image sensors. In such applications, the actions are not directly observed but must be interpreted from observed physical situations. In particular, as the perception ressources are usually limited, one crucial issue is whether more informed situations correspond to more specific actions.
The physical situations are given an extensional representation and partially ordered w.r.t some information specificity. The action models are represented using Description Logics, thus organized into subsumption taxonomies. The correspondance between both representation spaces is formalized and shown to behave monotonically w.r.t. the respective taxonomic structures under specific "homogeneity" conditions over the situations. The application of this formal framework to situation recognition is then discussed.
Keywords: action modeling, taxonomic representations, situation recognition.

1 Introduction

The global framework of the paper is the recognition of ongoing actions of groups of agents (human or artificial) in a scene watched by image sensors. The need for interaction between the observation and the interpretation tasks raises the general issue of the *structural correspondance* between the representation of *situations* – physical happenings observed in the scene – and the representation of *actions* – prototypical models used for situation recognition.
The situations reasonably need a *hierarchical structure* for capturing the fact that some situations are *"more informed"* than others, which is an important criterion when evaluating or comparing several candidate interpretations. The actions usually have a *taxonomic structure*: typically, an action is said more specific than another one if the formed refines the latter [Kautz]. Therefore one more issue is whether the correspondance is *hierarchical*, that is whether *more informed situations correspond to more specific actions*. From a strategic point of view, this property is highly desirably because it garanteed some *monotonicity* of situation recognition under limited perception ressources: that the actions recognized from observed situations – usually less informed than the real ones – subsume the real intended actions.
The literature in Plan Recognition just ignores these issues because most works apply to non physical domains, like speech-act understanding or intelligent help systems [Cohen et al.]. So, the actions are assumed to be directly observed [Kautz, Weida-Litman, Artale-Franconi]. Even works like [Dousson et al.], in the context of process monitoring, assume that the observations directly instantiate action models. The correspondance issue is naturally addressed in the domain of active vision [Black et al., Howards-Buxton, Kuniyoshi-Inoue], yet not seen as a *formal mapping between two heterogeneous representation spaces*.
The paper contributes by addressing these issues in the context of the surveillance of a urban site, a parking lot. The physical situations, possible intertwined, involve movements of groups of people and vehicles like *'somebody goes out of a building towards the parking lot, goes into a car and drives away'*. We present an extensional representation for the situations, with a partial order capturing a notion of information specificity. Actions modeling movements of *groups of anonymous agents* are represented as formal terms in a Description Logic, thus organized into subsumption taxonomies. We then establish the formal correspondance between both representation spaces: two definitions are actually proposed depending on how the groups of agents are interpreted in terms of individual behaviors and targets. Under specific "homogeneity" conditions the correspondance is shown to behave monotonically w.r.t. the respective taxonomic structures. Finally, we consider the application of our formal framework to situation recognition regarding both identification and recognition problems. In particular, we shed some light on the nonmonotonicity of reasoning.

2 Taxonomic representation of physical situations

In all the paper we call *taxonomy* any set T equipped with a partial order $\prec_T$ and a unique maximal element $\top_T$. Given elements t and t' in the taxonomy $(T, \prec_T, \top_T)$, t is said *more specific* than t' iff $t \prec_T t'$.
We start by giving initial data for describing the observations, by means of a *typed environment*. The physical objects participating in situations are typed according to an *Object* taxonomy and located in a geometrical 2D referential R. We assume a typing of R's geometrical positions into a domain of topological zones, *Zone*, also organized into a taxonomy — typically, one can distinguish between foot zones and car zones, foot zones could be further specialized into building areas, etc... A third taxonomy, *Attitude*, captures the *qualitative spatial relations* between objects — typically a Move attitude can be specialized into MoveFrom, MoveTo, etc...
The perception system outputs observations about physical objects located in the scene which after suitable data fusion are collected into facts, formally triples (o,t,p), where o is an object name, t is an object type in *Object*, and p represents a discrete trajectory in R. A *situation* is thus defined as a set of facts for distinct object names. A situation S is said *more informed* than a situation S' iff S informs about more objects than S' and for any common object S gives a more specific type and a more detailed trajectory than S'.

Definition 1 *A* **typed environment** *E is given by:*
- three taxonomies: Object, Zone, Attitude, describing respectively the object types, topological zones and qualitative spatial relations of E;
- a domain of names: O ;
- a geometrical referential: R ;
- a time space: T (discrete, linear);
- a mapping zone: $R \rightarrow Zone$.
A **trajectory** *in E is an element $\{(z_1,t_1),...,(z_n,t_n)\}$ of $2^{R\times T}$ such that $n \geq 2$ and $\forall i \neq j, t_i \neq t_j$. P denotes the set of trajectories in E.*
A **situation** *in E is an element of $2^{O\times Object\times P}$ such that: if (o,t,p) ≠ (o,t',p') in S, then o ≠ o'.*
For any situation S, objects(S) = {o, (o,t,p)∈S}.
A situation S is said **more informed** *than a situation S', written $S \ll S'$, iff objects(S') ⊆ objects(S) and ∀o∈objects(S'), if (o,t,p)∈S and (o,t',p')∈S' then $t \prec_{Object} t'$ and p'⊆p.*

The relation $\ll$ defines a partial order on the space of situations in E. It actually captures a notion of "information specificity" compatible with the standards of data fusion: aggregating new observations to one situation results in a more informed situation.
Note that situations should desirably satisfy other domain dependent properties like persistency of the objects, coherence of the trajectories according to speed laws, etc..., not addressed here.

3 Representing actions

3.1 Modeling choices

Following classical approaches in Plan Recognition [Kautz, Cohen] plans are seen as collections of actions together with temporal constraints. We implicitly rely on the definition of plan specificity of [Weida-Litman].
As opposed to plans, actions will capture only *time invariant knowledge* about the situations. The reason is that image processing will be performed in a non continuous way – one image each x seconds will be processed – so that actions have to be analysed from discrete, quasi-static observations. Therefore we do not need any temporal extension of DL as opposed to [Artale-Franconi], to represent actions.
Inspiring from the modeling of movements from an observer's point of view in [Kuyinoshi-Inoue], we will distinguish between:

- the *agents* – mobile objects during the action – and the *targets* – static objects during the action,
- *motion actions*, qualitatively describing how agents are moving relative to one target, and *movement actions* qualitatively describing how agents are moving in an absolute way.

To emphasize the presence of the target, we will speak in the following of *target actions* for motions and of *free actions* for movements.
For both kinds of actions, the agents are assumed to be *anonymous*. Indeed, the surveillance system will hardly track individual objects, especially human beings. An object will be recognized — typed as 'somebody' — but may not be identified as being one particular object — 'that person'. Unlike [Borgida,Di-Eugenio], we thus do not need "same-as" DL operators to express coreferences between objects.
Without loss in generality, we also consider target actions with a *single target*. Actions at several targets, like *'somebody going from one spot to another one'*, have to be described as temporal networks of elementary actions at one single target. An action may also specify further constraints like a *location* zone. Other aspects like qualitative estimates of the durations may be added.

3.2 Description Logic based representation

Recent works have promoted Description Logics (DL) as formal languages for action representation [Devanbu-Litman, Borgida] because they can both support the definition of taxonomic domains and provide built-in classification mechanisms [Nebel]. In [Weida-Litman] the benefit of using DL lies mainly in the clear management of action classification rather than in DL's expression power. Works like [Borgida, DiEugenio] show how DL can be used to represent complex compositional actions. Figure 1, together with the appendice, gives a flavour of our DL-based action representation. The reader is referred to [Nebel] for a detailed introduction DL.
DL helps to formalize action taxonomies. An action library is represented by an *action terminology*, that is a

set of action names together with their descriptions, as shown in Figure 1.

The taxonomic structure is given by the subsumption

Role declarations:
agt :< domain(Action) and range(Object) (*action's agents*)
loc :< domain(Action) and range(Zone) (*action's locations*)
tgt :< domain(Action) and range(Object) (*action's target*)
att :< domain(Action) and range(Attitude) (*attitudes*)
Concept declarations:

Object :< Anything	Zone:< Anything
Ped(estrian):< Object	FootZ(one) :< Zone
Car :< Object	ParkZ(one) :< Zone
Attitude :< Anything	Move:< Attitude
Movement :< Attitude	Close :< Move
Slow :< Movement	MoveFrom :< Move
Still :< Movement	MoveTo:< Move

Action := atleast(1,agt,Object) and atmost(1,tgt,Object)
Tgt.Action := Action and atleast(1,tgt,Object) and all(att,Move)
Free.Action := Action and atmost(0,tgt,Object) and all(att,Movement)
Ped.Moving := Free.Action and all(agt,Ped) and all(loc,FootZ) and all(att,Slow)
Car.Parked := Free.Action and all(agt,Car) and all(att,Still) and all(loc,ParkZ)
Sth.Moving.At.Sth := Tgt-Action and atleast(1,att,Move)
Ped.Moving.At.Car := Sth.Moving.At.Sth and atleast(1,agt,Ped) and all(tgt,Car)
Ped.Moving.To.Car := Ped.Moving.At.Car and atleast(1,att,MoveTo)
Ped.Moving.From.Car := Ped.Moving.At.Car and atleast(1,att,MoveFrom)
Ped.Moving.FromTo.Car := atleast(2,agt,Ped) and Ped.Moving.To.Car and Ped.Moving.From.Car
Ped.Taking.Car := Ped.Moving.To.Car and all(loc,ParkZ) and atleast(1,att,Close)

Figure 1: Sample of an action terminology

order :< induced by the terminology on the set of action names (cf appendice). Given a typed environment E, the notion of *action terminology in E* is then introduced as any DL terminology *TA* containing:

- the four role declarations and the three concept declarations Action, Target-Action and Free-Action of Figure 1,
- three concept names Object, Attitude and Zone, such that the sets of concepts names of *TA* subsumed respectively by Object, Attitude and Zone are isomorphic[1] to the corresponding taxonomies *Object, Attitude* and *Zone* of *E*.

The *initial action terminology*, written!xdvi *A(E)*, is defined as the least action terminology in E for set inclusion of names and terminological axioms[2] For any action terminology *TA* we call *target-action* (resp. *free-action*) any concept term subsumed by the concept Tgt-Action (resp. Free-Action).

[1] Two taxonomies $(T, \prec_T, \top_T)$ and $(T', \prec_{T'}, \top_{T'})$ are isomorphic iff there is a bijective mapping $f : T \to T'$ such that $f(\top_T) = \top_{T'}$ and $t \prec_T t' \Leftrightarrow f(t) \prec_{T'} f(t')$.

[2] It corresponds to a free term algebra.

4 Interpreting situations as actions

The essence of our approach is to abstract physical situations into formal DL-expressions and to classify the resulting descriptions in the action terminology *TA*. Exactly, any physical situation is associated to a set of DL-assertions in the language of *A(E)*: assertions about an hypothetical action instance to be recognized in *TA*. Therefore, interpreting a situation w.r.t. some action terminology can be based on the instance recognition mechanisms of DL. The correspondance is proved[3] hierarchical if the abstraction process behaves monotonically w.r.t. to the respective specificity orders. Before, some interpretation choices must be discussed.

4.1 Group vs troop interpretations

There are several ways of abstracting physical situations depending on how the set of agent's attitudes of one situation is meant, whether conjunctively or disjunctively. For example, a situation like '*somebody moving towards a car and somebody else moving away from the car*' can be interpreted as:

- either one *"group action"* Ped.Moving.FromTo.Car where the attitude constraint *sums up the collection* of the individual attitudes, and the global action corresponds to a g.l.b. (for :<) of the individual actions, Ped.Moving.To.Car and Ped.Moving.From.Car.
- or one *"troop action"* Ped.Moving.At.Car where the attitude constraint means the most specific attitude *common* to all the agents, and the global action corresponds to the l.u.b. of the individual actions.

The interpretation choices depend on the application domain. For our parking lot application, we chose the group interpretation for situations corresponding to target actions, the troop interpretation for situations corresponding to free actions.

4.2 Interpreting geometrical positions

The interpretation also involves translating the geometrical object positions into qualitative estimates of the relative attitudes of the agents w.r.t. the target in the case of target actions, or the absolute directions and the speed of the agents in case of free actions. Formally, two translation mappings are presupposed:

- $RA : P \times P \to Attitude$ gives from the trajectories p and p' of two objects o and o' the most specific qualitative estimate of the *relative attitude* of o w.r.t. to o' in the *Attitude* taxonomy[4].

[3] The proofs can be found in [Castel et al.].

[4] This can be done by comparing the sum vectors induced by the discrete trajectories, given suitable thresholds. Typically, if the sum vector of one mobile object meets the gravity center of one static target, then a MoveTo attitude is recognized. One could also exploit neuronal techniques for recognizing attitudes.

- $AA : P \rightarrow Attitude$ gives from the trajectory of any object the most specific qualitative estimate of its direction and speed[5].

4.3 Correspondance with target actions

As said before, situations corresponding to target actions get a group interpretation. Technically, the set of DL assertions associated to S, *describe(S,g,x)*, describes an abstract action instance x which is linked to all the qualitative attitudes of S's agents w.r.t. to the target g.

Definition 2 *Let TA be an action terminology in E. Let A be a target action description in TA. Let S be a situation. Let g be an object name. S is said* **to match** *A* **at target** *g in TA iff:*

$\exists (g, e_g, p_g) \in S$ *such that TA, describe(S,g,x)* $\models x::A$, *where describe(S,g,x)* $= \{x::\text{tgt}:g,\ g::t_g\} \cup \{x::\text{agt}:o,\ o::e,\ x::\text{att}:p,\ p::RA(p,p_g),\ \forall (o,e,p) \in S \setminus \{(g, e_g, p_g)\} \cup \{x::\text{loc}:zi,\ zi::zone(zi),\ \forall (o,e,p) \in S,\ \forall (zi,ti) \in p\}$.

By considering the initial action terminology *A(E)*, one can define for any situation S the equivalence class of its *most specific action descriptions* at a target g (shortly, MSA). For any such representative element, say MSA(S,g), and for any action terminology *TA*, S matches A at g in *TA* iff *TA* $\models$ MSA(S,g):<A. Consequently, *recognizing a situation* w.r.t. an arbitrary action taxonomy *TA* can proceed by *classifying* its MSA in *TA* — that is searching for most specific *TA*'s actions subsuming the MSA and for less specific *TA*'s actions subsumed by the MSA.

Definition 3 *Let E be a typed environment, S be a situation in E and g an element of objects(S).*
A term A of A(E) is said a **most specific action description** *of S at g iff:*

- S matches A at g in A(E),

- for any term A' such that S matches A' at g in A(E), then A(E) $\models A :< A'$.

Proposition 1 *Let E be a typed environment, S a situation in E and g an element of objects(S).*
If A and A' are most specific action descriptions of S at g, then $A(E) \models A :< A'$ *and* $A(E) \models A' :< A$.
Let TA be an action terminology in E and A a target action in TA. Let MSA(S,g) be a most specific action description of S at g. Then, S matches A at g in TA iff: $TA \models MSA(S,g) :< A$.

4.4 Correspondance with free actions

For troop interpretation each agent of the situation must individually satisfy the attitude constraints given by the action description. This means turning to local matching conditions, *free.describe(a,x,g)* for each object a to be interpreted as an agent, as in the following definition.

Definition 4 *Let E be a typed environment. Let TA be an action terminology and A a free action in TA. Let S be a situation in E. Then S is said* **to match (freely)** *A in TA iff* $\forall s \in S$, *TA , free.describes(s,x)* $\models x::A$, *where free.describes((o,e,p),x)* $= \{x::\text{agt}:o,\ o::e,\ x::\text{att}:p,\ p::AA(p)\} \cup \{x::\text{loc}:zi,\ zi::zone(zi),\ \forall (zi,ti) \in p\}$.

Results similar to 4.3 hold for free actions. Up to DL logical equivalence, the MSA of one situation S is now obtained as the *disjunction* (lub for :<) of the individual MSA's abstracted from the sets *free.describes(s,x)*, for each element $s \in S$.
For example, the situation of 4.1 '*somebody moving towards a car and somebody else moving from the car*' gets the following MSA's (1) and (2) under group and troop interpretation respectively:
(1) atleast(2,agt,Ped) and exactly(1,tgt,Car) and atleast(1,att,MoveFrom) and atleast(1,att,MoveTo),
(2) atleast(2,agt,Ped) and exactly(1,tgt,Car) and (atleast(1,att,MoveFrom) or atleast(1,att,MoveTo)).
Classifying (1) into the action terminology of Figure 1 gives Ped.Moving.FromTo.Car as immediate subsumer, while (2) gives Ped.Moving.At.Car.
By the way, let us notice that no Closed World Assumption is made for computing MSA's. If only two agents are observed in S, like in the previous situation, then one action with the 'atleast(2,agt,Ped)' constraint is only recognized. The likely inference that the two observed agents are the only ones should be performed at plan recognition level, depending on how the alleged action relates to previous and predicted ones.

4.5 Hierarchical correspondance

It turns out that more informed situations do not generally match more specific actions. The correspondance fails to be hierarchical because of possible "discontinuities" in the trajectories of the agents, which may induce "switch" of qualitative attitudes. Instead, if the trajectories were qualitatively invariant during the observation period of the situation then the same qualitative attitudes would be recognized in any sub-situation and the situation and its sub-situations would have the same MSA's. We slightly extend the classical notion of "homogeneity" — time invariance — to a monotonicity condition of the qualitative attitudes.

Definition 5 *Let E be a typed environment. A situation S in E is said* **homogeneous** *iff:*

- $\forall (o_1,t_1,p_1) \in S, \forall (o_2,t_2,p_2) \in S, \forall q_1 \in P, \forall q_2 \in P,$ $q_1 \subseteq p_1, q_2 \subseteq p_2 \ \Rightarrow RA(p_1,p_2) \prec_{Attitude} RA(q_1,q_2)$
- $\forall (o,t,p) \in S,\ \forall q \in P,\ q \subseteq p \Rightarrow AA(p) \prec_{Attitude} AA(q)$

Proposition 2 *: Let E be a typed environment. Let H be a situation in E and H' be a homogeneous situation in E, both to be interpreted as target actions at some object* $g \in objects(H') \cap objects(H)$. *Then:*
$H' \ll H \ \Rightarrow \ A(E) \models MSA(H',g) :< MSA(H,g)$.

A similar property holds for the correspondance with free actions.

[5] This implies a suitable partitionning of the *Attitude* taxonomy

5 Situation recognition

The formal framework presented so far essentially aims at clarifying the link between actions models and their physical occurrences, situations. We now discuss computational issues in the perspective of situation recognition.

5.1 Identification

Several independent plans may occur in the scene, like simultaneous departures or arrivals of people by car. Even inside a same zone, agents may belong to independent situations. This raises the problem of partitionning the set of observations in order to identify the different *situation units* to be interpreted as actions, then fused and recognized as plans.

Some combinatorics arises if an object can play both the role of an agent and the role of a target in different actions. The problem vanishes if in the underlying action terminology the agent types are functionally dependent of the target types, and if a given object type cannot be the target type of two incompatible actions [6]

These assumptions are quite reasonable for the parking lot application, though not for action domains involving relative motions of objects of the same type, like somebody running after somebody[7].

The computation cost also differs between group and troop interpretations. Troop interpretations yield disjunctive MSA's, hence potentially lead to expensive inferences and low performance of the DL classifier[8]. To avoid this problem, we will consider that the situations units corresponding to free actions are constituted of agents having identical qualitative behaviors.

We therefore propose the following reasoning strategy:

1. Given new observations, the new situations units are constituted of maximal groups of agents with one target: each potential target object is associated with any neighbor object whose type is compatible with the agent types prescribed to that target[9].
2. The remaining objects are associated into groups of "free" agents, according to the identity of their types and their absolute attitudes and to some proximity criterion.
3. The situation units are interpreted as shown in Section 4: by computing their MSA's and classifying them in the action terminology, that is computing both their immediate subsumers and subsumees.

Step (3) allows to distinguish between *observed actions* and *prototypical actions*. The MSA's indeed correspond to observed actions as abstracted from observed situations, while the classification results characterize the prototypical actions approximating them as known from the underlying action taxonomy. So (3) is just recognizing situations w.r.t. one action taxonomy, but not yet w.r.t. plan models. Situation recognition means eventually matching the observed actions with actions nodes *of candidate plans*. The question now is how the approximation result of step (3) makes sense for plan recognition.

5.2 Recognition

[Weida-Litman] formalize the notion of action recognition w.r.t. plans as follows: one observed action A_o can match one action node A_p of some plan iff both actions are *compatible* in the underlying action taxonomy, that is have a common subsumee A_r. Intuitively, A_o is the partial observation of some real action A_r, which must refine the action A_p in order to match the intended plan. This intuition presupposes the "observation monotonicity": that the observed actions are less specific that the real actions ($TA \models A_r :< A_o$). The strategic benefit is clearly that the search space during action recognition can be circumscribed to a subsumption class.

Section 4 aimed precisely at formally justifying such an "observation monotonicity" property, starting from physical situations instead of action instances[10]. Proposition 2 (hierarchical correspondance) states the observation monotonicity, but under two assumptions: the real situation should be (1) more informed than the observed situation and (2) homogeneous.

Assumption (1) is valid as long as the decomposition into situation units needs not to be revised: either by eliminating a "false" target or by splitting a situation unit into smaller independent ones. In a case like '*somebody is passing at a car just incidentally, before going elsewhere*', the maximal association strategy of 5.1 will produce at first an *overinformed situation unit* with a "false target" that can be diagnosed only later on with new observations. The problem is that revising the former interpretation as target action into one as free action does no more yield a move up inside a taxonomic class, as target actions and free actions define disjoint classes[11]. Action recognition thus remains nonmonotonic because of the possible switch from target actions to free actions.

Assumption (2) suggests that homogeneous situations are *persistent*, thus leading to plan interpretations maximizing homogeneity while minimizing the number of actions. This assumption is quite reasonable for surveillance applications where "erratic" moves — deviances from standard trajectories or non significant intermediate actions — need not to be diagnosed.

Let us finally notice that other preference criteria could also be considered, like statistical frequencies or risk factors. This raises a further issue regarding the integration of *preference hierarchies* [Bauer,Cayrol et al.] with the taxonomic structures discussed so far.

[6] Two elements A and A' are incompatible iff they have no common subsumee (they define disjoint taxonomic classes).

[7] In that case, one should find other criteria to discriminate between targets and agents.

[8] Efficient DL systems like BACK and CLASSIC don't allow concept disjunction [Back,Classic].

[9] Actually, a target type defines both an attraction range and types of associated agents.

[10] Remind that the actions are not directly observed !

[11] atmost(0,target,Object) and atleast(1,target,Object) are incompatible constraints.

5.3 A small example

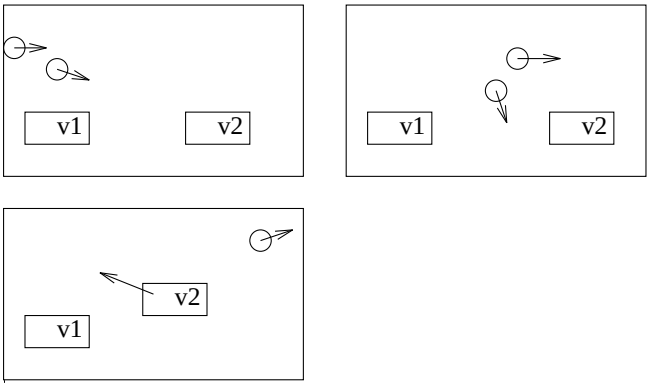


Figure 2: car departure and pedestrian traversal

Figure 2 represents three successive snapshots of a scene with two vehicles (the square boxes labelled v1 and v2) and two pedestrians (unlabelled circles). The arrows represent the direction vectors of the movements.
As shown in Figure 3 the strategy of 5.1 provides with an "objective" analysis of the scene, before exploiting the background knowledge about the plans. Note that, by

situation units	*(target, agents)*	*interpretations*
First image:		
S1	(v1, 2 pedestrians)	Ped.Moving.At.Car
S2	($\emptyset$, v2)	Car.Parked
Second image:		
S3	($\emptyset$, v1)	Car.Parked
S4	(v2, 2 pedestrians)	Ped.Moving.At.Car
Third image:		
S5	= S3	
S6	($\emptyset$, v2)	Car.Moving
S7	($\emptyset$, 1 pedestrian)	Ped.Moving
Revised situation units:		
S4a	($\emptyset$, 1 pedestrian)	Ped.Moving
S4b	(v2, 1 pedestrian)	Ped.Moving.At.Car
S1a	($\emptyset$, 1 pedestrian)	Ped.Moving
S1b	($\emptyset$, 1 pedestrian)	Ped.Moving

Figure 3: action interpretation of the scene

the maximal association strategy, S4 is chosen in the second image instead of associating one pedestrian to each car.
The next step proceeds by trying to match plan models. Whatever the reasoning for selecting candidate plans may be, it is clear that till the second image the pedestrians are likely following the same plan: going to v2 in order to leave with v2. This is contradicted by the third image as the expected action — v2 moves away and the pedestrians are no more seen — is not observed. To cope with S7[12], S4 must be revised into two smaller independent units: S4a and S4b. Then the situation composed of {S1a, S2, S4.b, S6} can match the plan Car.Departure, defined as the sequence of actions ((Ped.Moving and Car.Parked) ; Ped.Taking.Car ; Car.Moving). The action abstracted from S4.b is indeed compatible (less specific) with the plan's action Ped.Taking.Car — it corresponds to an incomplete observation of the pedestrian's motion towards the car. Two independent situations are eventually recognized: one car departure and one pedestrian traversal, the latter being reduced to the single action Ped.Moving.

[12] Assuming that no pedestrian may appear by magics.

Acknowledgement

This work is supported by the project PERCEPTION DGA-DRET France. I am grateful to M.Barat, P.Laublet, C.Castel, C.Saurel and C.Tessier for their helpful comments.

References

[Artale-Franconi] A.Artale, E.Franconi, *A Computational Account for a Description Logic of Time and Action*, KR'94, pp 3-14, 1994.
[Back] T.Hoppe, C.Kindermann, JJ.Quantz, A.Schmiedel, M.Fischer, BACK V5 Tutorial and Manual, KIT.Report 100, Technische Universität of Berlin, March 1993.
[Black et al.] J.Black et alias, *Action, Representation and Purpose : Reevaluating the foundations of Computational Vision*, panel session, IJCAI'93, vol. 2, pp 1661-66, 1993.
[Bauer] M.Bauer, *Integrating Probabilistic Reasoning into Plan Recognition*, KR'94, pp 621-24, 1994.
[Borgida] A.Borgida, *Towards a Systematic Development of Terminological Reasoners: Clasp reconstructed*, KR'92, pp 259-269, 1992.
[Castel et al.] C.Castel. V.Royer, C.Saurel. C.Tessier. Internal report available of the PERCEPTION Project, available at ONERA, Juanary 1995.
[Cayrol et al.] C.Cayrol, V.Royer, C.Saurel, *Managing Preferences in Assumption.Based Reasoning,* Information Processing and Management of Uncertainty (IPMU'92), LNCS 682, pp 13-22, 1992.
[Cohen et al.] P.R.Cohen, J.M.Morgan, M.E.Pollack, Eds, Intentions in Communication, the MIT Press, 1990.
[Classic] R.Brachman, *"Reducing" CLASSIC to Practice: Knowledge Representation Theory Meets Reality*, KR'92, pp 247-58, 1992.
[Devanbu-Litman] P.T.Devanbu, D.Litman, *Plan.based Terminological Reasoning*, KR'91, pp 128-138, 1991.
[Dousson et al.] C.Dousson, P.Gaborit, M.Ghallab, *Situation Recognition: Representation and Algorithms*, IJCAI'93, vol.1, pp 166-172, 1993.
[DiEugenio] B.DiEugenio, *Action Representation for Interpreting Purpose Clauses in Natural Language Instructions*, KR'94, pp 158-169, 1994.
[Howarth.Buxton] R.Howarth, H.Buxton, Selective Attention in Dynamic Vision, IJCAI'93, vol.2, pp 1579-84, 1993.
[Kautz] H.Kautz, *A Circumscriptive Theory of Plan Recognition*, pp 105-133 in [Cohen et al].
[Kuniyoshi-Inoue] Y.Kuniyoshi, H.Inoue, *Qualitative Recognition of Ongoing Human Action Sequences*, IJCAI'93, vol.2, pp 1600-09 , 1993.
[Nebel] B.Nebel, Reasoning and Revision in Hybrid Representation Systems, Lecture Notes in Artificial Intelligence 422, Springer Verlag 1990.
[Weida-Litman] R.Weida, D.Litman, *Terminological Reasoning with Constraint Networks and an Application to Plan Recognition*, KR'92, pp 282-293, 1992.

Closing the Terminology

Robert (Tony) Weida

IBM T. J. Watson Research Center
P.O. Box 218, Route 134
Yorktown Heights, NY 10598
weida@watson.ibm.com

Computer Science Department
Columbia University
New York, NY 10027
weida@cs.columbia.edu

1 Introduction

This paper pursues the idea of closing the set of concepts in a description logic (DL) knowledge base (KB) after the KB is developed and before problem solving begins. Essentially, our *closed-terminology assumption* (CTA) holds that all relevant concepts from the domain of interest are explicitly defined in the KB, and that every individual, once fully specified, will correspond directly to at least one concept. [1] For certain applications, this enables useful inferences which would not be possible otherwise. Consider configuration, where DL has already found practical success [Wright *et al.*,1993]. The standard *open-terminology assumption* (OTA), which presumes an incomplete terminology, is appropriate during knowledge engineering when we construct a model of systems, components, and related concepts. However, during a specific configuration task, closed-terminology reasoning may be more suitable because all relevant concepts, e.g., types of systems, are known in advance. We will show how this enhances our ability to identify concepts which an individual cannot instantiate and thus draw conclusions from the remainder. Imagine that a user incrementally specifies an individual computer system, along with its individual components, in collaboration with a configuration engine. In general, the user can make choices in any order and at any level of abstraction. We can exploit the closed taxonomy and its subsumption-based organization to (1) efficiently track the types of systems and components which are consistent with the user's current choices, (2) infer constraints on the system and components which follow from current choices and the taxonomy's specific contents, and (3) suitably restrict future configuration choices to those which are consistent with past choices. Thus, we can help focus the efforts of both the user and the configuration engine. The ideas described here evolved from my work on DL-based plan recognition in the T-REX system [Weida and Litman,1994; Weida and Litman,1992]. This work is implemented in the K-REP system [Mays *et al.*,1991]. Further details appear in [Weida,1995].

[1] We do not make a closed-world assumption over individuals.

2 Description Logic

DL provides a formal language for defining concepts and individuals [Woods and Schmolze,1992]. Key DL inferences include subsumption, classification and recognition. Concept C1 *subsumes* concept C2 when every instance of C2 is also an instance of C1. We write this as C2 $\Rightarrow$ C1. Implementations of DL maintain a concept taxonomy, or KB, where each concept subsumes its descendants and is subsumed by its ancestors. Whenever a new concept is defined, *classification* integrates it into the taxonomy. Then *recognition* determines the set of most specific concepts which an individual currently instantiates.

A *concept* is an intensional description of a class of *individuals*. Both are described in K-REP with a single language, where *roles* denote binary relationships between individuals. A role's *value restriction* is a concept which constrains the range of the relationship; only instances of the value restriction may *fill* the role. Cyclic descriptions are forbidden. A role also has *at-least* and *at-most* restrictions which constrain how many fillers it may have. Restrictions on role R are referred to as *value-restriction(R)*, *fillers(R)*, *at-least(R)*, and *at-most(R)*; they default to THING (the universal concept), the empty set, 0, and ∞ respectively. A concept or individual may inherit from *base* concepts. Local and inherited properties are combined by logical intersection. The set of roles restricted by concept or individual X is specified as follows, where R_X denotes role R of X:

Definition 1 *The* restricted roles *of concept or individual X are the roles R_X for which value-restriction(R_X) is properly subsumed by* THING, *or* $|$ *fillers(R_X)* $| > 0$, *or at-least(R_X)* > 0, *or at-most(R_X)* $< \infty$.

This paper uses concepts from the KB defined in Figure 1, where *all* gives a value restriction, *exactly* combines at-least and at-most restrictions of the same cardinality, and *the* combines a value restriction with a cardinality restriction of exactly one. The resulting concept taxonomy appears in Figure 2, with primitive concepts (see below) marked by an asterisk. Individuals such as IBM are not shown. The following individual is a COMPUTER-SYSTEM whose primary-storage, secondary-storage and operating-system roles are not yet filled:

```
(define-primitive-concept COMPANY)

(define-primitive-concept CPU)

(define-primitive-concept RISC)

(define-primitive-concept RAM)

(define-primitive-concept DISK)

(define-primitive-concept SYSTEM)

(define-primitive-concept OS
    SYSTEM)

(define-primitive-concept UNIX
    OS)

(define-concept IBM-CPU
    (and CPU (fills vendor IBM)))

(define-concept RISC-CPU
    (and CPU (the technology RISC)))

(define-concept IBM-RISC-CPU
    (and CPU
         (fills vendor IBM)
         (the technology RISC)))

(define-primitive-concept COMPUTER-SYSTEM
    (and SYSTEM
         (the vendor COMPANY)
         (all processor CPU) (at-least 1 processor)
         (all primary-storage RAM) (at-least 1 primary-storage)
         (all secondary-storage DISK)
         (all operating-system OS)))

(define-concept UNIPROCESSOR-SYSTEM
    (and COMPUTER-SYSTEM (exactly 1 processor)))

(define-concept DUALPROCESSOR-SYSTEM
    (and COMPUTER-SYSTEM (exactly 2 processor)))

(define-concept DISKLESS-SYSTEM
    (and COMPUTER-SYSTEM (exactly 0 secondary-storage)))

(define-concept RISC-MULTIPROCESSOR-SYSTEM
    (and COMPUTER-SYSTEM
         (all processor RISC-CPU) (at-least 2 processor)))

(define-concept DUAL-IBM-PROCESSOR-SYSTEM
    (and COMPUTER-SYSTEM
         (all processor IBM-CPU) (exactly 2 processor)))

(define-concept UNIX-RISC-SYSTEM
    (and COMPUTER-SYSTEM
         (all processor RISC-CPU)
         (the operating-system UNIX)))
```

Figure 1: Sample Concept Definitions

```
(create-individual COMPUTER-SYSTEM123
    (and COMPUTER-SYSTEM
         (fills vendor IBM)
         (fills processor 486DX-33MHz-123)))
```

It still inherits restrictions on those roles from COMPUTER-SYSTEM.

COMPUTER-SYSTEM is a primitive concept; whereas fully *defined* concepts specify necessary and sufficient conditions for class membership, *primitive* concepts specify only necessary conditions. Primitive concepts do not subsume other concepts unless the subsumption is explicitly sanctioned, e.g., COMPUTER-SYSTEM subsumes UNIPROCESSOR-SYSTEM. A description's primitiveness is characterized by the primitive concepts among its ancestors (inclusive) in the taxonomy:

Definition 2 *The* primitives *of concept or individual X are the primitive concepts in the set consisting of X and the transitive closure of its base concepts.*

For example, both primitives(COMPUTER-SYSTEM) and primitives(COMPUTER-SYSTEM123) are the set {COMPUTER-SYSTEM, SYSTEM}.

Many DL systems, including K-REP, support explicit declarations that sets of primitive concepts are mutually *disjoint*. Our sample KB has no disjointness declarations for brevity, but declaring suitable disjointness conditions in a KB is crucial to sound knowledge engineering. Moreover, it helps to guide recognition as we shall see.

3 Incremental Instantiation

To help decide if incremental specification of an individual may be finished, we distinguish between *concrete* and *abstract* concepts. In configuration, only concrete concepts can be included *per se* in a finished system. Abstract concepts represent the commonality among a class of concrete concepts. (All concepts in Figure 1 are abstract; concrete concepts are omitted for brevity.) For example, an actual system's processor may be of type 486DX-33MHz,which is concrete (fully specific), but not merely of type CPU, which is abstract (too general). Note that concrete concepts need not be leaves in the taxonomy. It is useful to introduce *bijective instantiation*, which demonstrates that concept C explicitly accounts for each of individual I's primitives and role restrictions:

Definition 3 *Individual I* bijectively instantiates *concept C iff*

1. $primitives(I) \equiv primitives(C)$
2. $restricted\text{-}roles(I) \equiv restricted\text{-}roles(C)$
3. *For every role R on restricted-roles(I)*
   (a) $at\text{-}least(R_I) \geq at\text{-}least(R_C)$
   (b) $at\text{-}most(R_I) \leq at\text{-}most(R_C)$
   (c) $value\text{-}restriction(R_I) \Rightarrow value\text{-}restriction(R_C)$
   (d) $fillers(R_I) \supseteq fillers(R_C)$

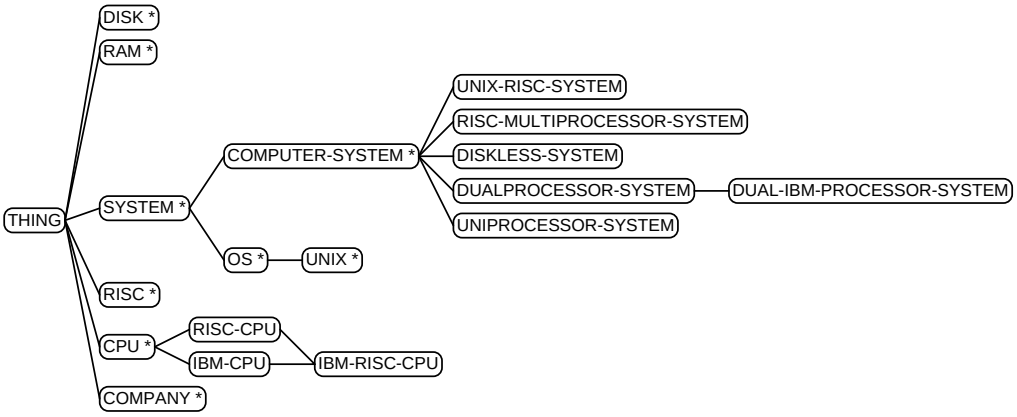


Figure 2: Sample Taxonomy

The distinction between *concrete* and *abstract* concepts is application-specific, but can be characterized in terms of bijective instantiation:

Definition 4 *A knowledge engineer designates concept C as* concrete *to indicate the possibility that an individual which bijectively instantiates C (independent of any subsumees) is sufficiently specific for the purposes of the intended application.*

In configuration, 486DX-33MHz would be designated concrete, whereas more general concepts such as CPU would be designated abstract:

Definition 5 *A concept is presumed to be* abstract *iff it is not concrete.*

The "possibility" alluded to in Definition 4 is realized if the individual in question is also *finished*:

Definition 6 *Individual I is* finished *when*

1. *I bijectively instantiates a concrete concept*
2. *For every role R on restricted-roles(I)*
   (a) $|fillers(R_I)| \geq at\text{-}least(R_I)$
   (b) *Every filler of* R_I *is finished*

The K-REP description language forbids cyclic descriptions, so this definition is well-founded. Observe that COMPUTER-SYSTEM123 instantiates SYSTEM and bijectively instantiates COMPUTER-SYSTEM, but is not yet finished. With respect to configuration, a finished individual system is complete and specific such that it can be ordered from the manufacturer. We could finish COMPUTER-SYSTEM123 by adding suitable role fillers which are themselves finished, including an operating system, one or more memory boards, etc.

4 Predictive Concept Recognition

Once an individual system is finished, the standard DL recognition inference establishes which concepts it instantiates. However, we want to recognize potential instantiations throughout the configuration process to inform both the user and the configuration engine. That is, given an unfinished description of an individual, we want to reason about which concepts it may come to instantiate. Such concepts are *consistent* with the individual. We commit to a CTA:

Definition 7 *Under the* closed-terminology assumption, *all relevant concepts are explicitly defined in the* KB, *and every individual will be finished as in Definition 6.*

Although the set of individuals is not closed, this assumption imposes constraints on the descriptions of individuals, and hence restricts the extension of each concept compared with OTA. It would be interesting to study the formal semantics of these properties.

The CTA is just right for applications like configuration, where domain modeling is completed prior to problem solving. When an individual is no longer subject to update, i.e., a customer has chosen a finished configuration, the CTA ensures that it will bijectively instantiate at least one *ultimate* concrete concept:

Definition 8 *Given that an individual will bijectively instantiate one or more concrete concepts once its description is finalized, those concepts are its* ultimate *concept(s).*

While an ultimate concept restricts every property of an individual, it may do so at an abstract level, in the sense that COMPUTER-SYSTEM requires a processor but only constrains it to be some CPU. Furthermore, the individual may decline to fill optional roles of an ultimate concept by specifying an at-most restriction of zero, e.g., (at-most 0 DISK). We also assume, provisionally, that the individual will be updated monotonically. Monotonic update of an individual entails adding base concepts and/or role restrictions. If our monotonic update assumption proves unfounded, we can back out of it gracefully.

4.1 Closed-Terminology Consistency

Under CTA, we assume that all relevant concepts are explicitly defined in the KB. During configuration, we want

to know which of them are CTA-consistent with a partially described individual system, and which are not. CTA is vital to our methodology: in contrast with OTA, an individual can be monotonically updated to instantiate a concept only if it can be monotonically updated to bijectively instantiate an explicitly defined subsumee of that concept (inclusive). To expedite recognition, we are also concerned with CTA-consistency between concepts. We decide CTA-consistency via the mutually recursive definitions that follow. For a pair of concepts, we must consider two cases. First, a pair of concepts are directly consistent under CTA whenever some bijective instantiation of one can also instantiate the other. We will see that RISC-MULTIPROCESSOR-SYSTEM and DUAL-IBM-PROCESSOR-SYSTEM meet this test. To capture this case, we introduce a *direct consistency* inference from concept C1 to concept C2, written C1 $\mapsto$ C2:

Definition 9 *C1 $\mapsto$ C2 iff*

1. *primitives(C1) $\subseteq$ primitives(C2)*
2. *No primitive of C1 is disjoint from any (additional) primitive of C2*
3. *restricted-roles(C1) $\subseteq$ restricted-roles(C2)*
4. *For every role R on restricted-roles(C1), R_{C1} and R_{C2} are* CTA-*consistent*

Intuitively, C1 $\mapsto$ C2 demonstrates that the primitives and role restrictions of C2 admit a bijective instantiation of C2 which also instantiates C1. Now we must define CTA-consistency for restricted roles:

Definition 10 *Roles R_X and R_Y are* CTA-consistent *iff*

1. *Their cardinality restrictions intersect*
2. *If at-least(R_X) > 0 or at-least(R_Y) > 0, then value-restriction(R_X) and value-restriction(R_Y) are* CTA-*consistent*
3. *| fillers(R_X) $\cup$ fillers(R_Y) | $\leq$ minimum(at-most(R_X), at-most(R_Y))*
4. *Every filler of R_X is* CTA-*consistent with value-restriction(R_Y), and every filler of R_Y is* CTA-*consistent with value-restriction(R_X)*

As discussed below, IBM-CPU and RISC-CPU are CTA-consistent. This, along with overlapping cardinality restrictions, means that DUAL-IBM-PROCESSOR-SYSTEM and RISC-MULTIPROCESSOR-SYSTEM have CTA-consistent processor roles. In addition, both concepts inherit all their primitives and remaining role restrictions from COMPUTER-SYSTEM, so DUAL-IBM-PROCESSOR-SYSTEM $\mapsto$ RISC-MULTIPROCESSOR-SYSTEM and *vice versa*. We conclude they are CTA-consistent.

There is a second, indirect case of CTA-consistency between concepts. Even if two concepts are not directly consistent, their consistency may still be explicitly sanctioned by a common (user-defined) subsumee. This situation can arise when each concept has a primitive or restricted role that the other lacks. For example, only from the existence of IBM-RISC-CPU can we conclude that IBM-CPU and RISC-CPU are CTA-consistent. As a result, we have:

Definition 11 *Concepts C1 and C2 are* CTA-consistent *iff C1 $\mapsto$ C2, or C2 $\mapsto$ C1, or there exists an explicit concept C3 such that C1 and C2 both subsume C3.*

This definition is justified as follows:

Theorem 1 *Under* CTA, *the extensions of concepts C1 and C2 intersect iff they are* CTA-*consistent.*
Proof: *See [Weida,1995].*

Now we examine CTA-consistency between an individual and a concept. A *direct consistency* inference from individual I to concept C, written I $\mapsto$ C, essentially follows Definition 9:

Definition 12 *I $\mapsto$ C iff*

1. *primitives(I) $\subseteq$ primitives(C)*
2. *No primitive of I is disjoint from any (additional) primitive of C*
3. *restricted-roles(I) $\subseteq$ restricted-roles(C)*
4. *For every role R on restricted-roles(I), R_I and R_C are* CTA-*consistent*

Intuitively, direct consistency of an individual with a concept establishes that the individual can bijectively instantiate the concept, perhaps after monotonic updates. Due to CTA and our monotonic observation assumption, an individual is always directly consistent with its ultimate concept(s). Consider this individual:

```
(create-individual COMPUTER-SYSTEM45
    (and COMPUTER-SYSTEM
         (all processor RISC-CPU)))
```

Although COMPUTER-SYSTEM45 is directly consistent with both RISC-MULTIPROCESSOR-SYSTEM and DUAL-IBM-PROCESSOR-SYSTEM, it instantiates neither: regarding RISC-MULTIPROCESSOR-SYSTEM, the cardinality of its processor role is not known to be at least 2, and regarding DUAL-IBM-PROCESSOR-SYSTEM, its processor role is neither restricted to fillers of type IBM-CPU nor a cardinality of exactly 2. However, monotonic updates might still add these restrictions later.

In the context of a KB, if an individual can be monotonically updated to instantiate some concept, then it can be monotonically updated to every subsumer of that concept. For example, imagine that the KB of Figure 1 also contained this concept:

```
(define-concept IBM-PROCESSOR-DEVICE
    (all processor IBM-CPU))
```

It would be classified with THING as its parent and DUAL-IBM-PROCESSOR-SYSTEM as its child. Then COMPUTER-SYSTEM45 would be indirectly consistent with IBM-PROCESSOR-DEVICE by way of DUAL-IBM-PROCESSOR-SYSTEM. Not all indirectly consistent concepts can be identified this way. Consider:

```
(create-individual COMPUTER-SYSTEM67
    UNIPROCESSOR-SYSTEM)
```

We could not make a direct consistency inference from COMPUTER-SYSTEM67 to IBM-PROCESSOR-DEVICE or to its only subsumee, DUAL-IBM-PROCESSOR-SYSTEM.

Still, COMPUTER-SYSTEM67 could be monotonically updated to:

(and UNIPROCESSOR-SYSTEM
(the processor IBM-CPU))

Then COMPUTER-SYSTEM67 would instantiate UNIPROCESSOR-SYSTEM such that it also instantiates IBM-PROCESSOR-DEVICE. To generalize, if individual I can potentially instantiate concept C' such that it also instantiates concept C, then I is *indirectly consistent* with C via C':

Definition 13 *Individual I and concept C are* indirectly CTA-consistent *iff there exists a concept C' such that I* $\mapsto$ *C', C* $\mapsto$ *C', and For every role R restricted by both I and C,* R_I *and* R_C *are* CTA-*consistent.*

CTA-consistency identifies the concepts an individual might instantiate once it is finished:

Definition 14 *Individual I and concept C are* CTA-consistent *iff they are directly or indirectly* CTA-*consistent.*

Its correctness is established by the following:

Theorem 2 *Under* CTA, *individual I can be monotonically updated to instantiate concept C iff I and C are* CTA-*consistent.*
Proof: *See [Weida,1995].*

All CTA-consistency relationships must hold under OTA too, but the converse is not true. For example, IBM-CPU and RISC-CPU are inherently OTA-consistent, but CTA-consistent only when a common subsumee has been explicitly defined.

Our CTA-consistency inferences are syntactic, hence somewhat dependent on the DL language under consideration. This framework can be extended for other description-forming operators found in K-REP, e.g., the existential role restriction operator, *some*. It also appears extensible for the core operators in the version of CLASSIC [Borgida *et al.*,1989] used in PROSE. We are not presently concerned with disjunction and negation operators because K-REP does not support them. It is not clear how they might be incorporated in the present framework, but importantly, CLASSIC has achieved notable success in the configuration arena without supporting either disjunction or negation [Wright *et al.*,1993].

4.2 Knowledge Base Augmentation

Testing for indirect consistency straight from Definition 13 would mean repeated run-time searches for a concept C' by which indirection is licensed. To speed this process, we augment the KB with concepts for internal use as follows:

Definition 15 *A* KB *is* augmented *iff for all concepts C1 and C2 such that C1* $\mapsto$ *C2, there exists an explicit concept C3 such that C3* $\equiv$ *C1* $\wedge$ *C2.*

For a KB containing only UNIPROCESSOR-SYSTEM and UNIX-RISC-SYSTEM, we would add a concept defined as:

(and COMPUTER-SYSTEM
(the processor RISC-CPU)
(the operating-system UNIX))

Its processor role reflects a value restriction of RISC-CPU from UNIX-RISC-SYSTEM and a cardinality restriction of exactly one from UNIPROCESSOR-SYSTEM. The insight is that whenever individual I instantiates concepts C1 and C2 simultaneously, it must instantiate their conjunction. In an augmented KB, possible conjunctions of directly CTA-consistent concepts are made explicit as system-defined concepts when the user has not already defined them. Then, two concepts are CTA-consistent just in case they have a common subsumee (subsumption is reflexive). Consequently, we can reduce indirect consistency testing to subsumption checking:

Definition 16 *Individual I is* indirectly CTA-consistent *with concept C in an augmented* KB *iff there exists a concept C' such that I* $\mapsto$ *C' and C'* $\Rightarrow$ *C.*

With an augmented KB, Definition 16 is correct:

Theorem 3 *Under* CTA *and with an augmented* KB, *individual I can be monotonically updated to instantiate concept C iff I and C are* CTA-*consistent using Definition 16 for indirect consistency instead of Definition 13.*
Proof: *See [Weida,1995].*

Recall our assumption that an individual is directly consistent with an ultimate concept, which is a concrete concept. Thus, augmentation by Definition 15 can be limited to cases where C2 is a concrete. We need only augment a KB once, after its development concludes and before problem solving begins. An augmented KB offers simple, fast identification of indirectly consistent concepts by traversing explicit subsumption links. The number of system-defined concepts in an augmented KB should be manageable in practice, assuming sufficiently distinct primitiveness among user-defined concepts. We will test this hypothesis empirically.

4.3 Knowledge Base Partitioning

Given an individual I, a KB, and our assumptions, predictive recognition determines every concept's status, or *modality*, with respect to I. We will say that concept C is *necessary* with respect to I if I instantiates C, else *optional* if I is consistent with C, else *impossible*. When C is optional, it is possible but not necessary that I will ultimately instantiate C. These definitions implicitly partition the taxonomy into regions as shown in Figure 3. Such a partition constitutes the *recognition state* of I.

As an individual is incrementally updated, we can track its recognition state. Initially, all concepts are optional except for the vacuous root concept, THING, which is trivially necessary. The monotonic update assumption implies that each update will change the modality of zero or more optional concepts to necessary or impossible. Referring to Figure 3, observe that the necessary and impossible regions expand as the individual is updated, further confining the optional region. (It is straightforward to test if an update is nonmonotonic. When this

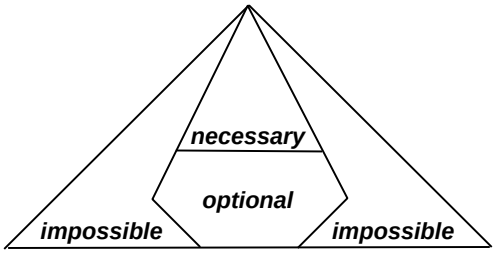


Figure 3: Partition of Concept Taxonomy

happens, we may need to re-expand the optional region.) Our objective is to efficiently bound the portion of the taxonomy containing optional concepts, thereby limiting the number of concepts which must be compared with the individual. We exploit the subsumption-based organization of the KB in two ways. First, the current partition of the KB is used to compute the next one. Second, the consequences of comparing an individual with a concept are propagated to other concepts. To these ends, we distinguish three sets of concepts:

1. The most specific necessary concepts, or MSNs.
2. The most general optional concepts, or MGOs.
3. The most specific optional concepts, or MSOs.

The frontiers of the optional region are maintained by the MGOs and the MSOs. These sets need not be disjoint. The MSNs serve to speed computation of the MSOs. Further details are omitted for brevity.

An initial user-specified configuration might be:

```
(create-individual COMPUTER-SYSTEM89
    (and COMPUTER-SYSTEM
         (at-least 2 processor)
         (at-least 1 secondary-storage)))
```

Considering Figure 2, the necessary concepts are THING, SYSTEM and COMPUTER-SYSTEM. The optional ones are UNIX-RISC-SYSTEM, RISC-MULTIPROCESSOR-SYSTEM, DUALPROCESSOR-SYSTEM, and DUAL-IBM-PROCESSOR-SYSTEM. The remainder are impossible. This recognition state is captured as:

```
MSNs = {COMPUTER-SYSTEM}
MGOs = {DUALPROCESSOR-SYSTEM,
        UNIX-RISC-SYSTEM,
        RISC-MULTIPROCESSOR-SYSTEM}
MSOs = {DUAL-IBM-PROCESSOR-SYSTEM,
        UNIX-RISC-SYSTEM,
        RISC-MULTIPROCESSOR-SYSTEM}
```

This recognition state indicates precisely which concepts COMPUTER-SYSTEM89 might eventually instantiate. Notice that UNIPROCESSOR-SYSTEM and DISKLESS-SYSTEM are ruled out. In fact, all other concepts which do not subsume COMPUTER-SYSTEM are ruled out, e.g., COMPANY is impossible. If the user now states that COMPUTER-SYSTEM89 should have exactly 4 processors, the subsequent recognition state will also rule out DUALPROCESSOR-SYSTEM and DUAL-IBM-PROCESSOR-SYSTEM:

```
MSNs = {COMPUTER-SYSTEM}
MGOs = {UNIX-RISC-SYSTEM,
        RISC-MULTIPROCESSOR-SYSTEM}
MSOs = {UNIX-RISC-SYSTEM,
        RISC-MULTIPROCESSOR-SYSTEM}
```

Notice how the MGOs succinctly capture the most general choices available in the current state. A user interface can exploit this by prompting the user to choose a subset of the MGOs which describe the desired system.

5 Constraint Derivation

We can take further advantage of predictive recognition to derive additional constraints on an individual. Suppose the KB is partitioned according to the current description of some individual, I. If I does not bijectively instantiate an MSN concept which is concrete, then by CTA, I will ultimately instantiate some optional concept. Therefore, the commonality among MGO concepts constitutes a set of implicit constraints imposed on I by the taxonomy under CTA. For example, if RISC-UNIPROCESSOR-SYSTEM and RISC-DUALPROCESSOR-SYSTEM are the MGOs for an individual, then it has at most 2 processors and all its processors are RISC-CPUs. Cohen, et al., have shown how to compute the *least common subsumer* (LCS) of a set of concepts [Cohen *et al.*,1992]. The LCS of the MGOs, written LCS(MGOs), is a concept representing their commonality (we never actually install the LCS in the KB). Any constraint on LCS(MGOs) not reflected in I should be added to I. The basic strategy is: until arriving at a fixed point, repeatedly

1. Compute LCS(MGOs)
2. If LCS(MGOs) implies further constraints on I
   (a) Fold LCS(MGOs) into the description of I, and
   (b) Incrementally update the recognition state of I

In this way, K-REP may be able to infer constraints on I after it is first created and whenever it is updated. Similar reasoning applies recursively to fillers of I's roles.

Recall the second recognition state for COMPUTER-SYSTEM89 in Section 4.3, and note that COMPUTER-SYSTEM is not a concrete concept. By computing the LCS of {UNIX-RISC-SYSTEM, RISC-MULTIPROCESSOR-SYSTEM}, K-REP discovers that COMPUTER-SYSTEM89's processors must be of type RISC-CPU. This conclusion can only come from closed-terminology reasoning about the KB's specific contents. As a result, future choices regarding the processors will be suitably constrained. The preceding discussion was simplified for clarity; we can sometimes do better because ultimately I must instantiate one of the most general optional concepts which are *concrete*.

6 Related Work

An early use of DL for configuration was reported in [Owsnicki-Klewe,1988], which cast the entire configuration problem as a matter of maintaining internal

KB consistency, i.e., logical contradictions should follow from all invalid configurations but no valid ones. This goal is more ambitious than ours or that of [Wright *et al.*,1993]. However, [Owsnicki-Klewe,1988] considered only "an (admittedly limited) example" and did not close the terminology. PROSE is a successfully deployed configurator featuring product KBs written in CLASSIC [Wright *et al.*,1993]. Like PROSE, our work positions the DL system as a product knowledge reasoner — a key module in a larger configurator architecture. In both cases, the DL system maintains internal KB consistency during configuration but relies on configuration-specific modules for further reasoning. We share the views of the PROSE developers that (1) DL fosters a reasonable, even natural approach to product knowledge representation, and (2) knowledge engineering efforts benefit from enforcing internal KB consistency.

This work draws on a predictive recognition methodology originally formulated for constraint networks representing rich temporal patterns, e.g., plans [Weida and Litman,1994; Weida and Litman,1992]. The present paper addresses new issues which arise in this setting, e.g., the presence of primitives and the interaction among types of role restrictions. Our configuration application led us to introduce the distinction between abstract and concrete concepts to DL. It also led us to devise inferences for characterizing the progress of incremental concept instantiation, i.e., bijective and finished instantiation. Other contributions of this paper include an incremental strategy for partitioning a subsumption-based taxonomy by modality, and novel use of the LCS inference to extract constraints on an individual from the KB during predictive recognition. Our predictive recognition algorithm bears an interesting resemblance to the candidate-elimination algorithm of [Mitchell,1982], however the candidate-elimination algorithm is given explicit negative examples that are used to exclude concepts, while we derive the concepts to be excluded using the CTA. Another difference is that candidate-elimination operates on sets of different positive and negative examples, while we are concerned with successive descriptions of the same (positive) instance. The work described in this paper is the first to pursue closed-terminology reasoning over a description logic KB.

7 Conclusion

This paper presents a predictive concept recognition methodology for DL which demonstrates, for the first time, the value of closed-terminology reasoning over a DL KB. These ideas apply to tasks such as configuration where all relevant concepts are known in advance. We take advantage of the closed taxonomy in several ways. As an individual system is incrementally described, we efficiently track the types of systems which may result. We partition the KB by categorizing concepts as necessary, optional or impossible with respect to the current description. This information is inherently useful for both user and configuration engine. We also exploit the current partition to derive implicit constraints on the system. This, in turn, may yield a more refined partition. Similar reasoning applies recursively to the system's components. Finally, we take advantage of the derived constraints to inform the user and the configuration engine, and to appropriately restrict future choices. Although we focused on configuration, our methodology is domain-independent.

Acknowledgements

I am extremely grateful to Diane Litman, Steffen Fohn, Bonnie Webber, Brian White, and an anonymous reviewer for valuable comments on earlier drafts, and to Eric Mays for useful discussions.

References

[Borgida *et al.*, 1989] A. Borgida, R. J. Brachman, D. L. McGuinness, and L. A. Resnick. Classic: A structural data model for objects. In *SIGMOD-89*, pages 58–67, 1989.

[Cohen *et al.*, 1992] W. Cohen, A. Borgida, and H. Hirsh. Computing least common subsumers in description logics. In *AAAI-92*, 1992.

[Mays *et al.*, 1991] E. Mays, R. Dionne, and R. Weida. K-rep system overview. *SIGART Bulletin*, 2(3):93–97, June 1991.

[Mitchell, 1982] T. M. Mitchell. Generalization as search. *Artificial Intelligence*, 18:203–226, 1982.

[Owsnicki-Klewe, 1988] B. Owsnicki-Klewe. Configuration as a consistency maintenance task. In *GWAI-88*, pages 77–87, Berlin, Germany, 1988.

[Weida and Litman, 1992] R. Weida and D. J. Litman. Terminological reasoning with constraint networks and an application to plan recognition. In *KR'92*, 1992.

[Weida and Litman, 1994] R. Weida and D. Litman. Subsumption and recognition of heterogeneous constraint networks. In *CAIA-94*, 1994.

[Weida, 1995] R. Weida. Closed-world reasoning and temporal reasoning in description logic for concept and plan recognition. Forthcoming Ph.D. dissertation, Columbia University, 1995.

[Woods and Schmolze, 1992] W. A. Woods and J. G. Schmolze. The kl-one family. *Computers and Mathematics with Applications*, 74(2-5), 1992.

[Wright *et al.*, 1993] J. R. Wright, E. S. Weixelbaum, G. T. Vesonder, K. E. Brown, S. R. Palmer, J. I. Berman, and H. H. Moore. A knowledge-based configurator that supports sales, engineering, and manufacturing at at&t network systems. *AI Magazine*, 14(3), 1993.

Towards a unified architecture for knowledge representation and reasoning based on terminological logics *

Liviu Badea
AI Research Department
Research Institute for Informatics
8-10 Averescu Blvd., Bucharest, ROMANIA
e-mail: badea@roearn.ici.ro

Abstract

This paper presents a *unified* architecture for knowledge representation and reasoning based on terminological (description) logics. The novelty of our approach consists in trying to use description logics not only for representing domain knowledge, but also for describing beliefs, epistemic operators and actions of intelligent agents in an unitary framework. For this purpose, we have chosen a decidable terminological language, called $\mathcal{ALC}_{reg+id(C)}$, whose expressivity is high enough to be able to represent actions and epistemic operators corresponding to the majority of modal logics of knowledge and belief.

Additionally, we describe practical inference algorithms for the language $\mathcal{ALC}_{reg+id(C)}$ which lies at the heart of our $\mathcal{RegAL}$ [1] knowledge representation system. The algorithms are sound and *complete* and can be used directly for deciding the validity and satisfiability of formulas in the propositional dynamic logic (PDL) by taking advantage of the correspondence between PDL and certain terminological logics [10].

1 Term subsumption languages

Term subsumption languages [2] (TSLs) are descendants of the famous KL-ONE language [4] and can be viewed as formalizations of the frame-based knowledge representation systems.

The relationship between TSLs and logic is analogous to the relationship between structured and unstructured programming languages. Indeed, the TSLs impose a certain discipline in the logical structure of a formula (concept) in the very same way in which the structured programming paradigm imposes a discipline in the control structure of a program. Although they somehow restrict the expressivity of the description language, TSLs are most of the time preferable to general logic because of their increased understandability and usability in building practical knowledge bases. Also, as opposed to general logic, certain TSLs may possess *decidable* inference problems while retaining a fairly high expressivity which enables them to represent complex ontologies.

The terminological description language usually provides a variety of concept and role constructors, including the boolean operators (conjunction $\sqcap$, disjunction $\sqcup$, and negation $\neg$). Value- $(\forall R{:}C)$, existential- $(\exists R{:}C)$ and number restrictions $(\leq_n R, =_n R, \geq_n R)$, role-value maps $(R_1 \sqsubset R_2)$ and structural descriptions $(C : R)$ are some of the most important concept constructors. We could also mention the following role constructors: $id(C)$ (the restriction of the identity role to the concept C), R^{-1} (role inverse), $R\lfloor C$ (range restriction), $R_1 \circ R_2$ (role composition), R^* (reflexive-transitive closure) and $R_1 \prec R_2$ (bindings used in structural descriptions).

Not all of the above constructors are independent. For example, role-value maps and structural descriptions can be expressed in a language that admits role negation [3] as:

$$\begin{aligned} R_1 \sqsubset R_2 &= \forall(R_1 \sqcap \neg R_2){:}\bot \\ C : R &= \exists R{:}C, \ \textit{where} \\ R &= R_1 \prec Q_1 \sqcap \ldots \sqcap R_n \prec Q_n \\ R_i \prec Q_i &= \neg(R_i \circ \neg Q_i^{-1}). \end{aligned}$$

Role-value maps and structural descriptions usually lead to very expressive but *undecidable* languages [12; 8]. These observations suggest the following conjecture: *"The only cause of undecidability of a reasonably expressive terminological language is the irreducible presence of role negation in the language."* Note that concept negation is usually harmless w.r.t. decidability, as opposed to role negation which usually leads to undecidable languages [9].

*This research was partially supported by the European Community project $^{PE}KADS$ (CP-93-7599).

[1] The $id(C)$ - *Reg*ular closure of the $\mathcal{ALC}$ language.

[2] Also known as terminological (or description) logics.

[3] and also other common concept and role constructors

2 Complete decision algorithms for the terminological language $\mathcal{ALC}_{reg+id(C)}$

The terminological language we are using in our knowledge representation system $\mathcal{RegAL}$ is $\mathcal{ALC}_{reg+id(C)}$, the regular closure of the well-known language $\mathcal{ALC}$ of Schmidt-Schauß and Smolka [11] extended with the role constructor $id(C)$.

In the following, we shall present *complete* inference algorithms [4] for $\mathcal{ALC}_{reg+id(C)}$. By taking advantage of the correspondence of $\mathcal{ALC}_{reg+id(C)}$ with the propositional dynamic logic (PDL) of programs [10], we shall be able to apply our algorithms for deciding the validity and satisfiability of formulas in PDL too.

As far as we know, there exists a single TSL system with complete inference algorithms and a reasonably high expressivity, namely $\mathcal{KRIS}$ [2]. The terminological language $\mathcal{ALCFNR}$ provided by $\mathcal{KRIS}$ extends the standard language $\mathcal{ALC}$ with attributes (functional roles), number restrictions and role conjunctions.

The language $\mathcal{ALC}_{reg+id(C)}$ we are using in $\mathcal{RegAL}$ was chosen having somewhat different goals in mind, namely to be able to represent procedural knowledge, actions and epistemic operators in our descriptive logic. Number restrictions and role conjunctions wouldn't have been very helpful in this context.

The satisfiability (consistency) of a concept in our terminological language can be tested by using a variant of the well known *tableaux calculus*, adapted to this specific context [6]. Starting from a formula which implicitly asserts the satisfiability of the given concept, the calculus tries to construct a model of the respective formula. In doing so, it may discover obvious contradictions (clashes) and report the inconsistency of the original formula, or it may come up with a complete clash-free model, thus proving the satisfiability of the formula. This method is directly applicable only if the language possesses the *finite model property* (which is fortunately the case with $\mathcal{ALC}_{reg+id(C)}$).

The tableaux calculus combines two different processes. The first is analogous to a refutation theorem prover which tries to discover contradictions, while the second concentrates on building models. In [6] a variant of the tableaux calculus (called rule-based calculus operating on constraints) is used for obtaining complete decision procedures for the satisfiability problem in the languages ranging between $\mathcal{ALC}$ and $\mathcal{ALCFNR}$. On the other hand, Franz Baader [1] succeeds in obtaining a practical decision algorithm for the regular closure $\mathcal{ALC}_{reg}$ of $\mathcal{ALC}$. As far as we know, no practical decision algorithms for languages more expressive than $\mathcal{ALC}_{reg}$ are known.

Adding the role constructor $id(C)$ to the language $\mathcal{ALC}_{reg}$ increases the expressivity but introduces substantial complications in the inference algorithms. These complications are mainly due to the fact that existential restrictions are no longer *separable* in the language $\mathcal{ALC}_{reg+id(C)}$.

The complete satisfiability checking algorithm is a consequence of the *reduction* and *cycle-characterization* theorems presented in [3]. The idea of the algorithm consists in reducing the satisfiability of a given concept to the satisfiability of several simpler concepts. This reduction process can be alternatively viewed as a process of model construction. In order to ensure the termination of the algorithm, we have to check for the presence of cycles at each reduction step. In case a cycle has been detected, the cycle-characterization theorem is used to determine its nature. As in the case of $\mathcal{ALC}_{reg}$, only the good cycles lead to a model, the bad cycles being merely shorthands for infinite reduction chains.

The satisfiability testing algorithm, presented in figure 1, involves a *preprocessing step* in which the following computations are performed:

1) The concept C to be tested is brought to the *negation normal form (nnf)*. The main difference viz. $\mathcal{ALC}_{reg}$ consists in having to consider the concepts I within $id(I)$ roles too. This has to be done depending on the context in which the role $id(I)$ appears (i.e. within an $\forall$ or a $\exists$ restriction) in order to facilitate the extraction of the proper conjuncts of C. More precisely, if $id(I)$ appears in an $\exists$ restriction, then $rnf_{\exists}(id(C)) = id(nnf(C))$, and if it occurs in an $\forall$ restriction, then $rnf_{\forall}(id(C)) = id(\neg nnf(\neg C))$.
2) Since comparisons between role expressions R occurring in C are quite frequent (especially when testing the existence of cycles), it seems to be a good idea to bring the roles R to a canonical form. This can be done by constructing for each role R the corresponding deterministic finite automaton *DFA* and by minimizing the disjoint union of these automata. The initial states of the resulting minimal deterministic finite automaton *mDFA* represent the canonical forms of the roles occurring in C.
3) Finally, the procedure *roles_to_mStates* replaces the roles occurring in C with the corresponding states of the *mDFA*. The replacements affect the concepts I inside $id(I)$ transitions of the *mDFA* too.

In the following, we shall make no distinction between a role, its corresponding state in the *mDFA* and the language accepted starting from this state. Also, the following substitutions are performed for all value- and existential restrictions in which $\varepsilon \in R$ (or, equivalently, the state of the *mDFA* corresponding to R is final):

$$\begin{aligned} \forall R{:}\,Ca &\rightarrow Ca \sqcap \forall(R \setminus \{\varepsilon\}){:}\,Ca \\ \exists R{:}\,Ce &\rightarrow Ce \sqcup \exists(R \setminus \{\varepsilon\}){:}\,Ce. \end{aligned}$$

The actual satisfiability testing algorithm extracts a conjunct of the given concept at a time, removes the *separable* existential restrictions and subsequently tries to determine the satisfiability of the remaining *nonseparable* conjunct.

[4] The validity and satisfiability problems in $\mathcal{ALC}_{reg+id(C)}$ are known to be decidable (more precisely, EXPTIME–complete).

$satisfiable(C)$
  $C' \leftarrow nnf(C)$
  $uDFA \leftarrow \emptyset$
  forall *roles* R *occurring in* C'
    $DFA \leftarrow role\_to\_DFA(R)$
    $uDFA \leftarrow DFA \cup uDFA$
  □
  $mDFA \leftarrow minimize(uDFA)$
  $C'' \leftarrow roles\_to\_mStates(C')$
  $sat(C'', [])$
□

$sat(C, L)$
  $Conj \leftarrow conjunct(C)$
  $sat\_conjunct(Conj, L)$
□

$sat\_conjunct(Conj, L)$
  if $cycle(Conj, L, \uparrow GoodBad)$ **then**
    if $GoodBad = good$ **then** *succeed*
    else *fail*
  else
    $\overline{Conj} \leftarrow proper\_conjunct(Conj)$
    assign a new unique label Ne *to all*
      $\exists^{no\_label} Re{:}Ce$ *restrictions*
    // $\overline{Conj} = \prod_i C_i \sqcap \prod_j \exists^{Ne} Re_j{:}Ce_j \sqcap \prod_k \forall\overline{Ra_k}{:}Ca_k$
    if $\prod_i C_i$ *contains a clash (i.e.* $C_{i_1} = \neg C_{i_2}$*)* **then**
      fail
    else
      // *solve the separable* $\exists$ *restrictions*
      // *and collect the nonseparable ones*
      $NS\_E \leftarrow sat\_separable\_exists(\prod_j \exists Re_j{:}Ce_j$
        $\sqcap \prod_k \forall\overline{Ra_k}{:}Ca_k, [\sqcap\_node(Conj)|L])$
      // *solve the nonseparable* $\exists$ *restrictions*
      $sat\_nonseparable\_exists(\prod_i C_i \sqcap NS\_E$
        $\sqcap \prod_k \forall\overline{Ra_k}{:}Ca_k, [\sqcap\_node(Conj)|L])$
    □
  □
□

$sat\_exists(\overline{C}_\exists, L)$
  $sat\_exists\_solved(\overline{C}_\exists, L)$
  or // *nondeterministic choice*
  $sat\_exists\_postponed(\overline{C}_\exists, L)$
□

Figure 1: The satisfiability testing algorithm for concepts in $\mathcal{ALC}_{reg+id(C)}$

$sat\_separable\_exists(\prod_j \exists Re_j{:}Ce_j \sqcap \prod_k \forall\overline{Ra_k}{:}Ca_k, L) \rightarrow NS\_E$
  // NS_E = *conjunct of nonseparable* $\exists$ *restrictions*
  $NS\_E \leftarrow \top$
  forall $\exists Re{:}Ce$ *in* $\prod_j \exists Re_j{:}Ce_j$
    $sat\_exists(\exists\overline{Re}{:}Ce \sqcap \prod_k \forall\overline{Ra}_k{:}Ca_k, L)$
    or // *nondeterministic choice*
    $NS\_E \leftarrow \exists Re{:}Ce \sqcap NS\_E$
  □
  return NS_E
□

$sat\_nonseparable\_exists(\prod_i C_i \sqcap NS\_E \sqcap \prod_k \forall\overline{Ra_k}{:}Ca_k, L)$
  $C \leftarrow \prod_i C_i \sqcap \prod_k \forall\overline{Ra}_k{:}Ca_k$
  forall $\exists^{Ne} Re{:}Ce$ *in* NS_E
    if $id(I) \in Re$ **then**
      $C \leftarrow (I \sqcap Ce) \sqcap C$
    else *fail*
    or // *nondeterministic choice*
    if $id(I)^{-1}Re \setminus \{\varepsilon\} \neq \emptyset$ **then**
      $C \leftarrow \left[I \sqcap \exists^{Ne}(id(I)^{-1}Re \setminus \{\varepsilon\}){:}Ce\right] \sqcap C$
    else *fail*
  □
  $sat(C, L)$
□

$sat\_exists\_solved(\overline{C}_\exists, L)$
  // $\overline{C}_\exists = \exists\overline{Re}{:}Ce \sqcap \prod_k \forall\overline{Ra}_k{:}Ca_k$
  if *there exists an* $R \in Re$ *such that* $R \neq id(\cdot)$ **then**
    $Ca' \leftarrow \prod_{k,\ R \in Ra_k} Ca_k \sqcap \prod_{k,\ R^{-1}Ra_k \setminus \{\varepsilon\} \neq \emptyset} \forall(R^{-1}Ra_k \setminus \{\varepsilon\}){:}Ca$
    // *solve the* $\exists$ *restriction*
    $sat(Ce \sqcap Ca', L)$
  else *fail*
□

$sat\_exists\_postponed(\overline{C}_\exists, L)$
  // $\overline{C}_\exists = \exists^{Ne}\overline{Re}{:}Ce \sqcap \prod_k \forall\overline{Ra}_k{:}Ca_k$
  let $R^{-1}Re$ *be the target state of the transition*
    $Re \xrightarrow{R} R^{-1}Re$ *with* $R \neq id(\cdot)$
  $Ca' \leftarrow \prod_{k,\ R \in Ra_k} Ca_k \sqcap \prod_{k,\ R^{-1}Ra_k \setminus \{\varepsilon\} \neq \emptyset} \forall(R^{-1}Ra_k \setminus \{\varepsilon\}){:}Ca$
  // *postpone the* $\exists$ *restriction*
  $sat(Ca' \sqcap \exists^{Ne}(R^{-1}Re \setminus \{\varepsilon\}){:}Ce, L)$
□

Definition 1 *A restriction* $\exists Re_j : Ce_j$ *is called* **separable** *w.r.t. the proper conjunct* [5] $\overline{C}_\sqcap = \prod_i C_i \sqcap \prod_j \exists Re_j : Ce_j \sqcap \prod_k \forall \overline{Ra}_k : Ca_k$ *iff the concept* $\overline{C}_{\exists_j} = \exists \overline{Re}_j : Ce_j \sqcap \prod_k \forall \overline{Ra}_k : Ca_k$ *is satisfiable. The proper conjunct* $\overline{C}_\sqcap$ *itself is called* **nonseparable** *iff none of its* $\exists Re_j : Ce_j$ *restrictions is separable.*

There are two possibilities of proving the satisfiability of the concept $\overline{C}_{\exists_j}$, namely by *solving* the existential restriction, or by *postponing* it.

In a similar way, the (nonseparable) existential restrictions from a nonseparable conjunct can be solved or postponed w.r.t. $id(I)$ transitions, but they cannot be separated because of possible interactions between the concepts I.

In order to be able to determine whether a given existential restriction has been obtained by *postponing* or by *solving* another existential restriction involved in a cycle, we shall attach a unique label N to each existential restriction $\exists^N Re : Ce$.

All existential restrictions are initially unlabeled. An unlabeled restriction $\exists^{no\_label} Re : Ce$ receives a new unique label N_j only when it reaches the "top level" of a conjunct[6] $C_\sqcap = \prod_i C_i \sqcap \prod_j \exists^{N_j} Re_j : Ce_j \sqcap \prod_k \forall Ra_k : Ca_k$.

When an existential restriction is postponed, its label is conserved and can be used to track an uninterrupted chain of postponings. Such a chain cannot correspond to a model unless at least one of the existential restrictions in the chain is eventually solved.

In the following, we shall see how the labels can be used to determine the nature of cycles. Let $\overline{C}_\sqcap$ and $\overline{C}'_\sqcap$ be the two concepts involved in a cycle. $\overline{C}_\sqcap$ and $\overline{C}'_\sqcap$ are equal, except maybe the labels N_j and N'_j of the existential restrictions $(j = 1, \ldots, n)$. Such a cycle will be represented by the *label-correspondence table* $\begin{pmatrix} N_1 & N_2 & \ldots & N_n \\ N'_1 & N'_2 & \ldots & N'_n \end{pmatrix}$, each column of this table being related to equal existential restrictions $\exists^{N_i} Re : Ce = \exists^{N'_i} Re : Ce$ from $\overline{C}_\sqcap$ and $\overline{C}'_\sqcap$ respectively. Because $\exists$ restrictions get unique labels when they reach the top level of a conjunct, we have $N_i \neq N_j$ and $N'_i \neq N'_j$ for $i \neq j$. The following theorem can be used in determining the nature of a cycle.

Theorem 1 *(cycle characterization)*

A cycle represented by the label correspondence table above is **bad** *(i.e. it does not induce a model) iff the label correspondence table contains a cyclic permutation, i.e. there exists a subset of indices* $\{j_1, j_2, \ldots, j_k\} \subset \{1, \ldots, n\}$ *such that* $N_{j_1} = N'_{j_2}$, $N_{j_2} = N'_{j_3}$, $\ldots$, $N_{j_{k-1}} = N'_{j_k}$, $N_{j_k} = N'_{j_1}$.

3 Representing epistemic operators in terminological logics

Since we are aiming at a unified architecture for knowledge representation based on terminological logics, we shall show that TSLs are powerful enough to represent epistemic operators corresponding to the majority of modal logics of knowledge and belief. Not only is it possible to describe in $\mathcal{RegAL}$ the knowledge/beliefs of several agents, but the different agents could have different epistemic operators with distinct modal properties so that we could study, for example, the interaction between an agent whose *knowledge* is necessarily true and another agent whose *beliefs* are just *consistent* and *believed to be true*, but not necessarily true in reality. One could even have more than one epistemic operator attached to the same agent in order to distinguish its beliefs from its knowledge.

Of course, in $\mathcal{RegAL}$ epistemic operators can be nested in an unrestricted fashion and they could even mention actions and plans. Also, the actions of some agent could modify the knowledge or beliefs of another agent so that it becomes possible to study the *communication* between agents in a unified framework.

In modal logic, an agent can imagine a set of possible worlds linked with the real world by the *accessibility relation*. The facts p known by the agent are facts which are true in all possible worlds.

Modal formulas are constructed by using the usual logical connectives together with the modal operators $\Box$ (necessity) and $\Diamond$ (possibility). The necessity modal operator $\Box$ will be interpreted in the following as an epistemic operator, the formula $\Box p$ being understood as "the agent knows the fact p".

Because of the fact that there is no unique interpretation of the modal notions of "necessity", "possibility", "knowledge", "belief" etc., there exists a large variety of modal systems which can be distinguished by the properties of the accessibility relation. Imposing, for instance, the reflexivity of the accessibility relation ρ in the modal system **T** is equivalent to requiring the truth of knowledge, while imposing the seriality of ρ leads to the consistency of knowledge. The table 1 presents some of the most common modal axioms together with the properties of the accessibility relation they induce.

The most common modal systems are defined by combinations of the modal axioms from table 1. They can be embedded in a term subsumption language by using *satisfiability preserving translations* into the TSL (see also [13]). In this way, problems formulated in terms of (modal) epistemic operators can be reduced to problems in a TSL which can be solved using the inference algorithms from the preceding sections.

The general translation scheme from a modal system

[5] In $\mathcal{ALC}_{reg+\,id(C)}$, it is important to distinguish between *simple* and *proper* conjuncts. The *simple conjuncts* are the ones obtained by ignoring possible $id(I)$ roles that could occur in the given concept C. The *proper conjuncts* can be obtained from the simple ones by taking into account the implicit disjunctions induced by possible $id(I)$ transitions of roles Ra occurring in value restrictions $\forall Ra : Ca$. For instance, $\forall id(I) : C = \neg I \sqcup C$.

[6] This happens in *sat_conjunct* after extracting a proper conjunct from a simple one.

Name	Modal axiom	Property of the accessibility relation	Comments
K.	$\Box(p \to q) \to (\Box p \to \Box q)$	valid in every standard Kripke frame	*Kripke's* axiom (*normality* axiom)
D.	$\Diamond\top$	serial	*deontic* axiom
T.	$\Box p \to p$	reflexive	*knowledge* axiom
B.	$p \to \Box\Diamond p$ $p \to \Box\neg\Box\neg p$	symmetric	*Brouwer* axiom
4.	$\Box p \to \Box\Box p$	transitive	*positive introspection* axiom
5.	$\Diamond p \to \Box\Diamond p$ $\neg\Box p \to \Box\neg\Box p$	euclidian	*negative introspection* axiom
U.	$\Box(\Box p \to p)$	almost reflexive	beliefs are believed to be true
(A.)	$\Box(\Diamond\Box p \to p)$	almost symmetric	

Table 1: Major modal axioms

into a TSL is the following (p' is the TSL concept corresponding to the modal formula p):

$$\begin{aligned} p &\longmapsto p \ (for\ atomic\ formulas) \\ \neg p &\longmapsto \neg p' \\ p \wedge q &\longmapsto p' \sqcap q' \\ p \vee q &\longmapsto p' \sqcup q' \\ \Box p &\longmapsto \forall \mathcal{L}(R) : p' \\ \Diamond p &\longmapsto \exists \mathcal{L}(R) : p'. \end{aligned}$$

Note that the modal operators $\Box$ and $\Diamond$ are translated into value- and existential restrictions in which roles of the form $\mathcal{L}(R)$ occur. Here $\mathcal{L}$ is the particular modal system and R an arbitrary role name representing the agent. The role $\mathcal{L}(R)$ stands for the accessibility relation and possesses all the properties this relation should have in the system $\mathcal{L}$. Thus, we could read the formula $\forall \mathcal{L}(R) : p'$ as: "the agent R knows the fact p' w.r.t. the modal system $\mathcal{L}$" ($\mathcal{L}$ gives us here the *type* of knowledge).

The table 2 presents the expression of $\mathcal{L}(R)$ for the most important *modal logics of knowledge* (in which knowledge is required to be true [7]) while the table 3 does the same thing for the *modal logics of belief* (in which beliefs are believed to be true). The axioms of reflexivity **T** and symmetry **B** from the modal logics of knowledge are replaced in the modal logics of belief by the weaker versions **U** (almost reflexivity) and **A** (almost symmetry) respectively.

Adding the deontic axioms $\exists \mathcal{OL}^+(R) : \top$ or, equivalently, $\exists \mathcal{L}(R) : q$ to the systems $\mathcal{OL}(R)$ in table 3 leads to the *deontic systems* $\mathcal{OL}^+(R)$ in which the beliefs are required to be *consistent*. Note that

$$\mathcal{OL}^+(R) = \mathcal{OL}(R) = \mathcal{L}(R) \circ id(q).$$

[7] except perhaps in the system **K**.

System	Axioms	$\mathcal{L}(R)$
K.	K	R
T.	KT	$R \sqcup id$
S4.	KT4	R^*
S5.	KT5	$(R \sqcup R^{-1})^*$
B.	KTB	$R \sqcup id \sqcup R^{-1}$

Table 2: The accessibility relation $\mathcal{L}(R)$ in the modal logics of knowledge

System	Axioms	$\mathcal{OL}(R)/\mathcal{OL}^+(R)$
OK/OK$^+$.	K/KD	$R \circ id(q)$
OT/OT$^+$.	KU/KDU	$(R \sqcup id) \circ id(q)$
OS4/OS4$^+$.	K4U/KD4U	$R^* \circ id(q)$
OS5/OS5$^+$.	K45/KD45	$(R \sqcup R^{-1})^* \circ id(q)$
OB/OB$^+$.	KUA/KDUA	$(R \sqcup id \sqcup R^{-1}) \circ id(q)$

Table 3: The accessibility relations $\mathcal{OL}(R)/\mathcal{OL}^+(R)$ in the modal logics of belief

The main advantage of our unifying approach is that the various types of knowledge corresponding to the aforementioned modal systems can be *amalgamated* in a single system. For example, we could describe a multi-agent system in which the knowledge $\mathcal{K}_i$ and beliefs $\mathcal{B}_i$ of the agents i can be mixed in an unrestricted fashion. By attaching a unique role name R_i to each agent i, we can write the epistemic operators corresponding to the knowledge and belief of agent i in the following way [8]

$$\begin{aligned} \mathcal{K}_i &= [\mathcal{R}_i] = [S4(R_i)] = [R_i^*] \\ \mathcal{B}_i &= [\mathcal{T}_i] = [KD4U(R_i)] = [R_i^* \circ id(q_i)]. \end{aligned}$$

[8] In order to simplify the notation, we shall write, in the following, $[R]C$ instead of $\forall R : C$.

where $\mathcal{T}_i$ verifies the deontic axiom $\exists\mathcal{T}_i{:}\top$, or equivalently, $\exists R_i^*{:}q_i$.

The common knowledge and common belief operators are $\mathcal{C} = [(\coprod_i \mathcal{R}_i)^*]$ and $\mathcal{D} = [(\coprod_i \mathcal{T}_i)^*]$ respectively.

Our method of integrating epistemic operators in a TSL is much simpler and more natural than other approaches [5; 7] which, on one hand, could deal with only one single type of knowledge at a time and, on the other, had to develop special purpose algorithms for treating the epistemic operators (because the underlying TSL had a too low expressivity to be able to express epistemic operators directly).

4 Representing actions and plans in a TSL

TSLs can be used not only for representing the domain knowledge or epistemic operators, but also for describing actions and plans. In order to develop a theory of action in TSLs, we shall regard a role of a TSL as an action which transforms the states x from the extension of the role's domain into the states y from the extension of its range. Thus, the value restriction $\forall R{:}C$ can be interpreted as the necessary precondition for the action R to achieve the postcondition C.

Conditions/facts from our theory of action will be represented in a TSL by concepts, while actions will be denoted by roles. An action $A : \langle In|Ctx|Out\rangle$ (having In as deleted preconditions, Ctx as context (preserved preconditions) and Out as created postconditions) can be described by the following terminological axiom, which is similar to a *total correctness assertion* from dynamic logic [9]

$$In \sqcap Ctx \subset \forall\exists A{:}(\neg In \sqcap Ctx \sqcap Out)$$

where $\forall\exists R{:}C \stackrel{def}{=} \exists R{:}\top \sqcap \forall R{:}C = \exists R{:}C \sqcap \forall R{:}C$.

The *planning problem* can be stated in the following way: "Given an initial state represented by the concept *Initial*, a final state (goal) *Final* and a repertory of actions $\{A_1, A_2, \ldots, A_n\}$, find a role chain $Plan = A_{i_1} \circ A_{i_2} \circ \ldots \circ A_{i_k}$ (or, more generally, a role term *Plan* formed from the roles $A_1, \ldots, A_n$ by applying the role constructors) such that $Initial \subset \forall\exists Plan{:}Final$."

This last equation assures us that the compound action *Plan* is applicable in a state verifying the preconditions *Initial* and that its application will produce a state verifying the goals *Final*.

5 Conclusions

This paper tries to present a unified approach to the domains of knowledge representation and reasoning from the viewpoint of terminological (description) logics. We have shown that TSLs are powerful enough to represent not only the domain knowledge in a particular application, but also the epistemic operators, actions and plans of a set of interacting agents. Because of our unifying approach, all these types of knowledge can be combined in an unrestricted fashion.

In order to support the reasoning involved, we have chosen a decidable terminological language, $\mathcal{ALC}_{reg+id(C)}$, for which we have developed the key inference algorithms. It should not be surprising that these algorithms are quite complex, because the underlying language has a high expressivity.

The resulting system, called $\mathcal{RegAL}$, is implemented in PROLOG and will be used in a very powerful knowledge-based systems development environment.

[9] This similarity should not be surprising since the planning problem is similar to the problem of program synthesis starting from input/output specifications.

References

[1] BAADER F. *Augmenting concept languages by the transitive closure : An alternative to terminological cycles.* IJCAI-91, pp. 446-451.

[2] BAADER F., HOLLUNDER B. *KRIS: Knowledge Representation and Inference System – System Description.* DFKI TM-90-03.

[3] BADEA LIVIU. *A unitary theory and architecture for knowledge representation and reasoning in Artificial Intelligence* (in Romanian) PhD thesis, Bucharest Polytechnic University, 1994.

[4] BRACHMAN R.J., SCHMOLZE J.G. *An Overview of the KL-ONE Knowledge Representation System.* Cognitive Science 9 (2) 1985.

[5] DONINI F.M., LENZERINI M., NARDI D., SCHAERF A., NUTT W. *Adding Epistemic Operators to Concept Languages.* Proceedings KR-92, Boston.

[6] HOLLUNDER B., NUTT W., SCHMIDT-SCHAUSS M. *Subsumption Algorithms for Concept Description Languages.* ECAI-90, pp. 384-353, Pitman, 1990.

[7] LAUX A. *Integrating a Modal Logic of Knowledge into Terminological Logics.* DFKI RR-92-56.

[8] PATEL-SCHNEIDER P.F. *Undecidability of Subsumption in NIKL.* Artificial Intelligence 39 (1989), pp. 263-272.

[9] SCHILD KLAUS. *Undecidability of Subsumption in U.* KIT Report, Technische Universität Berlin, October 1988.

[10] SCHILD KLAUS. *A correspondence theory for terminological logics: preliminary report.* IJCAI-91, pp. 466-471.

[11] SCHMIDT-SCHAUSS M., SMOLKA G. *Attributive concept descriptions with complements.* Artificial Intelligence 48 (1), pp. 1-26, 1991.

[12] SCHMIDT-SCHAUSS M. *Subsumption in KL-ONE is undecidable.* Proceedings KR-89, pp. 421-431.

[13] TUOMINEN H. *Translations from Epistemic into Dynamic Logic.* ECAI-88, pp. 586-588.

A Hybrid Integration of Rules and Descriptions, with F(rames)-Logic as an Underlying Formalism

Mira Balaban
Dept. of Mathematics and Computer Science
Ben-Gurion University of the Negev
P.O.B. 653, Beer-Sheva 84105, Israel
mira@black.bgu.ac.il
phone: (972)-7-461622 fax: (972)-7-472909

Abstract

Descriptions and *Rules* are different, complementary, essential forms of knowledge. Descriptions are analytic and closed; rules are contingent and open. Historically, descriptions and rules were developed along separate lines, by different communities ([10; 7; 5; 6]). The two forms can be integrated either by compiling one form within the other, or by constructing a hybrid framework. The hybrid solution keeps the modular independent status of each approach, but needs an underlying integration framework, in which a coherent compositional semantics can be defined. In this paper we use F-Logic ([8]) as an underlying framework for a hybrid construction of descriptions and rules. The hybrid framework, termed $\mathcal{DFL}$, is *modular*, and enjoys a *compositional semantics*. In $\mathcal{DFL}$, the knowledge base manages a database of explicit descriptions, by consulting two separate reasoners: DL – The *Description Languages* reasoner, and R – The *Rules* reasoner. The reasoners can operate under different semantical policies. Four different compositional semantics possible for the hybrid $\mathcal{DFL}$ framework are discussed and compared.

1 Architecture and Mode of Operation of a $\mathcal{DFL}$ KB

1.1 Architecture

A $\mathcal{DFL}$ knowledge base manages a database of explicit descriptions (a terminology and assertions). The knowledge base reasons about the given descriptions by consulting two separate reasoners:

1. DL – The *Description Languages* reasoner: A decidable reasoner, that reasons on the basis of the intended meaning of the terminological operators that form the descriptions.
2. R – The *Rules* reasoner, that reasons on the basis of given rules and some agreed upon semantical policy (e.g., perfect model).

This architecture is described in Figure 1.

While in query mode the $\mathcal{DFL}$ manager dispatches queries to the two reasoners. The reasoners make efforts to answer. If they succeed, they return an answer(s) to the manager. While reasoning, the reasoners may find out new descriptions, which they add directly to the database. The DL reasoner can operate under the so called *Open World Assumption (OWA)*, while the R reasoner can adopt the more conventional *Closed World Assumption (CWA)*.

1.2 Mode of Operation

We use the industrial plants example from the BACK manual ([7]), to demonstrate the operation.

The *Descriptions Database*, D, is given in tables 1 and 2, below. Table 1 includes the *terminology* descriptions, and Table 2 includes the *assertion* descriptions. We assume that the DL reasoner has a decidable oracle for answering queries about descriptions built with the operators: **and, all, some**, (primitive-)**not, domain, range, inv, trans, comp**[1]. The R reasoner consults the following rules:

$r1$. A terminological rule: A *radioactive_material* that is also a *waste*, is a *toxic_waste*.
$X \in toxic\_waste \longleftarrow$
$X \in \mathbf{and}(radioactive\_material, waste)$.

$r2$. A place L in which a product of a *dangerous_plant* is *buried_at*, is a *risky_place*.
$L \in risky\_place \longleftarrow$
$Y \in dangerous\_plant,\ (Y, X) \in produces,$
$(X, L) \in buried\_at$.

$r3$. If a *plant* is *located_at* a *risky_place*, it is a *dangerous_plant*.
$Y \in dangerous\_plant \longleftarrow$
$Y \in plant,\ (Y, X) \in located\_at,$
$X \in risky\_place$.

$r4$. A rule with negation: If a *plant uses_up* a material that cannot be shown to be a *dangerous_product*,

[1] After consulting Franz Baader and Klaus Schild, I realize that it is not clear whether this collection of operators, where **and** is used for concepts' and for roles' conjunction, yields a decidable DL. However, I prefer to leave the example as is.

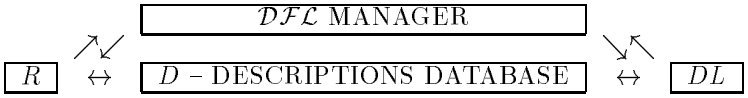


Figure 1: Architecture of a $\mathcal{DFL}$ KB

Kind	No.	Description in words	description
Primitive-concept	t1)	*product is_a top*	$product \leq top$
	t2)	*place is_a top*	$place \leq top$
	t3)	*type is_a top*	$type \leq top$
	t4)	*degree is_a top*	$degree \leq top$
	t5)	*danger_degree is_a degree*	$danger\_degree \leq degree$
	t6)	*energy is_a product*	$energy \leq product$
	t7)	*mechanical_product is_a product*	$mechanical\_product \leq product$
	t8)	*safe_product is_a product*	$safe\_product \leq product$
	t9)	*dangerous_product is_a product*	$dangerous\_product \leq product$
	t10)	*material is_a product*, not an *energy*	$material \leq \mathbf{and}(product, \mathbf{not}(energy))$
	t11)	*waste is_a product*, but not an *energy*	$waste \leq \mathbf{and}(product, \mathbf{not}(energy))$
	t12)	*radioactive_material is_a material*	$radioactive\_material \leq material$
	t13)	*safe_material is_a material*	$safe\_material \leq material$
	t14)	*toxic_waste is_a waste*	$toxic\_waste \leq waste$
	t15)	*chemical_waste is_a material* and *is_a waste*	$chemical\_waste \leq \mathbf{and}($ $material, waste)$
	t16)	*plant is_a top* that is *located_at* *place* and has a *type*	$plant \leq \mathbf{and}(top, \mathbf{all}(located\_at, place),$ $\mathbf{all}(is\_of\_type, type))$
Primitive-role	t17)	A *plant produces product*s, or, *produces* is a relation between *plant*s and *product*s	$produces \leq \mathbf{and}(\mathbf{domain}(plant),$ $\mathbf{range}(product)\ )$
	t18)	A *product* may be *buried_at* a *place*	$buried\_at \leq \mathbf{and}(\mathbf{domain}(product),$ $\mathbf{range}(place)\ )$
	t19)	*located_at* is a relation between objects and *place*s	$located\_at \leq \mathbf{range}(place)$
	t20)	A *product* may *directly_contain* *product*s	$directly\_contains \leq \mathbf{and}($ $\mathbf{domain}(product), \mathbf{range}(product))$
	t21)	*degree_of* is a relation between *product*s and *degree*s	$degree\_of \leq \mathbf{and}(\mathbf{domain}(product),$ $\mathbf{range}(degree)\ )$
Defined-concept	t22)	A *mechanical_plant* is a *plant* that *produces* only *mechanical_product*s	$mechanical\_plant \doteq \mathbf{and}(plant,$ $\mathbf{all}(produces, mechanical\_product))$
	t23)	A *dangerous_plant* is a *plant* that *produces* a *dangerous_product*	$dangerous\_plant \doteq \mathbf{and}(plant,$ $\mathbf{some}(produces, \mathbf{some}($ $degree\_of, danger\_degree)))$
	t24)	A *risky_place* is a *place* where a *toxic_waste* is *buried_at*	$risky\_place \doteq \mathbf{and}(place, \mathbf{some}($ $\mathbf{inv}(buried\_at), toxic\_waste))$
Defined-role	t25)	*produced_by* is inverse of *produces*	$produced\_by \doteq \mathbf{inv}(produces)$
	t26)	*contains* is the transitive closure role of *directly_contains*	$contains \doteq \mathbf{trans}(directly\_contains)$
	t27)	*uses_up* is the composition of *produces* and *contains*, restricted to *material*s as the range concept	$uses\_up \doteq \mathbf{and}($ $\mathbf{comp}(produces, contains),$ $\mathbf{range}(material)\ )$

Table 1: Tbox – Terminology

Kind	No.	Description in words	description
Concept-member	a1)	$plant_1$ is a *mechanical plant*	$plant_1 \in mechanica\_plant$
	a2)	$waste_2$ is a *wast* which is a *radioactive material*	$waste_2 \in \mathbf{and}(wast, radioactive\_material)$
	a3)	$product_1$ is a *chemical waste*	$product_1 \in chemical\_waste$
Role-member	a4)	$waste_1$ is a waste product of $plant_1$	$(plant_1\ , waste_1) \in produces$
	a5)	$plant_1$ *produces* a product $product_2$	$(plant_1\ , product_2) \in produces$
	a6)	$plant_2$ *produces* a product $product_1$	$(plant_2\ , product_1) \in produces$
	a7)	$waste_1$ is *buried_at* place l	$(waste_1\ , l) \in buried\_at$
	a8)	$waste_2$ is *buried_at* place *dump*	$(waste_2\ , dump) \in buried\_at$
	a9)	$plant_1$ is *located_at dump*	$(plant_1\ , dump) \in located\_at$
	a10)	$product_2$ *directly_contains* $product_1$	$(product_2, product_1) \in directly\_contains$
	a11)	The *degree_of ecology_mindedness* is *high*	$(ecology\_mindedness, high) \in degree\_of$

Table 2: Abox – Assertions

it can be assumed to be a $safe\_material$.
$X \in safe\_material \longleftarrow$
$Y \in plant,\ (Y, X) \in uses\_up,$
$\mathbf{not}(X \in dangerous\_product)$[2].

$r5$. A rule with negation: A *mechanical_product* that cannot be shown to be *buried_at* some place, can be assumed to be a $safe\_product$.
$X \in safe\_product \longleftarrow$
$X \in mechanical\_product,$
$\mathbf{not}(\ (X, L) \in buried\_at\ ).$

$r6$. A rule that extends the terminology: If *ecology_mindedness* is *high*, then *chemical_waste is_a dangerous_product*.
$chemical\_waste \leq dangerous\_product \longleftarrow$
$(ecology\_mindedness, high) \in degree\_of.$

$r7$. A rule that extends the terminology[3]: If *ecology_mindedness* is *high*, then if a plant produces a *dangerous_product*, then all of its products are considered *dangerous_product*s.
$\mathbf{some}(produces, dangerous\_product) \leq$
$\mathbf{all}(produces, dangerous\_product) \longleftarrow$

$(ecology\_mindedness, high) \in degree\_of.$

Reasoning: Queries are descriptions that might include variables, and the reasoners are expected to provide instantiations to the variables, in case of a positive answer. We assume that the $\mathcal{DFL}$ manager adds the answers of the reasoners to the database D.

1. Iteration between the reasoners is demonstrated in Table 3.
2. Reasoning with rules with negation is demonstrated in Table 4.
3. Reasoning with rules that extend the terminology is demonstrated in Table 5. Note that the order of rules' application matters. If $r6$ is consulted before $r4$, then $q9$ cannot be concluded. Hence, the reasoning process is time dependent.

2 Replacing the Standard Set-theoretic Semantics by the OO Semantics of F-Logic

In [3] it was argued that a description language L_P is a syntactic variant for a sorted F-Logic language, with sorts for concepts, roles, and objects. The syntactic abbreviations are summarized in Table 6. Note that the table **is not a translation** from DLs to F-Logic, but just a set of abbreviations. DL terms are, already, F-Logic terms; the table summarizes syntactic variations and agreed upon abbreviations.

The semantics of L_P, viewed as an F-Logic language, is defined over a partially ordered domain U, with a greatest and a least elements, where terms are mapped to elements of U, and the symbols *top* and *bottom* are assigned the greatest and least elements, respectively. The meaning of formulae is directly obtained from the meaning of their F-Logic variants. Formulae are interpreted by interpreting $\doteq$ and $\leq$ between concept terms as equality and the partial ordering, respectively; $\in$ between an object symbol and a concept term is interpreted as the membership binary relation over U; $\doteq$ and $\leq$ between role terms are interpreted as methods' equality and implication, respectively; $\in$ between a pair of object symbols and a role term is interpreted as a method's value assertion.

An important notion defined in [3] was that of a *corresponding theory* for L_P, which was defined as an F-Logic

[2] Note that this **not** operator is the F-Logic's negation as failure operator, and not the DL negation operator on primitive concepts (as in $t11$ and $t12$, for example). The distinction can be made on a syntactical basis.

[3] This rule is outside the consensus of DLs.

[4] A simpler, although less intuitive, abbreviation is obtained if an L_P formula $o \in c$ is replaced by the L_P^{FL} formula $o :: c$.

[5] The subscribed quantifier "$\forall_{ind}$" quantifies over the sort of individuals.

No	Query	Reasoner	Answer	Justification
$q1$)	$?X \in mechanical\_product$	DL	$X := waist_1$	$(t22), (a1), (a4)$
$q2$)	$?X \in mechanical\_product$	DL	$X := product_2$	$(t22), (a1), (a5)$
$q3$)	$?X \in toxic\_waste$	R	$X := waste_2$	$(r1), (a2)$
$q4$)	$?X \in risky\_place$	DL	$X := dump$	$(t24), (t18), (a8), (q3)$
$q5$)	$?X \in dangerous\_plant$	R	$X := plant_1$	$(r3), (t22), (a1), (a9), (q4)$
$q6$)	$?X \in risky\_place$	R	$X := l$	$(r2), (q5), (a4), (a7)$
$q7$)	$?(X, Y) \in contains$	DL	$X := product_2,\ Y := product_1$	$(t26), (a10)$
$q8$)	$?(X, Y) \in uses\_up$	DL	$X := plant_1,\ Y := product_1$	$(t27), (a5), (q7), (a3), (t15)$

Table 3: Iteration between the reasoners

No	Query	Reasoner	Answer	Justification
$q9$)	$?X \in safe\_material$	R	$X := product_1$	$(r4), (a1), (t22), (q8)$
$q10$)	$?X \in safe\_product$	R	$X := product_2$	$(r5), (q2)$

Table 4: Reasoning with negation

No	Query	Reasoner	Answer	Justification
$q11$)	$?X \in dangerous\_product$	R	$X := product_1$	$(r6), (a11), (a3)$
$q12$)	$?X \in \mathbf{some}(produces, dangerous\_product$	DL	$X := plant_2$	$(a6), (q11)$
$q13$)	$?X \in \mathbf{all}(produces, dangerous\_product$	R	$X := plant_2$	$(r7), (a11), (q12)$

Table 5: Reasoning with rules that extend the terminology

$\gamma \in L_P$	$\gamma^{FL} \in L_P^{FL}$
$c_n \doteq c$	$c_n \doteq c$
$c_1 \leq c_2$	$c_1 :: c_2$
$o \in c$	$o : c$[4]
$r_n \doteq r$	$\forall_{ind}$[5] $X, Y, (X[r_n \twoheadrightarrow \{Y\}] \equiv X[r \twoheadrightarrow \{Y\}])$
$r_1 \leq r_2$	$\forall_{ind} X, Y, (\ X[r_1 \twoheadrightarrow \{Y\}] \ \longrightarrow \ X[r_2 \twoheadrightarrow \{Y\}])$
$(o_1, o_2) \in r$	$o_1[r \twoheadrightarrow \{o_2\}]$

Table 6: Syntactic variations between a DL to a sorted F-Logic language

theory FL_P such that for every set of formulae Γ, and a formula γ in L_P:

$$\Gamma \models \gamma \quad iff \quad FL_P,\ \Gamma^{FL} \models \gamma^{FL}$$

(The first $\models$ is the description languages' logical implication relation, while the second is F-Logic's. The FL superscript stands for the F-Logic's notational variant.) A corresponding theory for $P = \{$**and, all, at-least1, and-role**$\}$ was given. A major result of [3] is that given a corresponding theory FL_P to L_P, F-Logic provides a full account to L_P, i.e., it correctly simulates logical implication and subsumption relations, while preserving the direct semantics. In particular, we had the following corrolary:

Corollary 2.1 *Let t_1, t_2 be terms of L_P, and FL_P an F-Logic's theory that corresponds to L_P. Then, t_1 is subsumbed by t_2 in a terminology Γ iff $FL_P,\ \Gamma^{FL} \models (t_1 \leq t_2)^{FL}$.*

3 Compositional Semantics

Let L_P be the language of descriptions in D, FL_P be its corresponding F-Logic's theory, and $RULES$ be the set of rules that R consults. Then, a straightforward *non-hybrid* approach to the meaning of KB, can define the models of KB as F-Logic models I such that: $I \models D \cup RULES \cup FL_P$. This approach is inappropriate from a knowledge representation point of view. The hybrid KB should provide specific services, that fit the mode of operation outlined above. The two main principles are: **Modularity** – The KB should be able to provide separate services, based on the DL or the R reasoners, and **Compositional behavior** – The semantics should be composed from the separate semantics of DL and R, which may operate along different reasoning policies. Two desirable properties are: **Query sensitivity**, and **Open behavior** – The KB should be tolerant to changes in the separate services. Clearly, neither of these principles is kept if the "global" F-Logic theory $D \cup RULES \cup FL_P$ defines the semantics of KB.

In this section we define four alternative compositional semantics H, F, $singleF$, and OF, that respect the two main principles of the hybrid construction. All four semantics are sets of syntactic objects, either in DL terms, or in terms of the underlying F-Logic formalism. They are constructed by *iteration* of the *separate* semantics $\mathcal{DL}$ and $\mathcal{R}$ of the DL and the R reasoners, respectively. $\mathcal{DL}$ and $\mathcal{R}$ are also sets of syntactic objects. This way the principles of *modularity* and *Compositionality* are kept. The general structure of the compositional semantics is visualized in Figure 2. $\mathcal{DFL}$ is formally defined as follows:

Define: $T(KB) \stackrel{def}{=} \mathcal{DL} \cup \mathcal{R}$

and $T^0(KB) = S^0$ –semantics dependent initial version.

$$T^{k+1}(KB) = T(T^k(KB)) \quad k \geq 0$$

$$T^{\omega}(KB) = \bigcup_{k=0}^{\infty} T^k(KB)$$

Then: $\mathcal{DFL}(KB) \stackrel{def}{=} T^{\omega}(KB)$

The four semantics differ in the separate $\mathcal{DL}$ and $\mathcal{R}$ being used, and in the sort of syntactic objects being processed. In the H semantics the syntactic objects are ground atoms of the underlying F-Logic formalism; in the F and the $singleF$ semantics the syntactic objects are ground descriptions; in the OF semantics the syntactic objects are not necessarily ground descriptions, and also rules of F-Logic ([4]). Hence, H is neither query sensitive nor open, F and $singleF$ are query sensitive but not open, and OF is both query sensitive and open. The expressivity relations between the four semantics are:

$$H \leq F = singleF \leq OF$$

with the reservations: 1)$F = singleF$ holds only when the R reasoner consults a set of definite positive rules, without negation, and 2) The OF semantics is defined only for an R reasoner that consults a set of definite positive rules. For detailed explanations and proofs consult [2].

4 Inference and Future Work

The compositional semantics suggests bottom-up inferencing, where a "by-product" of the inference process, is the derivation of mutiple conclusions, beyond the requested goal. The Normalize-Compare methods, the assertional reasoning of [11], and *magic-sets* approach are relevant. For a rules set with negation, bottom-up evaluation is close to computation of extensions in Reiter's default logic ([9]). Relationship to non-monotonic reasoning in DLs ([1]), and to F-Logic's non-monotonic inheritance mechanism are relevant.

Acknowledgements

I am grateful to Veronique Royer and Michael Kifer, who provided detailed comments on an earlier draft of this paper. I would like to thank also Mike Codish for introducing me to the non-ground s-semantics approach, and for endless fruitful discussions.

References

[1] F. Baader and B. Hollunder. How to prefer more specific defaults in terminological default logic. In *IJCAI-93*, pages 669–674, 1993.

[2] M. Balaban. The f(rames)-logic approach for description languages ii: A hybrid integrating of rules and descriptions. Technical Report FC 94-10, Department of Mathematics and Computer Science, Ben-Gurion University, Beer Sheva, Israel, 1994. ftp black.bgu.ac.il, cd ftp/pub/mira.

[3] M. Balaban. The f-logic approach for description languages. *Annals of Mathematics and Artificial Intelligence*, 1995. To appear.

[4] A. Bossi, M. Gabbrielli, G. Levi, and M. Martelli. The s-semantics approach: Theory and applications. *J. of Logic Programming*, 12, 1993.

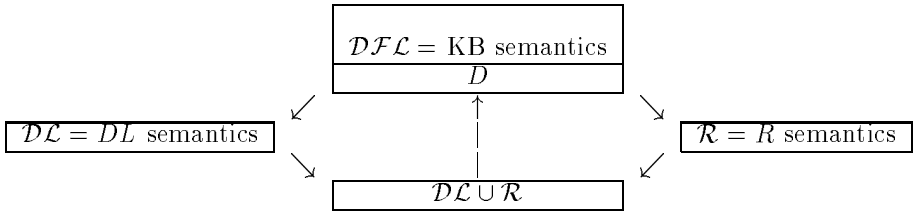


Figure 2: Structure of the compositional semantics

[5] F. Donini, M. Lenzerini, D. Nardi, A. Schaerf, and W. Nutt. Adding epistemic operators to concept languages. In *KR-92*, pages 342–353, 1992.

[6] P. Hanschke and . Hinkelmann. Combining terminological and rule-based reasoning for abstraction processes. In *German Conference on AI-92, Springer LNCS 671*, pages 0–0, 1992.

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

[8] M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *JACM*, 1995. To appear.

[9] R. Reiter. A logic for default reasoning. *J. of Artificial Intelligence*, 13(1–2):81–132, 1980.

[10] 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.

[11] V. Royer and J. Quantz. On intuitionistic query answering in description bases. In *CADE*, 1994.

CARIN: A Representation Language Combining Horn rules and Description Logics

Alon Y. Levy*
AT&T Bell Laboratories
AI Principles Research Dept.
600 Mountain Ave., Room 2C-406
Murray Hill, NJ, 07974
levy@research.att.com

Marie-Christine Rousset
L.R.I. U.R.A C.N.R.S
University of Paris-Sud
Building 490, 91405,
Orsay Cedex, France
mcr@lri.lri.fr

1 Introduction

Horn rule languages have formed the basis for many AI expert system applications. Though Horn rules are natural for representing many application domains, one of their significant limitations is that they are not expressive enough to model domains with a rich hierarchical structure. In contrast, description logics have been designed especially to model such domains, and have considered in detail the problem of reasoning about relationships between classes of objects in a domain. Naturally, many applications would significantly benefit from having the expressive power of both formalisms. This has been the driving force behind previous work on designing hybrid representation languages (e.g.,[Brachman *et al.*,1985; Donini *et al.*,1991; Hanschke and Hinkelman,1992; Forster and Novotny,1994; Abecker and Wache,1994]). Two prime examples of such applications are information gathering from multiple sources [Levy *et al.*,1995b; Knoblock and Levy,1995], where we require the expressive power of both formalisms in order to model *meta-data* about information sources, and the problem of modeling of complex physical devices [Rousset,1994; Fikes *et al.*,1991], in which we need to model both the structural aspects of a device (which are best modeled by a description logic), and its behavior aspects (which are naturally modeled using Horn rules). Furthermore, *constraint databases* and query languages with constraints have recently received significant attention in database research [Kanellakis *et al.*,1990; Brodsky *et al.*,1993; Levy *et al.*,1994; Levy and Sagiv,1995; Kanellakis,1995]. Since description logics can be viewed as a rich language for describing constraints, integrating them with Horn rules can result in a useful language for constraint databases.

In this paper we describe CARIN, a novel family of representation languages, that cleanly integrate Horn rules and description logics. CARIN combines the two formalisms by allowing knowledge bases to have, in addition to a terminological component, a set of *extended* Horn rules. Extended Horn rules are rules in which concept and role literals defined in the terminological component may appear in their antecedents. The semantics of CARIN are derived naturally from the semantics of its component languages, which are both subsets of first order logic with equality. We address the key problem in CARIN, which is the development of sound and complete inference procedures for the language. As we show, (and has been observed in [Donini *et al.*,1991]), applying standard Horn rule inference procedures (e.g., SLD resolution) to a set of extended rules is a sound procedure, but not complete.

A core problem in reasoning in CARIN knowledge bases is the *existential entailment* problem. Informally, this problem is to decide whether an arbitrary conjunctive query over concepts and roles defined in a description logic entails a union of such conjunctive queries. The main result of this paper is an existential entailment algorithm for queries over the $\mathcal{ALCNR}$ description logic [Buchheit *et al.*,1993], which is a fairly expressive logic. This algorithm entails several important results:

1. It provides a sound and complete inference procedure for non-recursive CARIN knowledge bases in which the description logic used is $\mathcal{ALCNR}$, or any of its subsets.
2. It can be used in conjunction with constrained logics [Bürckert,1994] to provide a backward chaining algorithm for arbitrary CARIN rules (though this procedure will not be complete).
3. It provides the first algorithm for subsumption of arbitrary conjunctive queries over $\mathcal{ALCNR}$ and its subsets. In particular, it can be used for optimizing queries by rewriting them to use precomputed *views*, extending the results of [Levy *et al.*,1995a].
4. We propose an approach for reasoning in a hybrid language such as CARIN, based on computing a set of *enhanced rules* that precompiles inferences made by the terminological component. The key advantage of this approach is that it enables us to use the known efficient algorithms for reasoning with Horn rules at run-time. Existential entailment plays a central role in proving the existence of the set of enhanced rules, and our algorithm can be used to compute such a set.

*Part of this work was done while the author was visiting Université de Paris-Sud.

In the following sections we describe the CARIN languages, the problems arising in making inferences in CARIN, the existential entailment algorithm and its corollaries.

2 The CARIN Languages

The motivation behind the design of CARIN is to obtain a language that has the expressive powers of both Horn rules and description logics, and integrates the two formalisms by enabling the Horn rules to exploit unary and binary predicates whose properties are defined in the description logic. Therefore, a CARIN knowledge base contains three parts: the terminology, rules and ground facts:

Terminology $\mathcal{T}$: A terminological component, defined in a description logic $\mathcal{L}$. In this paper we consider CARIN languages in which the terminological component is the language $\mathcal{ALCNR}$ or one of its subsets. Complex concepts and roles in $\mathcal{ALCNR}$ can be defined using the following syntax (A denotes a primitive concept, P_i's denote primitive roles, C and D represent arbitrary concepts and R denotes an arbitrary role):

$C, D \rightarrow$	$A \mid$	(primitive concept)
	$\top \mid \bot \mid$	(top, bottom)
	$C \sqcap D \mid C \sqcup D \mid$	(conjunction, disjunction)
	$\neg C \mid$	(complement)
	$\forall R.C \mid$	(universal quantification)
	$\exists R.C \mid$	(existential quantification)
	$(\geq\ n\ R) \mid (\leq\ n\ R)$	(number restrictions)
$R \rightarrow$	$P_1 \sqcap \ldots \sqcap P_m$	(role conjunction)

An $\mathcal{ALCNR}$ terminology contains a set of *inclusions* of the form $C \sqsubseteq D$. Intuitively, an inclusion states that every instance of of C must be an instance of D. The terminological component is also used to define a set of *named* complex concepts. Such a definition can be given by two inclusions, $C \sqsubseteq D$ and $D \sqsubseteq C$, (abbreviated $C := D$), where C is the name and D is an arbitrary description. Note that $\mathcal{ALCNR}$ allows for terminological cycles.

Rule base $\mathcal{R}$: a set of extended Horn rules, $\mathcal{R}$, i.e., logical sentences of the form: $(\forall \bar{X})\, p_1(\bar{X}_1) \wedge \ldots \wedge p_n(\bar{X}_n) \Rightarrow q(\bar{Y})$, where $\bar{X}$ includes all the variables in the rule, and $\bar{Y} \subseteq \bar{X}$. The predicates appearing in the antecedent of the rule may be either ordinary predicates or concepts and roles defined in $\mathcal{T}$.

Ground facts $\mathcal{A}$: a set of ground facts for the concepts, roles and predicates appearing in $\mathcal{T}$ and $\mathcal{R}$.

Semantics

The semantics of a CARIN knowledge base are derived in a natural way from the semantics of its components. An interpretation I contains a non-empty domain $\mathcal{O}^I$. It assigns an element a^I in $\mathcal{O}^I$ to every constant a in the KB, and relation p^I over $\mathcal{O}^I$ of arity n to every predicate p of arity n in the KB. In particular, it assigns a unary relation over $\mathcal{O}^I$ to every concept in $\mathcal{T}$, and a binary relation over $\mathcal{O}^I \times \mathcal{O}^I$ to every role in $\mathcal{T}$.

An interpretation I is a model of a CARIN knowledge base ($\mathcal{T}$,$\mathcal{A}$,$\mathcal{R}$) if it is a model of each of it components, defined as follows.

An interpretation I is a model of $\mathcal{T}$ if $C^I \subseteq D^I$ for every inclusion $C \sqsubseteq D$ in the terminology, and the extensions of the complex concepts satisfy the following equations ($\sharp\{\}$ denotes the cardinality of a set):

$\top^I = \mathcal{O}^I$, $\bot^I = \emptyset$, $(C \sqcap D)^I = C^I \cap D^I$,
$(C \sqcup D)^I = C^I \cup D^I$, $(\neg C)^I = \mathcal{O}^I \setminus C^I$,
$(\forall R.C)^I = \{d \in \mathcal{O}^I \mid \forall e : (d,e) \in R^I \rightarrow e \in C^I\}$
$(\exists R.C)^I = \{d \in \mathcal{O}^I \mid \exists e : (d,e) \in R^I \wedge e \in C^I\}$
$(\geq\ n\ R)^I = \{d \in \mathcal{O}^I \mid \sharp\{e \mid (d,e) \in R^I\} \geq n\}$
$(\leq\ n\ R)^I = \{d \in \mathcal{O}^I \mid \sharp\{e \mid (d,e) \in R^I\} \leq n\}$
$(P_1 \sqcap \ldots \sqcap P_m)^I = P_1^I \cap \ldots \cap P_m^I$

I is a model of a rule r if whenever α is a mapping from the variables of r to the domain $\mathcal{O}^I$, such that $\alpha(\bar{X}_i) \in p_i^I$ for every atom of the antecedent of r, then $\alpha(\bar{Y}) \in q^I$, where $q(\bar{Y})$ is the consequent of r. Finally, I is a model of a ground fact $p(\bar{a})$ if $\bar{a}^I \in p^I$.

It should be noted that CARIN does not allow concept and role atoms to appear in the consequents of the Horn rules because of the underlying assumption that the terminological component *completely* describes the hierarchical structure of classes in the domain, and therefore, the rules should not allow to make new inferences about that structure.

Reasoning in CARIN

The key issue that needs to be addressed in order to build systems based on CARIN is to develop algorithms for making sound and complete inferences from CARIN knowledge bases. As the following examples will show, combining Horn rules and description logics enables us to make more inferences than can be made from either of them alone. However, they will also show that using the known techniques for reasoning with Horn rules (or slight modifications of them) will not be complete. Consider the following terminology, $\mathcal{T}_1$:

eur-ass-company :=
    company $\sqcap$ $\exists$ASSOCIATE.european-company
am-ass-company :=
    company $\sqcap$ $\exists$ASSOCIATE.american-company
fellow-company :=
    company $\sqcap$ $\forall$ASSOCIATE.american-company
no-fellow-company :=
    company $\sqcap$ $\forall$ASSOCIATE.$\neg$ american-company
foreign-associate-company :=
    eur-ass-company $\sqcup$ am-ass-company

the rules $\mathcal{R}_1$:

r_1 : made-by(X,Y)$\wedge$no-fellow-company(Y) $\Rightarrow$
    price(X,usa,high)
r_2 : made-by(X,Y)$\wedge$company(Y)$\wedge$associate(Y,Z)$\wedge$
    american-company(Z)$\wedge$ monopoly(Y,X,usa) $\Rightarrow$
        price(X,usa,high)

and the ground facts

$\mathcal{A}_1$: { made-by(a,b), monopoly(b,a,usa),
    am-ass-company(b) }

$\mathcal{A}_2$: { `made-by(a,b)`,`monopoly(b,a,usa)`,
`foreign-associate-company(b)`}

First, note that $\mathcal{T}_1, \mathcal{A}_1, \mathcal{R}_1 \models$ `price(a,usa,high)`. The fact `am-ass-company(b)` and $\mathcal{T}_1$ entail that `b` has some `associate` that is an `american-company`. Therefore, even though no rule of $\mathcal{R}_1$ can be totally *instantiated* on $\mathcal{A}_1$, the missing conjuncts of r_2 are entailed by the KB.

Second, note that $\mathcal{T}_1, \mathcal{A}_2, \mathcal{R}_1 \models$ `price(a,usa,high)`. Here $\mathcal{T}_1 \cup \mathcal{A}_2$ does not entail the antecedent of any *single* rule in $\mathcal{R}_1$. However, we can make the inference by reasoning by cases. If `b` has no American associates, then `no-fellow-company(b)` will be entailed, and `price(a,usa,high)` will follow from r_1. If `b` has at least one American associate, then `price(a,usa,high)` will follow because the antecedent of rule r_2 will be entailed, as explained above.

To summarize, there are two aspects of traditional Horn rule reasoning inference mechanisms that make them inadequate for CARIN rules. The first is that they proceed by considering each rule in *isolation*, and the second is that for each rule they try to *instantiate* the antecedent in order to derive the consequent. The examples above show that both of these aspects will lead to incompleteness when applied to CARIN rules. The example with $\mathcal{A}_1$ showed that a KB may entail the antecedent of a single rule without being able to instantiate the antecedent in the KB. The example with $\mathcal{A}_2$ showed that a KB may entail the *union* of the antecedents of two rules without entailing either of them. It should be noted that these problems have been observed also in [Donini *et al.*,1991], where the rules considered were more restricted in that they did not allow role predicates to appear.

3 An Existential Entailment Algorithm

In this section we describe an algorithm for existential entailment over $\mathcal{ALCNR}$ terminologies. Formally, the existential entailment problem for a description logic $\mathcal{L}$ is the following. Let $\beta, Q_1, \ldots, Q_n$ be existential sentences of the form

$$(\exists\ \bar{Y})\, p_1(\bar{Y}_1) \wedge \ldots \wedge p_m(\bar{Y}_m).$$

The predicates in Q_i and β may be defined in a terminology $\mathcal{T}$, in the language $\mathcal{L}$. The existential entailment problem is to decide whether

$$\beta \cup \mathcal{T} \models Q_1 \vee \ldots \vee Q_n.$$

The existential entailment problem is important because it forms the core of several algorithms for making inferences from CARIN knowledge bases. Whatever control strategy we use for making inferences, the basic operation of checking whether an antecedent of a rule can be instantiated needs to be replaced by a more general procedure in which we check whether a set of ground facts $\mathcal{A}_0$ and the terminology $\mathcal{T}$ entail the union of the antecedents of a set of rules. Since the conjunction of set of ground facts $\mathcal{A}_0$ can be viewed as the sentence β, an existential entailment algorithm is appropriate for this purpose. Since the terminology *alone* cannot sanction any inferences about literals involving predicates that are not concepts or roles, the entailment of ordinary literals from $\mathcal{A} \cup \mathcal{T}$ is straightforward and involves matching them with ordinary facts of $\mathcal{A}$. Therefore, the general existential entailment problem can be restricted to the existential entailment problem where predicates appearing in the existential sentences are only role and concept predicates.

In particular, the existential entailment algorithm we describe for $\mathcal{ALCNR}$ entails the following results:

1. It provides a sound and complete inference procedure for CARIN knowledge bases with non-recursive rules. When the rules are non-recursive, they can be *unfolded*, and therefore deriving a fact of the form $p(\bar{a})$ amounts to checking whether $\mathcal{A} \cup \mathcal{T}$ entails the union of the rules in $\mathcal{R}$ (after unfolding) resulting from unifying their heads with $p(\bar{a})$. [1]

2. The existential entailment algorithm can be combined with constrained SLD-resolution [Bürckert,1994] to provide a goal directed backward chaining algorithm on CARIN rules.[2] Informally, at every step of the backward chaining, the existential entailment algorithm is used to check if the residual constraints from the rules are entailed by the knowledge base. Note however that this procedure will not be complete on recursive rules. This procedure has the advantage of being goal-directed, and therefore likely to yield better performance in many cases, compared to forward reasoning.

3. The existential entailment algorithm provides the first algorithm for deciding subsumption of *arbitrary* conjunctive queries over $\mathcal{ALCNR}$, including arbitrary inclusions. Previous work has only considered restricted forms of conjunctive queries. In particular, it provides a method for deciding query *containment* (analogous to [Sagiv,1988]), which is an important tool in optimizing rule bases. Recently, it was shown in [Levy *et al.*,1995a] that containment is the core to solving the problem of rewriting queries using materialized views. Therefore, our existential entailment algorithm can be used to extend this result to queries over description logics.

4. The algorithm provides the core of an inference method based on computing a set of *enhanced rules*, described in the Section 3.2.

[1] It should also be noted that sound and complete inferences can be made from non-recursive CARIN knowledge bases *without* unfolding all the rules ahead of time. Informally, a ground fact $p(\bar{a})$ is entailed from a CARIN knowledge base if it is entailed from *every* canonical interpretation we construct in the existential entailment algorithm.

[2] This observation is due to Hans-Jürgen Bürckert.

3.1 Existential Entailment Algorithm for $\mathcal{ALCNR}$

Our algorithm is based on extending the technique of *constraint systems* used in [Buchheit *et al.*,1993] for a satisfiability checking algorithm for $\mathcal{ALCNR}$ knowledge bases. Informally, the satisfiability checking algorithm begins with an initial constraint system, constructed from a given knowledge base, which is a finite representation of all its models. It then applies to the system a set of *propagation rules* that generate a set of *completions*. Each completion is a refinement of the initial constraint system, in which some implicit constraints have been made explicit, and some non-deterministic choices were made (as a result of disjunction in the KB). Each completion represents a subset of the models of the KB. Some completions contain explicit *clashes*, (e.g., they state that an object belongs both to a class and to its complement). Such completions are clearly unsatisfiable, and therefore do not represent any model of the KB. The key property shown in [Buchheit *et al.*,1993] is that if a completion is *clash-free* then it is satisfiable, and therefore, if one clash-free completion is found, the KB is satisfiable. They show that a completion is satisfiable by exhibiting a *canonical interpretation* of the completion. The key to guaranteeing this property is the condition used to terminate the application of the propagation rules.

Our algorithm begins in a similar fashion, by constructing a constraint system for the KB containing the terminology $\mathcal{T}$ and the sentence β. It then generates the set of all clash-free completions, and for each completion S it checks whether Q is satisfied in the canonical interpretation of S. Since we are using canonical interpretations to test entailment and not satisfiability, we need to be sure that if Q is satisfied in the canonical interpretation of a completion S, then it will be satisfied in *all* models of S. This stronger property is guaranteed by modifying the condition for terminating the application of the propagation rules. In what follows, we explain constraint systems in detail, the propagation rules and the novel termination condition.

Constraint Systems

In our discussion, we denote the set of variables and constants that appear in Q or β by $\mathcal{V}$. From this point on, we refer to elements of $\mathcal{V}$ as *individuals*.[3] In describing constraint systems, we introduce an alphabet of variable symbols $\mathcal{W}$, with a well-founded total ordering $\prec_{\mathcal{W}}$. The alphabet $\mathcal{W}$ is disjoint from $\mathcal{V}$. We denote elements of $\mathcal{W}$ by the letters u, v, w, x, y, z. The term *object* refers to elements of $\mathcal{V} \cup \mathcal{W}$ (i.e., either variables or individuals). Objects are denoted by the letters s, t. Elements in $\mathcal{V}$ are denoted by the letters a, b.

A constraint system is a non-empty set of constraints of the form

$$s : C, sPt, \forall x.x : C, s \neq t,$$

where C is a concept and P is a role name.

Suppose S is a constraint system and $R = P_1 \sqcap \ldots \sqcap P_k$ $(k \geq 1)$ is a role. We say that an object t is an R-successor of an object s if $sP_1t, \ldots, sP_kt \in S$. We say that t is a *direct successor* of s if it is the R-successor for some role R. The *successor* relationship denotes the transitive closure of the direct-successor relation. The *direct-predecessor* and the *predecessor* relations are the inverses of direct-successor and successor, respectively. We say that s and t are *separated* in S if $s \neq t \in S$. Finally, we denote by $S[x/s]$ the constraint system obtained from S by replacing each occurrence of the variable x by the object s.

For a variable s in a constraint system S, we define the function $\sigma(S, s) := \{C \mid s : C \in S\}$. Two variables in $s, t \in S$ are said to be concept-equivalent if $\sigma(S, s) = \sigma(S, t)$. Intuitively, two variables are concept-equivalent in S if, as far as the constraints in S are concerned, they have the same properties. The function σ will be used for the termination condition.

An interpretation I satisfies the constraint $x : C$ if $\alpha^{\mathcal{I}}(x) \in C^{\mathcal{I}}$, the constraint xRy if $(\alpha^{\mathcal{I}}(x), \alpha^{\mathcal{I}}(y)) \in R^{\mathcal{I}}$, the constraint $x \neq y$ if $\alpha^{\mathcal{I}}(x) \neq \alpha^{\mathcal{I}}(y)$, and the constraint $\forall x.x : C$ if $C^{\mathcal{I}} = \Delta^{\mathcal{I}}$. An interpretation is a model of a constraint system S if it satisfies every constraint in S.

Given a terminology and a sentence β, the initial constraint system S_β is constructed as follows. If $p_i(\bar{Y}_i)$ is of the form $C(a)$, we put $a : C$ in S_β. If $p_i(\bar{Y}_i)$ is of the form $R(a, b)$ we put $aP_1b, \ldots, aP_nb$ in S_β, if $R = P_1 \sqcap \ldots \sqcap P_n$. If $C \sqsubseteq D$ is an inclusion in the terminology, we add $\forall x.x : \neg C \sqcup D$ to S_β. If a and b are two individuals in β that are *not* existentially quantified, then we add $a \neq b$ to S_β. It is easy to see that the models of S_β are precisely the models of β.

Completions are obtained by applying the propagation rules below. We apply the rules according to the following strategy:

- we do not apply a rule to a variable x if we can apply a rule to an individual or to a variable for which $y \prec_{\mathcal{W}} x$,
- we apply rules that generate new variables (i.e., rules 4 or 5) only if no other rules are applicable.

Because of rules 4 and 5, the propagation may not terminate. Therefore, these rules use the notion of a *blocked* variable to guarantee termination. We will elaborate on this notion shortly.

1. $S \rightarrow_{\sqcap} \{s : C_1, s : C_2\} \cup S$
   if 1. $s : C_1 \sqcap C_2$ is in S,
   2. $s : C_1$ and $s : C_2$ are not both in S.
2. $S \rightarrow_{\sqcup} \{s : D\} \cup S$
   if 1. $s : C_1 \sqcup C_2$ is in S,
   2. neither $s : C_1$ nor $s : C_2$ are in S,
   3. $D = C_1$ or $D = C_2$.
3. $S \rightarrow_{\forall} \{t : C\} \cup S$
   if 1. $s : \forall R.C$ is in S,
   2. t is an R-successor of s,
   3. $t : C$ is not in S.
4. $S \rightarrow_{\exists} \{sP_1y, \ldots, sP_ky, y : C\} \cup S$
   1. $s : \exists R.C$ is in S,

[3] Note that $\mathcal{V}$ may contain constants used in the KB.

2. $R = P_1 \sqcap \ldots \sqcap P_k$,
3. y is a new variable,
4. there is no t such that t is an R-successor of s in S and $t : C$ is in S,
5. if s is not blocked.

5. $S \rightarrow_{\geq} \{sP_1y_i, \ldots, sP_ky_i \mid i \in 1..n\} \cup \{y_i \neq y_j \mid i, j \in 1..n, i \neq j\} \cup S$
if 1. $s : (\geq\ n\ R)$ is in S,
2. $R = P_1 \sqcap \ldots \sqcap P_k$,
3. $y_1, \ldots, y_n$ are new variables,
4. there do not exist n pairwise separated R-successors of s in S,
5. if s is not blocked.

6. $S \rightarrow_{\leq} S[y/t]$
if 1. $s : (\leq\ n\ R)$ is in S,
2. s has more than n R-successors in S,
3. y, t are two R-successors of s which are not separated

7. $S \rightarrow_{\forall x} \{s : C\} \cup S$
if 1. $\forall x.x : C$ is in S,
2. s appears in S,
3. $s : C$ is not in S.

A constraint system is said to be a *completion* when no propagation rule applies to it. It contains a *clash*, and is therefore unsatisfiable, if it contains

- $\{s : \bot\}$, or
- $\{s : A, s : \neg A\}$, or
- $\{s : (\leq\ n\ R)\} \cup \{sP_1t_i, \ldots, sP_kt_i \mid 1..n+1\} \cup \{t_i \neq t_j \mid i, j \in 1..n+1, i \neq j\}$ where $R = P_1 \sqcap \ldots \sqcap P_k$.

Propagation Termination and Canonical Interpretations

In [Buchheit *et al.*,1993], a variable v was said to be blocked if there exists another variable w, such that $w \prec_{\mathcal{W}} v$, and $\sigma(S, v) = \sigma(S, w)$. The variable w was said to be the *witness* of v. Given this termination condition, the canonical interpretation I_S of a completion S was defined as follows. Its domain is the set of objects in S, and each individual in S is mapped to itself (i.e., $\alpha^{I_S}(s) = s$). The extensions of concepts and roles are defined as follows:

- $s \in C^{I_S}$ if and only if $s : C \in S$.
- $(s, t) \in R^{I_S}$ if and only if
  1. $sRt \in S$, or
  2. s is blocked, w is its witness, and wRt is in S.

Unfortunately, this termination condition will not suffice in our context. Consider the terminology consisting of the single inclusion $C \sqsubseteq \exists R.C$, and $\beta = C(a)$. Beginning with the constraint $a : C$, the propagation rules would generate the constraints $a : \exists R.C$, aRv_1, $v_1 : C$. The same would be repeated for v_1, i.e., the constraints $v_1 : \exists R.C$, v_1Rv_2, $v_2 : C$ would be generated. The variable v_2 is blocked (because of the witness v_1), and therefore the propagation would terminate. In the canonical interpretation of the completion, the extension of C would be $\{a, v_1, v_2\}$, and of R would be $\{(a, v_1), (v_1, v_2), (v_2, v_2)\}$. As a consequence, the canonical interpretation would satisfy the sentence $\exists X_1, X_2 R(X_1, X_2) \wedge R(X_2, X_2)$ which is clearly not entailed by the completion.

Intuitively, the problem is that in the process of creating the canonical interpretations, we are forced to assign successors to the blocked variables (in our example $(v_2, v_2) \in R^{I_S}$). Introducing these successors may cause the canonical interpretation to contain connectivity patterns among variables that do not exist in *every* model of the completion. We now describe our revised termination condition that guarantees that such spurious patterns are not introduced. Our condition will refine the previous one by considering the σ values not only of a variable, but also of its neighbors.

The *n-tree* of a variable v is the tree that includes the variable v and its successors, whose distance from v is at most n direct-successor arcs. We denote the set of variables in the n-tree of v by $V_n(v)$. Two variables $v, w \in S$ are said to be n-tree equivalent in S if there is an isomorphism $\psi : V_n(v) \rightarrow V_n(w)$, that conserves the direct-successor relation, and such that for every $s \in V_n(v)$, $\sigma(S, \psi(s)) = \sigma(S, s)$. Intuitively, two variables are n-tree equivalent if the trees of depth n of which they are roots are isomorphic. We denote by D_Q the number of literals in Q. A variable v will now be said to be blocked if the following holds:

1. v is the leaf of a D_Q-tree rooted in a variable v_1,
2. there exists a predecessor v_2 of v_1 such that v_1 and v_2 are D_Q-tree equivalent, and v_1 is not a node in the D_Q-tree of v_2.

In constructing canonical interpretations of our completions, the witness of the variable v is now $\psi(v)$, where ψ is the isomorphism between the n-trees of v_1 and v_2. To ensure that we can correctly detect blocked variables, we apply the propagation rules in a breadth first fashion. That is, we only apply a rule to a variable whose distance from an individual in S is n direct-successor arcs, if no rule is applicable to a variable with distance less than n.

The following theorem establishes the main property of our termination condition:

Theorem 3.1: *Let S be a completion of S_β, and let I_S be its canonical interpretation. If $I_S \models Q$, then $S \models Q$.*

Proof sketch: The proof of the theorem is based on showing that if ρ is a mapping from the variables of Q to the domain of I_S, that shows that Q is satisfied in I_S, then there exists a mapping ρ', that shows that Q is satisfied, such that ρ' does *not* use any arc emanating from a blocked variable in I_S. Since ρ' uses only constraints that are *explicit* in S, these constraints will be satisfied in every model of S. □

In our example, the propagation rules would also generate the variables $v_3, \ldots, v_6$ similar to v_1 and v_2. The variable v_6 would be deemed blocked, because it is the leaf of a 2-tree rooted in v_4, and the 2-trees rooted in v_1 and v_4 are isomorphic. The witness of v_6 would be v_3, and therefore, the canonical interpretation of the completion would have (v_6, v_4) in R^{I_S}. Now, the sentence

$\exists X_1, X_2 R(X_1, X_2) \wedge R(X_2, X_2)$ is not satisfied in the canonical interpretation.

The following theorem establishes the correctness of our algorithm.

Theorem 3.2:

- *Applying the propagation rules to S_β will terminate.*
- *The entailment $\mathcal{T} \cup \beta \models Q$ holds if and only if Q is satisfied in the canonical interpretation of every clash-free completion of S_β.*

Proof sketch: The first part follows from the fact that there are only a finite number of possible n-trees. For the second part, the *only if* direction follows from showing that the canonical interpretation of a completion of S_β is a model of $\mathcal{T} \cup \beta$. The *if* direction follows from Theorem 3.1 and showing that every model of $\mathcal{T} \cup \beta$ is a model of *some* clash-free completion of S_β. □

Following the analysis of [Buchheit *et al.*,1993], it can be shown that the number of completions is at most doubly exponential in the size of $\mathcal{T}$,β and Q. Therefore, since checking satisfiability of Q in each completion can be done in exponential time, the time complexity of our algorithm is doubly exponential. It should be noted that the complement of our problem (i.e., non-entailment) is a generalization of the satisfiability problem considered in [Buchheit *et al.*,1993], and which was given a non-deterministic exponential time algorithm there.

3.2 Reasoning Based on Rule Rewriting

In the full version of the paper we propose a novel approach to the problem of reasoning in a hybrid language such as CARIN that is aimed to exploit existing algorithms for optimizing reasoning on Horn rules. Our approach is to find a new set of *enhanced* rules $\mathcal{R}^e$ such that applying a conventional Horn-rule inference engines to $\mathcal{R}^e$ will yield a *complete* inference procedure.

Formally, given a terminology $\mathcal{T}$ and a set of rules $\mathcal{R}$[4], our goal is to find a set of rules $\mathcal{R}^e$, that will satisfy the following two conditions:

- Soundness: $\mathcal{R} \cup \mathcal{T} \models \mathcal{R}_e$,
- Completeness: for *any* set of ground facts $\mathcal{A}$, if p is a not a concept or a role, and $\bar{a}$ is a tuple of objects, then

$$\mathcal{R} \cup \mathcal{T} \cup \mathcal{A} \models p(\bar{a}) \Longrightarrow \mathcal{R}^e \cup \mathcal{A} \models p(\bar{a}).$$

Intuitively, the set of enhanced rules *precompiles* the inferences sanctioned by the terminology. The assumption is that the precompilation is done off line, and therefore, it can be a more expensive operation. In our example, we would add the following rules to $\mathcal{R}_1$:

r_3 : `made-by(X,Y)`∧`am-ass-company(Y)`∧
`monopoly(Y,X,usa)`⇒ `price(X,usa,high)`
r_4 : `made-by(x,y)`∧`monopoly(y,x,usa)`∧
`foreign-associate-company(b)` ⇒ `price(x,usa,high)`

[4]Note that we are looking for $\mathcal{R}^e$ which are *independent* of the ground facts in the KB.

In the full version we define the *terminology boundedness* condition[5], and show that if a description logic satisfies it then the existential entailment algorithm can be used to prove the existence of a finite set of enhanced rules. We also show that the language consisting of $\sqcap, \forall\ R.C$ and $(\leq\ n\ R), (\geq\ n\ R)$ (but no terminological cycles) satisfies this condition.

4 Discussion

We presented CARIN, a family of languages that cleanly integrate two formalisms, Horn rules and description logics, both of which have been useful in practical applications. We addressed the key issue of designing sound and complete inference procedures for CARIN knowledge bases. Our main result is an existential entailment algorithm for $\mathcal{ALCNR}$ terminologies, which entails a sound and complete inference procedure for non-recursive CARIN knowledge bases, using $\mathcal{ALCNR}$ as the terminological component. A corollary of our work is the algorithm for subsumption of arbitrary conjunctive queries over $\mathcal{ALCNR}$ and its subsets, which is important for designing query languages for description logics, and for optimizing rule bases.

Several works considered the integration of Horn rules and description logics. Some works (e.g., (AL-log [Donini *et al.*,1991], TaxLog [Abecker and Wache,1994]) had the goal of using a description logic as a rich *typing* language on the variables already appearing in the rules (which could also be recursive). In those works, only unary predicates from the description logics are allowed in the rules. Other works (e.g., [Brachman *et al.*,1985; Hanschke and Hinkelman,1992]) considered a more tight integration of the two formalisms, which is more in the spirit of CARIN. In particular, role predicates were allowed in the rules. For example, KRYPTON [Brachman *et al.*,1985] combined a rule language (more expressive than Horn rules) with a much less expressive description logic than $\mathcal{ALCNR}$. In these works, the reasoning engine was modified by either adding resolution steps to consider the inferences sanctioned by the terminological component, or by modifying the unification operation underlying the resolution engine. These approaches are either incomplete or guarantee only refutation completeness, relying on a full first-order logic theorem prover.

A different approach to integrating rules and description logics is to add rules as an additional constructor in description logics (e.g., (CLASSIC [Brachman *et al.*,1991], BACK [Petalson,1991], LOOM [MacGregor,1988]). These works allowed only rules of a restricted form: $C(x) \Rightarrow D(x)$, where C and D are concepts. Furthermore, the rules are generally not integrated in subsumption inferences but they are just used to derive additional knowledge about concept instances. The work of Forster and Novotny [Forster and Novotny,1994] is in the same spirit.

[5]This condition guarantees that the entailment of a fact $p(a)$ depends on a number of objects in the KB that is bounded by a constant which depends only on the terminology $\mathcal{T}$.

Though full Horn rules were considered, the inferences drawn from the terminological part had priority over inferences drawn from the rule component. MacGregor [MacGregor,1994] and Yen [Yen,1990] describe algorithms for determining rule-specificity and classification of arbitrary predicates in LOOM, which are an instance of the existential entailment problem described here. However, since subsumption in LOOM is undecidable, their algorithms are not complete either.

Our work is the first step in studying the inference problem for CARIN knowledge bases. We are exploring several directions of research. First, we are interested in finding more efficient inference algorithms for CARIN knowledge bases that contain terminologies in languages more restricted than $\mathcal{ALCNR}$. Second, we are extending our methods to deal with recursive rules. An important direction we are pursuing is finding additional languages for which a set of enhanced rules can be computed. In particular, we are considering conditions that are less restrictive than the terminology boundedness condition but still guarantee the existence of enhanced rules.

References

[Abecker and Wache, 1994] Andreas Abecker and Holger Wache. A layer architecture for the integration of rules, inheritance, and constraints. In *Proceedings of ICLP 94 post conference workshop on the Integration of Declarative Paradigms*, 1994.

[Brachman *et al.*, 1985] Ronald J. Brachman, Victoria P. Gilbert, and Hector J. Levesque. An essential hybrid reasoning system: Knowledge and symbol level accounts of krypton. In *Proceedings IJCAI-85.*

[Brachman *et al.*, 1991] R. J. Brachman, A. Borgida, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Resnick. Living with CLASSIC: When and how to use a KL-ONE-like language. In John Sowa, editor, *Principles of Semantic Networks.*

[Brodsky *et al.*, 1993] Alexander Brodsky, Joxan Jaffar, and Michael J. Maher. Toward practical constraint databases. In *Proceedings of VLDB-93.*

[Buchheit *et al.*, 1993] Martin Buchheit, Francesco M. Donini, and Andrea Schaerf. Decidable reasoning in terminological knowledge representation systems. *Journal of Artificial Intelligence Research*, 1993.

[Bürckert, 1994] Hans-Jürgen Bürckert. A resolution principle for constrained logics. *Artificial Intelligence*, 66:235–271, 1994.

[Donini *et al.*, 1991] Francesco Donini, M. Lenzerini, D. Nardi, and A. Schaerf. A hybrid system integrating datalog and concept languages. In *Working notes of the AAAI Fall Symposium on Principles of Hybrid Reasoning*, 1991.

[Fikes *et al.*, 1991] Richard Fikes, Thomas Gruber, Yumi Iwasaki, Alon Levy, and Pandurang Nayak. How things work project overview. Knowledge Systems Laboratory, Stanford University, technical report No. KSL 91-70, 1991.

[Forster and Novotny, 1994] Peter Forster and Bernd Novotny. Integration of rule inferences into a terminological component. In *Proceedings of the Eighth International Symposium on Methodologies for Intelligent Systems*, pages 31–42, Charlotte, North Carolina, USA, October 1994. Oak Ridge National Laboratory.

[Hanschke and Hinkelman, 1992] Philipp Hanschke and Knut Hinkelman. Combining terminological and rule-based reasoning for abstraction processes. DFKI Research Report, 1992.

[Kanellakis *et al.*, 1990] P.C. Kanellakis, G.M. Kuper, and P.Z. Revesz. Constraint query languages. In *Proceedings of the 9th ACM Symp. on Principles of Database Systems*, pages 299–313, 1990.

[Kanellakis, 1995] P.C. Kanellakis. Tutorial: Constraint programming and query languages. In *Proceedings of PODS-95.*

[Knoblock and Levy, 1995] Craig A. Knoblock and Alon Y. Levy, editors. *Working Notes of the AAAI Spring Symposium on Information Gathering from Heterogeneous Distributed Environments.* American Association for Artificial Intelligence., 1995.

[Levy and Sagiv, 1995] Alon Y. Levy and Yehoshua Sagiv. Semantic query optimization in datalog programs. In *Proceedings PODS-95.*

[Levy *et al.*, 1994] Alon Y. Levy, Inderpal Singh Mumick, and Yehoshua Sagiv. Query optimization by predicate move-around. In *Proceedings of VLDB-94.*

[Levy *et al.*, 1995a] Alon Y. Levy, Alberto O. Mendelzon, Yehoshua Sagiv, and Divesh Srivastava. Answering queries using views, 1995. In *Proceedings PODS-95.*

[Levy *et al.*, 1995b] Alon Y. Levy, Divesh Srivastava, and Thomas Kirk. Data model and query evaluation in global information systems. *Journal of Intelligent Information Systems, Special Issue on Networked Information Discovery and Retrieval.*, 5 (2), September 1995.

[MacGregor, 1988] R. M. MacGregor. A deductive pattern matcher. In *Proceedings of AAAI-88.*

[MacGregor, 1994] Robert M. MacGregor. A description classifier for the predicate calculus. In *Proceedings of AAAI-94.*

[Petalson, 1991] C. Petalson. The BACK system : an overview. In *Proceedings of the SIGART bulletin*, volume 2(3), pages 114–119, 1991.

[Rousset, 1994] Marie-Christine Rousset. Knowledge Formal Specifications for Formal Verification: a Proposal based on the Integration of Different Logical Formalisms. In *Proceedings of ECAI-94.*

[Sagiv, 1988] Yehoshua Sagiv. Optimizing datalog programs. In Jack Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 659–698. Morgan Kaufmann, Los Altos, CA, 1988.

[Yen, 1990] John Yen. A principled approach to reasoning about the specificity of rules. In *Proceedings of AAAI-90*.

Objects, Classes, Specialization and Subsumption

Amedeo Napoli
CRIN CNRS–INRIA Lorraine
B.P. 239, 54506 Vandœuvre-lès-Nancy Cedex, France
(Email: napoli@loria.fr)

Position paper for the 1995 International Workshop on Description Logics,
Roma, June 1995.

1 Introduction

In this position paper, we are mainly concerned with a family of object-based representation systems, in which knowledge is organized in hierarchies of generic and specific objects representing real-world concepts and individuals. The characteristics of these systems rely on the integration of characteristics associated with object-oriented systems [Masini *et al.*,1991] and description logics [Nebel,1990], including specialization, inheritance, subsumption and classification. We formally define a relation, named *object-based subsumption*, that is used to organize generic and specific objects, according to their definitions. We then study the interrelations existing between object-based subsumption and specialization.

In the last part of the position paper, we introduce a discussion about teaching knowledge representation, taking as a basis object-based representation systems and description logics. Lastly, we end this position paper with a recapitulation of our past work in the field of description logics.

2 Hierarchical Organizations of Knowledge

2.1 Object-Based Representation Systems and Description Logics

In *object-based representation systems*, real-world knowledge is represented by generic and specific objects. A generic object, or *class*, has an identity and is composed of a set of *properties* describing the behavioral and static characteristics of a real-world concept. Thus, a class has a state and a behavior, and it can be used to generate a set of *instances*, often called *objects*, describing real-world individuals (instances of real-world concepts). Classes are organized in a hierarchy $\mathcal{H} = (\mathcal{C}, \preceq, \omega)$, where $\mathcal{C}$ is a set of classes, $\preceq$ is a partial ordering and ω is the root of the hierarchy $\mathcal{H}$, i.e. a special class that is supposed to be the greatest element of $\mathcal{C}$ for $\preceq$. Moreover, the hierarchical organization of classes involves *knowledge* or *property sharing*, supported by the transitivity of $\preceq$ and depending on the semantics of $\preceq$. Knowledge sharing can be monotonic or nonmonotonic. It is usually used to exhibit implicit knowledge for information retrieval purposes and for default reasoning, i.e. to infer the existence and the values of properties.

In object-based representation systems, the partial ordering $\preceq$ usually is the *specialization* relation, denoted by $\preceq_{Inh}$. The specialization relation is used top-down: new classes depend on existing (and more general) classes, i.e. new properties are added and/or the values of *shared* or *inherited* properties may be overridden in new classes [Ducournau and Habib,1991]. More precisely, if $\mathtt{B} \preceq_{Inh} \mathtt{A}$, then $\mathtt{P_A} \subseteq \mathtt{P_B}$, where $\mathtt{P_A}$ denotes the set of properties attached to the class `A`. *Inheritance* is the mechanism managing property sharing and supporting default reasoning: given a class `C` and a property `p`, inheritance is used to find in $\mathcal{H}$ the value of `p` for `C`. A hierarchy $\mathcal{H} = (\mathcal{C}, \preceq_{Inh}, \omega)$ with an associated inheritance mechanism is called an *inheritance hierarchy*.

In description logics, concepts, roles and individuals represent real-world concepts, their properties and their instances. The *subsumption* relation is used to organize concepts and roles in hierarchies: a concept `C` *subsumes* a concept `D` if and only if `C` is necessarily more general than `D`, i.e. the set of individuals denoted by `D`, or extension of `D`, is necessarily included in the extension of `C`.

Below we give a list of the main differences existing between description logics and object-based representation systems:

- In description logics, the definition of a new concept is a monotonic operation, while the specialization of a class can be nonmonotonic because of overridding. Moreover, if concepts have a natural set-based interpretation, this is useless to associate the same kind of set-based semantics to classes. Only trivial inferences can be drawn according to such a semantics: `i` is in the extension of the class `C` if and only if `i` has been defined as an instance of `C`.
- Roles can be viewed as necessary and sufficient conditions for defined concepts –this is the basis of classification– while class properties are only necessary conditions, such as this is the case for primitive concepts.
- In description logics, the classification process has an inductive character: the hierarchy is built up from concepts and individuals, and inferences are a

consequence of the classifier work. In object-based systems, specialization is used top-down and the emphasis is put on the fact that a class B inherits the properties of a more general class A. Inheritance is a deductive mechanism and it guides the specialization process.

Specialization and subsumption are both used to organize knowledge in hierarchies and to derive properties and values from already defined properties. While inheritance is mainly a mechanism for knowledge sharing, subsumption is used to derive complex inferences that do not only rely on the transitivity of the relation. However, subsumption does not support the nonmonotonic character of default reasoning.

Related works are the following: a comparison of class-based formalisms [Calvanese *et al.*,1994], a combination of classification and default reasoning [Padgham and Nebel,1993], a study on the similarities and differences between concept definitions and types in programming language [Borgida,1992], and an adaptation of description logics techniques to conceptual schema design [Bergamaschi and Sartori,1992]. In the next section, we give a formal basis for combining specialization and subsumption in the context of object-based representation systems.

2.2 The Object-Based Subsumption

In this section, we first divide class properties into definitional and functional properties, and then we introduce the object-based subsumption relation, i.e. subsumption for definitional properties.

A class C can be considered as a pair (N_C, P_C), where N_C denotes the name of the class and P_C denotes the set of properties associated with C. Usually, $P_C = (S_C, A_C, M_C)$, where S_C is a set of system properties used for system manipulation and recording information about C, A_C is a set of *attributes* and M_C is a set of *methods*, i.e. pieces of code that can be activated by means of message sending to compute or to update values [Masini *et al.*,1991]. Moreover, two different properties in a class have two different names.

In the following, A_C denotes the so-called definitional properties or definitional attributes of the class C, and we will only be interested in this set of properties.

Definition 1 (Definitional Properties)
A definitional attribute a *in a class* C*, denoted by* a(C)*, defines a part of the state of* C*, Data can be associated with an attribute* a*: a value* value(a(C)) *and specifications about that value. The specifications can be a type declaration* range(a(C))*, type restrictions such as a list of instances* enumeration(a(C)) *or a numerical interval* interval(a(C))*, and a cardinality restriction* cardinality(a(C))*.*

We will suppose that our object-based context is composed of a set of classes organized in an inheritance hierarchy $\mathcal{H} = (\mathcal{C}, \preceq_{Inh}, \omega)$, and of the set of the extensions of these classes. The *O-subsumption* relation is defined as follows:

Definition 2 (Object-Based Subsumption)
A class C O-subsumes *a class* D*, denoted by* $D \preceq_{Obj} C$*, if and only if* $\forall\ a \in A_C, \exists\ a \in A_D$*:* $a(D) \preceq_{Att} a(C)$*.*

$a(D) \preceq_{Att} a(C)$*, if and only if the following statements hold:*

$range(a(D)) \preceq_{Inh} range(a(C))$*: the type of* a *can be a primitive type, e.g.* Number *or* String*, or a class in* $\mathcal{H}$*.*

$enumeration(a(D)) = \{o_1, .., o_m\} \subseteq enumeration(a(C)) = \{o_1, .., o_n\}$*.*

$interval(a(D)) = [min(a(D)), max(a(D))] \subseteq interval(a(C)) = [min(a(C)), max(a(C))]$*, where* min *and* max *are numbers.*

$cardinality(a(D)) = [cmin(a(D)), cmax(a(D))] \subseteq cardinality(a(C)) = [cmin(a(C)), cmax(a(C))]$*, where* cmin *and* cmax *are integers.*

$value(a(D)) = value(a(C))$ *if* $value(a(C))$ *is an atomic value, and* $value(a(C)) \subseteq value(a(D))$ *if* $value(a(C))$ *is a list of atomic values.*

The O-subsumption relation is based on inclusion of sets of definitional attributes. As this is the case for defined concepts in description logics, a set of definitional attributes can be considered as a set of necessary and sufficient conditions for a class D to be subsumed by a class C.

Given a class C and the set A_C of the definitional attributes of C, it is possible to associate with C a set AP_C of attribute pairs $\{a_i, a_j\}_{i,j=1,..,n,i\neq j}$, where n is the cardinality of A_C. Attributes in A_C being non comparable for $\preceq_{Att}$, the set AP_C is an antichain[1] of the partial ordering $\preceq_{Att}$. Then, $\preceq_{Obj}$ can be considered as a suborder of the distributive lattice of antichains associated with $\preceq_{Att}$ and denoted by $\mathbf{2}^{\preceq_{Att}}$ [Aigner,1979]. Therefore, $\preceq_{Obj}$ is a partial ordering that can be used to organize classes in a hierarchy. Furthermore, meet ($\wedge$) and join ($\vee$) operations can be defined on the definitional parts of classes (and can be used for classification if needed).

2.3 O-Subsumption and Specialization

The links between O-subsumption and specialization are described in the two following propositions.

Proposition 1 (Monotonic Specialization $\rightarrow$ O-Subsumption)
If specialization $\preceq_{Inh}$ *is a monotonic operation in the inheritance hierarchy* $\mathcal{H} = (\mathcal{C}, \preceq_{Inh}, \omega)$*, i.e. only definitional attributes are considered and values cannot be overridden, then* $\preceq_{Inh}$ *is a O-subsumption relation on the set* $\mathcal{A}$ *of definitional attributes associated with classes in* $\mathcal{C}$*.*

Proposition 2 (O-Subsumption $\rightarrow$ Inheritance Hierarchy)
Given a set of classes $\mathcal{C}$ *and the set* $\mathcal{A}$ *of definitional attributes associated with classes in* $\mathcal{C}$*, a O-subsumption relation gives rise to an inheritance hierarchy: if* $D \preceq_{Inh} C$*, then* $A_C \subseteq A_D$ *and* D *shares with* C *the value of every definitional attribute in* A_C*.*

[1] A subset of a partially ordered set is called an *antichain* if every two elements of the subset are incomparable.

The two above propositions can be summarized as follows: O-subsumption and monotonic specialization have the same behavior on definitional attributes. However, if nonmonotonic specialization is allowed or functional properties are considered, then these two behaviors are no more equivalent. New definitions of $\preceq_{Att}$ and $\preceq_{Obj}$ taking into account nonmonotonic specialization and functional properties must be given.

3 Concluding Remarks

3.1 Teaching Knowledge Representation

Description logics have become logical systems of main interest and they are taking more and more importance in the field of knowledge representation. Thus, they have their place in a course on knowledge representation. Examples on the definitions, the capabilities and the use of description logics can be found in [Nebel,1990] and [Brachman *et al.*,1991] for example. However, no effort has been made until now to collect a set of introductory and more sophisticated examples that could be used as a basis for lectures on description logics and objet-based representation systems as well. Many subjects can be covered, e.g. subsumption algorithms, classification-based reasoning, incompleteness results, strengths and limitations of descriptions logics, etc. Then, it could be interesting to discuss the possibility of sharing examples and terminological knowledge bases during the 1995 International Workshop on Description Logics, relying on the fact that most of the researchers working on description logics will attend the workshop.

3.2 Past Works in the Field of Description Logics

The past works of the author in the field of description logics are theoretical and practical works on the use of classification-based reasoning techniques (as defined in description logics) in the context of object-oriented systems. The practical aspects of these works are mainly concerned with the design of knowledge-based systems for organic synthesis planning [Napoli,1991] [Napoli,1992b]. The theoretical aspects are mainly concerned with the duality programming – representation in object-based contexts. The author has organized a workshop held at the IJCAI Conference in Chambéry (1993) on this topic [OBRS,1993] [Dekker and Napoli,1994]. The integration of reasoning and representation techniques in object-oriented contexts has been (briefly) examined in [Napoli,1994], and the integration of structural part-whole relations and procedural knowledge in description logics is (briefly) discussed in [Napoli,1992a].

References

[Aigner, 1979] M. Aigner. *Combinatorial Theory*. A Series of Comprehensive Studies in Mathematics 234. Springer-Verlag, Berlin, 1979.

[Bergamaschi and Sartori, 1992] S. Bergamaschi and C. Sartori. On Taxonomic Reasoning in Conceptual Design. ACM *Transactions on Database Systems*, 17(3):385–422, 1992.

[Borgida, 1992] A. Borgida. From Type Systems to Knowledge Representation: Natural Semantics Specifications for Description Logics. *International Journal of Intelligent and Cooperative Information Systems*, 1(1):93–126, 1992.

[Brachman *et al.*, 1991] 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 Language. In J. Sowa, editor, *Principles of Semantic Networks: Explorations in the Representation of Knowledge*, pages 401–456. Morgan Kaufmann Publishers, Inc., San Mateo, California, 1991.

[Calvanese *et al.*, 1994] D. Calvanese, M. Lenzerini, and D. Nardi. A Unified Framework for Class-Based Representation Formalisms. In *Proceedings of the Fourth International Conference on Principles of Knowledge Representation and Reasoning (KR'94), Bonn, Germany*, pages 109–120, 1994.

[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.

[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.

[Masini *et al.*, 1991] G. Masini, A. Napoli, D. Colnet, D. Léonard, and K. Tombre. *Object-Oriented Languages*. Academic Press, London, 1991.

[Napoli, 1991] A. Napoli. Subsumption and Classification-Based Reasoning in Object-Based Representations. In C. Peltason, K. von Luck, and C. Kindermann, editors, *Terminological Logic Users Workshop – Proceedings – KIT-REPORT 95*, pages 124–133. Department of Computer Science, Technische Universitaet Berlin, 1991.

[Napoli, 1992a] A. Napoli. Representation of Partial Order Relations and Procedures in Object-Based Representation Systems. In *AAAI Fall Symposium Series – Issues in Description Logics: Users Meets Developers, Cambridge, Massachusetts*, pages 61–63, 1992.

[Napoli, 1992b] A. Napoli. Subsumption and Classification-Based Reasoning in Object-Based Representations. In *Proceedings of the 10th ECAI, Vienna*, pages 425–429, 1992.

[Napoli, 1994] A. Napoli. Studies about the Integration of Classification-Based Reasoning and Object-Oriented Programming. In F. Baader, M. Lenzerini, W. Nutt, and P.F. Patel-Schneider, editors, *Working Notes of the 1994 Description Logics Workshop*, pages 60–62. DFKI Saarbruecken, 1994.

[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.

Combining Description Logic Systems with Information Management Systems

Lin Padgham
Department of Computer Science
RMIT
Melbourne 3001, Victoria, Australia
email: linpa@cs.rmit.edu.au

1 Introduction

Description logic systems are powerful tools for representing knowledge and making logical inferences. They are extremely valuable as a part of many applications. However there are many applications where the services of a D.L. system would be very valuable, but are not in themselves sufficient. An accepted view of D.L. systems is that they provide a particular functionality to an application, but that they will be used in a larger context. Despite this view there is little if any literature on how to interface a D.L. system to another system dealing with the same application information, and the attendant problems of communication between the systems, consistency maintenance, etc.

We have done some preliminary work [KAL94,PLK95] in interfacing CLASSIC to an object centred information/database system, LINCKS [PL94,Pad88] which we have been working on. Our belief is that such a combination would provide valuable increased functionality to many potential applications of both systems. We expect that there are also applications which could not be served by either system alone, but where the combination of these two systems would provide an adequate base.

In the following sections we describe the approach we have taken to coupling the two systems, some advantages of this coupling and some issues which have arisen. We assume that readers are familiar with CLASSIC, but give a short overview of LINCKS in order to provide the context for the rest of the paper.

2 LINCKS

LINCKS is an object centred multi-user database system developed for complex information system applications where editing and browsing of information in the database is of paramount importance. The focus is on sharing of small information chunks which combine to make up complex information objects used by different users for different purposes. The information chunks are semi-structured containing a part which is well structured to facilitate addition of A.I. processing within the system, and a part which is unstructured and suitable for management by the user.

The system contains information regarding the history of objects and actions within the system. Past versions of objects can be accessed and reactivated.

LINCKS allows a system user or application developer to interactively define varying views on the underlying database objects. This is particularly useful in applications where the same information is to be used for different purposes or by people in differing roles, thus requiring a different composition and display.

Although all objects are built up of small pieces, the user interface presents an integrated and holistic view of complex objects, allowing editing over the entire object using an emacs like editor.

The LINCKS system is developed at the University of Linköping and is available on the Internet with a Gnu license. It has currently been ftp'd by over 1400 sites. Further information on LINCKS is available on the world wide web (http://www.ida.liu.se/labs/iislab/).

3 Architectural Overview

The combined system is under the primary control of LINCKS, where CLASSIC services are available via a menu. Only a subset of CLASSIC is actually available from LINCKS in our initial prototype. LINCKS communicates with CLASSIC by sending lisp commands through an asynchronous Unix pipe, and recieving back lists of commands that should be executed by LINCKS in order to update the database with the knowledge which has been derived by CLASSIC.

Hooks within CLASSIC are used to collect information regarding updates to the CLASSIC knowledge base, which are then propagated to the LINCKS information base. It is not currently possible to make changes directly in the LINCKS knowledge base, and propagate these to CLASSIC. Changes of information which is relevant to CLASSIC can be made only by editing the lisp expression which defines the information for CLASSIC (and thus also for LINCKS). However it would be desirable to add this capability.

4 Advantages of Combination

There are a number of obvious advantages in combining a description logic system with a more general information

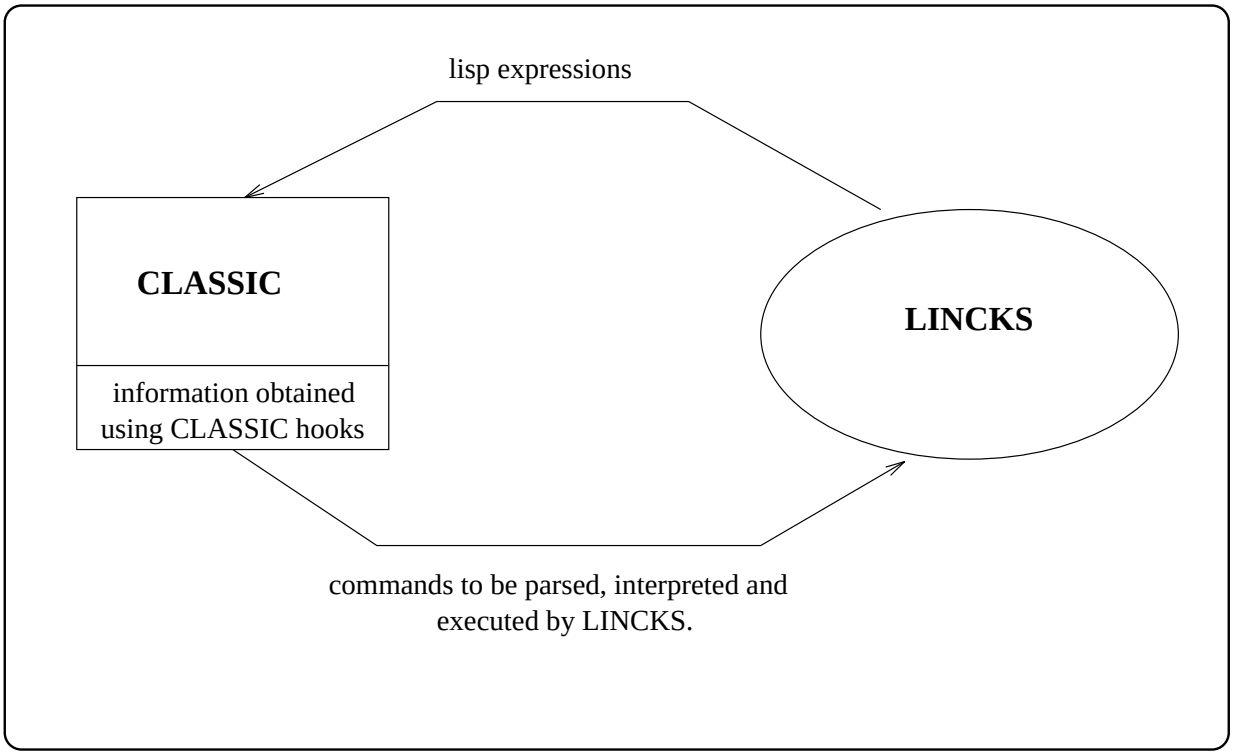


Figure 1: Architecture of LINCKS-CLASSIC coupling

management system. These include:

- Ability to apply classification to only a select portion of an object, leaving other attributes outside of the classification process.
- Ability to apply classification and associated inference only to certain individuals in the database.
- Control over when inferencing should occur, thus avoiding the necessity for constant D.L. processing. It becomes possible to run the D.L. as a background process on demand.
- Increased functionality based on the associated system. In our case this (potentially) includes:
  - access to previous versions of individuals and types
  - multiple views of objects, using differing levels of granularity
  - advanced editing support for composite objects
  - hypertext style interface allowing easy access to related objects, and browsing of the concept hierarchy
  - template style information regarding possible/necessary roles for an individual

5 Issues in the Combined System

In the prototype which we have currently developed, not all of the above advantages are actually realised. The editing capabilities of LINCKS are not able to be used as there are no methods implemented for recognising when editing of individuals is relevant to CLASSIC, and ensuring that the changes are propagated to the CLASSIC knowledge base.

A lack of exact match between the expressivity in CLASSIC, and the language which LINCKS uses for defining types (or views), results in an inability to exactly match corresponding definitions. For example CLASSIC has the ability to state that the number of role fillers is in a given range, whereas LINCKS can only state a fixed number or an infinite number. LINCKS has the ability to place an ordering on role fillers which is not available in CLASSIC.

There are a number of difficulties involved in ensuring consistency between the information in the two systems. A lack of ability to protect objects from editing in LINCKS means that constraints in CLASSIC (e.g. that concept definitions cannot be modified) cannot easily be propagated to LINCKS.

6 Conclusions

There are a number of gains available in coupling a D.L. system to a more general information management system in order to develop applications. However there is a considerable amount of work to be done to make the system models and expressivity compatible.

The hooks provided by CLASSIC for supplying information to external systems were extremely valuable in communicating to LINCKS the information necessary to

modify the LINCKS information in accord with CLASSIC inferences. We will be modifying LINCKS in order to improve the connection with CLASSIC, and expect that we will in future be able to report in more detail on advantages of this approach, and techniques used to overcome difficulties.

References

Kal94 Kalmelid, S., *Combining a Terminological Reasoner with an Object-Centered Database*, Final Undergraduate Thesis, Department of Computer and Information Science, Linköping University, 1994.

Pad88 NODE : A Database for Use by Intelligent Systems, *Proceedings of Third International Symposium on Methodologies for Intelligent Systems*, North-Holland, 1988, pp. 190–199.

PLK95 Lin Padgham, Patrick Lambrix and Stefan Kalmelid, Integrating a Description Logic System and an Object-Centered Database System, *Proceedings of the International Symposium on Knowledge Retrieval, Use and Storage for Efficiency, KRUSE '95*, August 1995.

PL94 Lin Padgham and Jonas Löwgren, A User Interface Management Approach for Object Oriented Database Applications, *Journal of Systems and Software*, Vol. 27 No. 3, Dec., 1994; pages 183-205

A Semantics-Driven Query Optimizer for OODBs

Jean Paul Ballerini[*], Domenico Beneventano[◇], Sonia Bergamaschi[◇], Claudio Sartori[◦], Maurizio Vincini[◇]

1 DLs techniques for OODB query optimization

Semantic query optimization uses problem-specific knowledge (e.g. integrity constraints) for transforming a query into an *equivalent* one (i.e., with the same answer set) that may be answered more efficiently. The optimizer is applicable to the class conjunctive queries is based on two fundamental ingredients. The first one is the *ODL* description logics proposed as a common formalism to express: class descriptions, a relevant set of *integrity constraints* rules (IC rules), queries as ODL types. The second one are DLs (Description Logics) inference techniques exploited to evaluate the logical implications expressed by IC rules and thus to produce the *semantic expansion* of a given query. The optimizer tentatively applies all the possible transformations and delays the choice of beneficial transformation till the end. Some preliminar ideas on filtering activities on the *semantically expanded query* are reported. A prototype semantic query optimizer (*ODB-QOptimizer*) for object-oriented database systems (OODBs) is described.
Let us briefly explain the main ingredients of our approach for semantic query optimization, firstly proposed in [4; 3].

- ODL *description logics for generalized database schema*

ODL (Object Description Logics) was proposed in [5] and extends the expressiveness of implemented description logics languages in order to represent the semantics of complex object data models (*CODMs*), recently proposed in the areas of deductive databases [1] and object oriented databases [6]. For instance, a distinction between *values* and *objects* with identity and, thus, between *value types* and *class types* (briefly called classes) has been introduced (i.e. concepts are partitioned into the ones representing objects sets and the one representing values sets); type constructors, such as *tuple*, *set* and *sequence* often supported by CODMs are directly expressible. Most importantly, the representation and management of *cyclic classes*, i.e., classes which directly or indirectly make reference to themselves, as CODMs usually require, are supported.

The extension to ODL introduced in the present work allows the declarative formulation of a relevant set of database integrity constraints. Actual database schemata are, in fact, given in terms of base classes (i.e. primitive concepts) while further knowledge is expressed with integrity constraints, which should guarantee data consistency. In general, constraints go beyond DLs expressivity and are expressed in various fashions, depending on the database data model: e.g. subsets of first order logic, inclusion dependencies and predicates on row values, procedural methods in OO environments. In particular, we extend ODL by: *quantified path types* and *integrity constraints* rules. The former extension has been introduced to deal easily and powerfully with nested structures. Paths, which are essentially sequences of attributes, represent the central ingredient of OODB query languages to navigate through the aggregation hierarchies of classes and types of a schema. In particular, we provide *quantified* paths to navigate through set types. The allowed quantifications are existential and universal and they can appear more than once in the same path. By means of path types (a path type is a type associating a path to an ODL type) we can express a relevant set of integrity constraints.

Viewing a database schema as a set of ODL *inclusion statements* [7] allows the declarative formulation of another relevant set of integrity constraints, expressing *if then rules* whose antecedent and consequent are ODL *virtual* types (i.e. defined concepts). For example, it is possible to express correlations between structural properties of the same class or sufficient conditions for populating subclasses of a given class. A *generalized database schema* can be thus defined as a set of inclusion statements between general ODL types. Inclusion statements

[*]Centro Interdipartimentale di Ricerca in Filosofia del Diritto e Informatica Giuridica, Università di Bologna, Via Galliera 3, I-40121 Bologna, Italy.

[◦] Dipartimento di Elettronica, Informatica e Sistemistica, Università di Bologna, Viale Risorgimento 2, I-40136 Bologna, Italy.

[◇] Dipartimento di Scienze dell'ingegneria, Università di Modena, Via G. Campi 213/B, I-41100 Modena, Italy.

e_mail: { jpballerini, dbeneventano, sbergamaschi, csartori}@deis.unibo.it

constitute a generalization of the description logics perspective which perfectly fits the usual database viewpoint.

In order to illustrate our approach, let us introduce the Company domain example which describes part of the organizational structure of a company (see table 1 for its ODL description).

Classes descriptions are given with a syntax richer than usual DLs (see type constructors and paths). As an example, the class Storage of table 1 describes departments with a category property and stocking materials all of quantity between 10 - 300. Let us briefly describe in words IC rules of table 1:

R_1 says that, having a risk greater equal to 10, is a sufficient condition for a material to be an smaterial. Note that it is different from giving for smaterial the following description: $\texttt{Material} \sqcap \triangle[\texttt{risk}: 10 \div \infty]$, as this last description expresses a necessary condition whereas R_2 expresses a sufficient condition). R_2, R_3 and R_4 are rules on storages: R_2 says that, stocking at least an smaterial is a sufficient condition for a storage to be an sstorage; R_3 says that, if a storage is managed by a manager with a level from 6 to 12, it is of category "A2". R_4 constrains storages of category "A2", stocking at least a material having a risk greater equal to 15, to be managed by a tmanager.

Let us give as an example on paths and of the relevance of multiple quantifications to increase expressivity. The type $S = (\triangle.\texttt{stock}.\exists.\texttt{item}.\triangle.\texttt{feature}.\exists$: *"F1"*) individuates all the domain objects (objects are denoted by the $\triangle$ operator) such that *at least* one of the stocked materials has a feature *"F1"*. On the other hand, the type $S' = (\triangle.\texttt{stock}.\forall.\texttt{item}.\triangle.\texttt{feature}.\exists$: *"F1"*) individuates *all* the domains objects such that *all* the stocked materials has a feature *"F1"*.

- *Queries and DLs inference techniques*

A relevant set of queries, corresponding to the so called single-operand queries [8] , can be expressed as *virtual* ODL types. Note that, this assumption is an important restriction as it excludes operations comparable to relational joins. However, it is important to realize that join is a crucial operation in relational databases largely because of the limitation of the relational model of data. In OODB environment we have an *implicit join* of the classes on a class-composition hierarchy rooted at the target class of the query. Subsumption computation, incoherence detection and canonical form generation can be used to produce the *semantic expansion* $EXP(Q)$ of a query Q. It is a transformed query which incorporates any possible restriction which is not present in the original query but is *logically implied* by the query and by the overall schema (classes + IC rules). Following the approach of [9; 10] for semantic query optimization, we perform the semantic expansion of the types included at each nesting level in the query description. Type expansion 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* [5]. Following this approach, the semantic expansion of a query coincides with the *most specialized query* among all the semantically equivalent ones[4]. Moreover, during the transformation, we compute and substitute in the query, at each step, the *most specialized generalization classes* satisfying the query. Let us give two simple query optimization examples related to Company schema.

Q: "Select storages storing *at least* a material having a risk ≥ 15"

$$Q = \texttt{Storage} \sqcap (\texttt{stock}.\exists.\texttt{item}: \texttt{Material} \sqcap (\triangle.\texttt{risk}: 15 \div \infty))$$

The applicable rules are R_1, R_2, leading to the semantic expansion of Q:

$$EXP(Q) = \texttt{SStorage} \sqcap (\texttt{stock}.\exists.\texttt{item}: \texttt{SMaterial} \sqcap (\triangle.\texttt{risk}: 15 \div \infty))$$

In this way, we can optimize the query by obtaining the *most specialized generalization* of the classes involved in the query SStorage and SMaterial.

2 Detection of eliminable factors

The above example show an effective query optimization, independent from any specific cost model and due to the substitution of the classes mentioned in the query with their specializations. In fact, by substituting Storage with SStorage and Material with SMaterial, we reduced the number of classes to be accessed to process the query as we implicitely eliminate the need to access the classes Material Storage.

A same general query optimization can be obtained if we reduce the number of classes on a class-composition hierarchy rooted at the query. In the literature [9; 10] this problem is generally addressed as *constraint removal*, i.e., removal of implied restrictions from the query.

Our approach is to detect the *eliminable* factors of a query, that is, the factors logically implied by the query, by exploiting subsumption. Furthermore, as we will show in the following this activity must be performed on the semantic expansion of a query to be conducted to a maximum degree.

As an example, let us consider the query:
Q_1: "Select storages of category *"A2"* managed by a TManager":

$$Q_1 = \underbrace{\texttt{Storage} \sqcap (\triangle.\texttt{managed-by}: \texttt{TManager})}_{S_1} \sqcap \underbrace{(\triangle.\texttt{category}: \textit{"A2"})}_{S_2}$$

By rule R_3 and by the description of TManager, we have, with respect to the overall schema, S_2 subsumes S_1 and thus, S_2 can be eliminated from Q_1.

$$
\begin{array}{lll}
\textit{Classes} & : & \left\{ \begin{array}{lll}
\texttt{Manager} & \sqsubseteq & \triangle[\texttt{name: String}, \texttt{salary: } 40 \div \infty, \texttt{level: } 5 \div 15] \\
\texttt{Department} & \sqsubseteq & \triangle[\texttt{name: String}, \texttt{managed-by: Manager}] \\
\texttt{TManager} & \sqsubseteq & \texttt{Manager} \sqcap \triangle[\texttt{level: } 8 \div 12] \\
\texttt{Material} & \sqsubseteq & \triangle[\texttt{name: String}, \texttt{risk: Int}, \texttt{feature: } \forall\{\texttt{String}\}] \\
\texttt{SMaterial} & \sqsubseteq & \texttt{Material} \\
\texttt{Storage} & \sqsubseteq & \texttt{Department} \sqcap \triangle[\texttt{category: String}, \\
 & & \qquad \texttt{stock: } \forall\{[\texttt{item: Material}, \texttt{qty: } 10 \div 300]\}] \\
\texttt{SStorage} & \sqsubseteq & \texttt{Storage}\exists.
\end{array} \right. \\
\textit{IC rules} & : & \left\{ \begin{array}{lll}
R_1 & : & \texttt{Material} \sqcap (\triangle.\texttt{risk: } 10 \div \infty) \sqsubseteq \texttt{SMaterial} \\
R_2 & : & \texttt{Storage} \sqcap (\triangle.\texttt{stock}.\exists.\texttt{item: SMaterial}) \sqsubseteq \texttt{SStorage} \\
R_3 & : & \texttt{Storage} \sqcap (\triangle.\texttt{managed-by}.\triangle.\texttt{level: } 6 \div 12) \sqsubseteq (\triangle.\texttt{category: } \textit{"A2"}) \\
R_4 & : & \texttt{Storage} \sqcap (\triangle.\texttt{category: } \textit{"A2"}) \\
 & & \sqcap(\triangle.\texttt{stock}.\exists.\texttt{item}.\triangle.\texttt{risk: } 10 \div \infty)) \sqsubseteq (\triangle.\texttt{managed-by: TManager})
\end{array} \right.
\end{array}
$$

Table 1: The Company domain schema

If we test the redundancy of a factor against the query, we exploit only sufficient conditions to eliminate a factor. We can have necessary and sufficient conditions if we test the redundancy of a factor against the semantic expansion of the query. This is shown by the following example:

$$
Q_2 = \underbrace{\texttt{Storage} \sqcap (\triangle.\texttt{stock}.\exists.\texttt{item}.\triangle.\texttt{risk: } 10 \div \infty)}_{S_3} \sqcap \underbrace{(\triangle.\texttt{managed-by: TManager})}_{S_4}
$$

In this case, S_4 does not subsume S_3, thus, S_4 is not redundant w.r.t. the query Q_2. However, if we consider the semantic expansion of the query Q_2:

$$
\begin{array}{lll}
EXP(Q_2) & = & \left. \begin{array}{l} (\triangle.\texttt{managed-by: TManager}) \end{array} \right\} S_4 \\
 & & \left. \begin{array}{l} \sqcap\ \texttt{SStorage} \sqcap (\triangle.\texttt{category: } \textit{"A2"}) \\ \sqcap\ (\triangle.\texttt{stock}.\exists.\texttt{item: SMaterial} \\ \sqcap\ (\triangle.\texttt{risk: } 10 \div \infty)) \end{array} \right\} S_5
\end{array}
$$

S_4 is redundant w.r.t. $EXP(Q_2)$, as, S_4 subsumes S_5. The query Q_2 can thus be reduced to S_5, avoiding the access to the class `TManager`.

3 ODB-QOptimizer : a semantics driven query optimizer

ODB-QOptimizer is composed of two modules: the ODL-DESIGNER prototype [2] and the GES prototype.

ODL-DESIGNER was developed in Sicstus Prolog at CIOC-CNR, Bologna, as part of the LOGIDATA$^+$ project [2] and implements DLs reasoning techniques for advanced database management systems handling complex objects data models. It is an active tool which supports automatic building of type taxonomies for complex object database systems, preserving coherence and minimality with respect to inheritance. It implements the theoretical framework of [5]. Due to the generality of the ODL formalism, this tool can be used as a kernel component for schema acquisition for object-oriented database system.

GES was developed at the Department of Engineering Sciences of the University of Modena to produce the semantic expansion of a query Q. It has been developed in C, to achieve better performance.

The current version of ODB-QOptimizer is a quite rudimentary quick prototype, including extensions to ODL-DESIGNER to support if then rules. Its main limits include a keyboard-based user interface and a loose coupling between the different programming environments Prolog and C.

As you can see in figure 1, having a pre-existing ODL database schema *previously pre-processed* (see below) by ODL-DESIGNER we can insert a query Q and begin its expansion by activating ODB-QOptimizer first. ODL-DESIGNER is activated and applies DLs techniques to the query Q (virtually added to the ODL schema) giving as output the detection of the query Q as incoherent or, if it is coherent, the list of the subsumers types (including the rules antecedents) of all the types contained at each nesting level in the query (i.e. the file GES-IN.TXT). If the query is coherent, ODB-QOptimizer activates the GES component which receives GES-IN.TXT as input file; GES expands the query ($EXP^i(Q)$) by adding all the consequents of the applicable rules as conjunction factors and sends a message to ODB-QOptimizer about its execution. Two cases are distinguished: at least one rule was applied (the semantic expansion is probably not completed) or no rule was applied (the semantic expansion has been completed). In the former case, ODB-QOptimizer activates ODL-DESIGNER again sending to it the GES ouput file GES-OUT.PL which contains the performed expansion $EXP^i(Q)$ as a new query, say Q'. In the latter case the iteration stops, having obtained the semantic expansion of the query $EXP(Q)$.

Performance issues are very critical for the optimization activity, since every query must be optimized at run time before its execution. We found two solutions to this problem (to be implemented in the ODL-DESIGNER C version):

- pre-processing of schema, in order to materialize the transitive closure of the classes and types hierarchies and to tag each rule with respect to the contents of its antecedent
- fast access, via indexing, to the only rules which are *relevant* to a given query.*

*

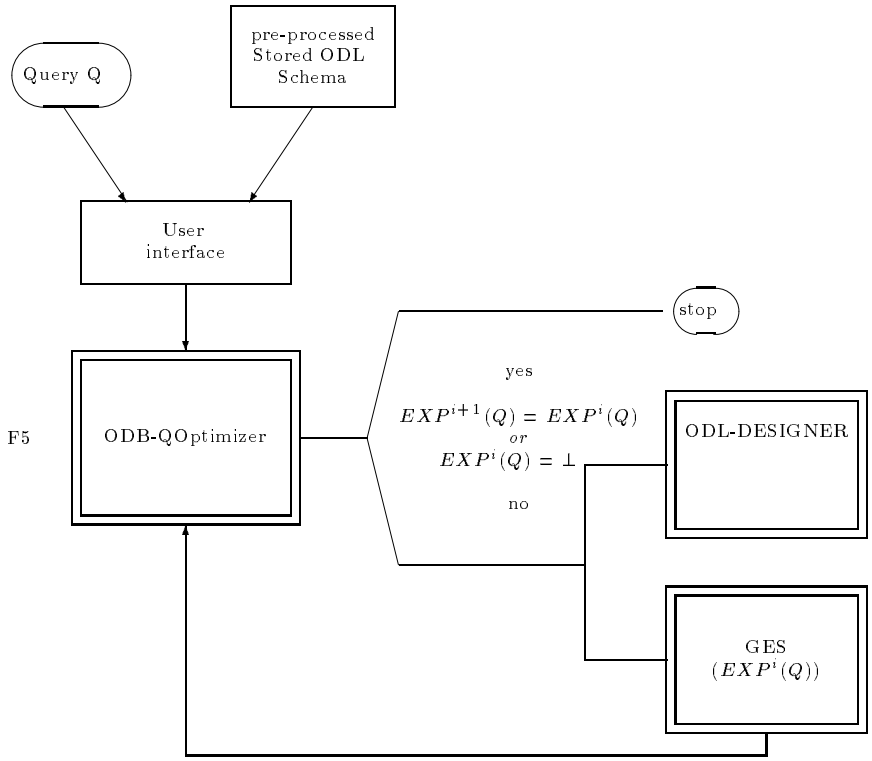


Figure 1: ODB-QOptimizerFunctional Architecture

References

[1] S. Abiteboul and P. Kanellakis. Object identity as a query language primitive. In *SIGMOD*, pages 159–173. ACM Press, 1989.

[2] J. P. Ballerini, S. Bergamaschi, and C. Sartori. The ODL-DESIGNER prototype. In P. Atzeni, editor, *LOGIDATA+: Deductive Databases with complex objects*. Springer-Verlag, 1993.

[3] D. Beneventano, S. Bergamaschi and C. Sartori. Using Subsumption for Semantic query optimization in OODB. *International Workshop on Description Logics*, DFKI Technical Report D-94-10.

[4] 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.

[5] 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*, 4,185-203 (1994).

[6] C. Lécluse and P. Richard. Modeling complex structures in object-oriented databases. In *Proceedings of the 8th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database-Systems*, pages 360–367, Philadelphia, PA, 1989.

[7] 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.

[8] W. Kim. A model of queries for object-oriented database systems. In *Int. Conf. on Very Large Databases*. Amsterdam, Holland, August 1989.

[9] 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.

[10] 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.

Metalevel for an efficient query answering

Bermúdez J. and **Illarramendi A.** and **Blanco J.M.** and **Goñi A.**
Facultad de Informática, Universidad del País Vasco.
Apdo. 649, 20.080 San Sebastián. SPAIN
e-mail: jipbeanj@si.ehu.es

1 Introduction

The integration of heterogeneous and autonomous information sources is a requirement for the new type of cooperative information systems. Heading to this goal we have built a system which allows the integration of heterogeneous and autonomous relational databases using a terminological system ([Illarramendi *et al.*,1991], [Blanco *et al.*,1994b], [Blanco *et al.*,1994a]).

The databases integration process begins with a translation step. A semantically rich terminological representation of each relational schema to be integrated is created. It is followed by a proper integration step.The integrated schema is generated by first, defining correspondences among data elements (concepts and roles) of the terminologies that must be integrated and then, applying some integration rules. Moreover, after translation and integration a mapping information that links data elements of the integrated schema with the underlying databases is also generated.

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 can be used to formulate queries over. The goal of the query processing step is that of finding the answers to these queries in an efficient way. Four different tasks are involved in the query processing step: (1) parsing of the query, (2) semantic optimization, that is, transforming the query into another one with the same answer, but that can be processed more efficiently, and detection of inconsistencies, (3) identification of the cached parts of the query and if possible looking for the answer in the cache memory, and in other cases (4) generation of an optimal plan to obtain the non-cached parts of the query from the underlying databases. This last task involves the translation of the query into relational queries over different databases. In this paper we present an alternative approach to the fourth task, based on the classification culture and trying to take advantage from well known techniques in other areas of systems programming.

We propose a metalevel definition mechanism to classify metaconcepts that will represent relevant features of concepts and roles from the plan generation point of view. Each metaconcept will be associated with an ad hoc generic plan. A proper instantiation of that plan will efficiently generate the answer to a query belonging to that metaconcept. The relevant features of concepts and roles that must be described are (1) the syntactic structure of its definition, and (2) the semantic interrelations satisfied by their components. The aim of this paper is to present a description language developed for those metaconcepts, and a subsumption notion associated to it.

Other works that deal with the integration task ([Spaccapietra *et al.*,1992], [Bergamaschi and Sartori,1992], [Arens *et al.*,1993], [Brachman *et al.*,1993], [Sheth *et al.*,1993]) do not follow this approach. Moreover, as far as we know, the only terminological system that supports a metalevel notion is Omega([Attardi and Simi,1984], [Attardi and Simi,1986], [Hewitt *et al.*,1980]). However, it exceeds our needs, and its powerful expressiveness makes subsumption undecidable, which is inappropriate for our goal.

In the rest of this paper we present our proposal of a description language for metaconcepts. We start defining the basic units and a hierarchical organization among them. Later, we introduce composite expressions of metaconcepts and the notion of subsumption. Finally, we discuss some application examples.

2 The Proposal

As said in the introduction, our work is guided by our interest in introducing a classification approach in the task of plan generation for the query processing step. The next subsections present the main features of the metalevel description language according to the relevant features of terms[1] stated above:(1) syntactic structure and (2) semantic interrelations.

2.1 Basic patterns

Concerning the syntactic structure, patterns specify syntactic categories -metaconcepts- to which the terms can belong to. The basic patterns have been extracted from the collections of term constructors of the terminological language considered. For instance, the basic pattern πr::ANYROLE,c::ANYCONCEPT.**all**(r,c) defines

[1] Hereafter we will use the word terms to refer to concepts and roles.

```
<basic pattern>  ::= ANYCONCEPT
                   | ANYROLE
                   | NAMED-CONCEPT
                   | NAMED-ROLE
                   | prim <pattern name>
                   | π(var::<basic pattern>)*.<term pattern>
<term pattern>   ::= <operator> (<arg>)*
<operator>       ::= <symbol>
<arg>            ::= <var> | <value>
```

Figure 1: Basic Patterns

the syntactic category of the *all* concepts. π binds variables to form parameterized patterns in which the free variables of the term-pattern are just those bounded by π.

We also consider as basic some primitive patterns that cannot be specified by syntactic features alone. For example, **prim** FUNCTIONAL-ROLE defines the syntactic category of the roles that are restricted to be functions. Moreover, there is a similitude between primitive patterns and primitive concepts. In both cases, its membership is usually asserted. For instance, **prim** CACHED defines the syntactic category of concepts and roles that have precalculated extensions. That is, those terms for which we don't need to elaborate a plan to discover their individuals because their extensions are cached. Notice that this information is desirable in a context of efficient retrieval. Membership to **prim** CACHED metaconcept must be asserted because this information is non-definitional.

Figure 1 defines the basic pattern category. The constants ANYCONCEPT and ANYROLE represent the universal metaconcepts of concept and role terms respectively. The NAMED-CONCEPT constant is a subclass of ANYCONCEPT which includes the concepts that are denoted by a name, in contrast with those denoted by expressions. The same applies to NAMED-ROLE and ANYROLE. Thus, all concepts of the integrated terminology are in NAMED-CONCEPT because there is a name associated to their definitions. Bounds of variables act as types. For basic patterns only constant and primitive patterns are allowed as bounds.

Every operator of a parameterized pattern has an attribute associated: the *type* attribute which tells us the primitive or constant pattern of a term that matches the parameterized pattern definition. For example *type* (**all**)=ANYCONCEPT. The basic patterns are arranged in a partial order hierarchy which is defined as the least order relation $\preceq$ respecting the following:

- ANYCONCEPT and ANYROLE are maximal with respect to $\preceq$.
- NAMED-CONCEPT $\preceq$ ANYCONCEPT and NAMED-ROLE $\preceq$ ANYROLE.
- For every parameterized pattern P with operator op: P $\preceq$ *type*(op).
- When defining a primitive pattern P, at most one basic pattern Q can be related as P $\preceq$ Q.

For example:

- πr::ANYROLE,c::ANYCONCEPT.**all**(r,c) $\preceq$ ANYCONCEPT
- **prim** FUNCTIONAL-ROLE $\preceq$ ANYROLE
- **prim** CACHED (is maximal with respect to $\preceq$)

Furthermore, we have defined a binary relation *is inconsistent with* between basic patterns. P *is inconsistent with* Q if it is impossible for them to share an individual. This relation is antireflexive, symmetric and satisfies a sort of transitivity: P $\preceq$ Q and Q *is inconsistent with* R implies that P *is inconsistent with* R. By definition, ANYCONCEPT *is inconsistent with* ANYROLE, and every parameterized pattern is *is inconsistent with* each other. We say that two basic patterns P, Q are consistent if not P *is inconsistent with* Q.

2.2 Complex patterns

Basic patterns are the units to build up complex patterns. Parameterized patterns properly connected can be represented as trees: the operators are nodes and variables are labels for the edges connecting their corresponding arguments (there is an edge for each variable appearing in the introduction of the operator). The leaves of these trees are the constant or primitive patterns which serve as types for the variables, or the values that instantiate the variables.

Patterns can be seen as formal language specifications. Figure 2 shows the operations selected for these specifications. The $\cup$ and $\cap$ operators describe the union and intersection, respectively, of the corresponding language specifications. π-expressions are generalized parameterized patterns where the free variables of the body are just those in the declaration part[2]. Those expressions are fundamental for composing specifications whose meaning is the composition of the meanings of the parts. Restriction on a π-expression either substitutes variables for instance values or restricts the type of a variable to a more specific pattern (a subsumee of the substituted pattern). Constraint addition specifies those terms of the pattern expression part that satisfy the constraint part. The allowed constraints are predicates that correspond to the functionalities provided by the underlying terminological system. We also admit definitions, including recursive ones, that associate a name to a pattern. Figure 3 shows some examples.

Considering the previous model, subsumption between patterns becomes inclusion between languages. Pattern Q *subsumes* pattern P (P $\sqsubseteq$ Q) if the language specified by P is a subset of that specified by Q. Basic patterns form the skeleton hierarchy upon which we build the subsumption hierarchy. For basic patterns P, Q: P $\preceq$ Q implies P $\sqsubseteq$ Q. The $\cup$ and $\cap$ operators specify, respectively, the least upper bound and greatest lower bound of their operands with respect subsumption. ANYCONCEPT $\cup$ ANYROLE is the maximum of this hierarchy. Composition into π-expressions and restriction are monotonic,

[2]Technical problems of variable name clashes are not treated here.

<pattern> ::=<pattern name>
| <basic pattern>
| <pattern> ∪ <pattern>
| <pattern> ∩ <pattern>
| <π-expression>
| <π-expression>(<substitution>)
| <pattern> + <constraint>
<π-expression>::= π(var::<pattern>)*. <pattern>
<substitution> ::= <var> ← <arg>
| <var>::<pattern>
<constraint> ::=terminological system services

Figure 2: Complex Patterns

ALL= π r::ANYROLE, c::ANYCONCEPT. **all**(r,c)
ATLEAST= π r::ANYROLE, n::NAT. **atleast**(n,r)
AND= π c1::ANYCONCEPT, c2::ANYCONCEPT.
and(c1,c2)
ALL-ATLEAST= ALL(c::ATLEAST)
ONE-ROLE-QUERY= π s::ANYROLE.
ALL(r ← s, c::ONE-ROLE-QUERY(s ← s))
∪ ATLEAST(r ← s)
∪ AND(c1:: ONE-ROLE-QUERY(s ← s),
c2:: ONE-ROLE-QUERY(s ← s))
DISJ-RANGE= ALL + [*range*[3]r **disj** c]

Figure 3: Pattern Examples

and constraint addition specifies more specific patterns. The binary relation *is inconsistent with* can be extended straightforwardly to patterns adding the following rule: if P ⊑ Q and Q *is inconsistent with* R then P *is inconsistent with* R. The intersection of two inconsistent patterns specifies the empty language of terms.

3 Applications

The person responsible for the integration will be able to define patterns (metaconcepts), that will be classified by the system, and associate with it a plan if desired. Inheritance provides for alternative plans associated with its subsumers. Queries over the integrated terminologies will be recognized as members of patterns. The preferred plan for the answer is the first associated plan encountered, moving upwards the subsumption hierarchy. (Alternative plans will be available in case of multiple inheritance).

This framework allows to tailor the plans, improving the possibilities of semantic transformations or case analysis of queries.

Let us see the plans as SQL queries. Let X-table be the relational table that supports the concept or role named X. Let **key** be the key attribute for the tables supporting concepts and **fst** and **snd** the attributes supporting the tuples of roles.

Example 1: To the metaconcept ALL(r::ANYROLE, c::ALL(r←s,c←d)) we associate a plan that avoids nested queries (that would result from the compositional naive translation of the semantics of its concept members).

```
SELECT ANYTHING-table.key
FROM ANYTHING-table
MINUS
SELECT r-table.fst
FROM r-table
WHERE r-table.snd=s-table.fst
AND s-table.snd NOT IN
        SELECT d-table.key
        FROM d-table
```

Example 2: During the integration process, a mapping information is created which allows us to establish that the role *age* is a function. Queries such as **atleast**(5,*age*) will be recognized as belonging to the syntactic category π r::FUNCTIONAL-ROLE, n::NAT. **atleast**(n,r) + [n>1] and the plan associated should be the one that informs about an empty set answer query. Notice that we discover cases where the terminological subsumption algorithm does not respond **atleast**(5,*age*)⊑ NOTHING.

Example 3: Suppose an integrated terminology where A-ENDED ⊑ LETTER-ENDED, the role social security identifier is defined as *soc-sec* :<**domain**(PERSON) **and range**(KEY-ID) and the terminological system can deduce LETTER-ENDED **and** KEY-ID ⊑ NOTHING. Moreover, the mapping information registers that the role *soc-sec* represents a non null key attribute for persons (that is, *soc-sec* is total on its domain).

A user who ignores the characteristics of the social security identifier may ask for PERSON **and all**(*soc-sec*, A-ENDED). Usual subsumption algorithms cannot discover that there is no person satisfying the description. Nevertheless, the query can be recognized as an individual of the metaconcept

AND(c1::ANYCONCEPT,
c2::ALL(r::**prim** DOM-TOTAL,
c::ANYCONCEPT))
+ [*dom* r ≡ c1, range r **disj** c]

Notice that, on the one side, the extension of any concept satisfying this pattern are members of c1 whose fillers for r fall in c. On the other, for those concepts, it must be verified:

1. *dom* r ≡ c1 and r is total on its domain, so r has fillers for any member of c1 and
2. range r **disj** c, so fillers for r are never in c.

Therefore we get a contradiction which implies that the extension is the empty set. The plan associated is the same as the one mentioned in the previous example.

[3]Any role has two attributes: *dom* and *range*.

References

[Arens *et al.*, 1993] Y. Arens, C.Y. Chee, C. Hsu, and C.A. Knoblock. Retrieving and integrating data from multiple information sources. *International Journal of Intelligent and Cooperative Information Systems*, 2(2):127–158, 1993.

[Attardi and Simi, 1984] G. Attardi and M. Simi. Metalanguage and reasoning across viewpoints. In *Proceedings 6th European Conf. on Artificial Intelligence. Pisa.Italy*, 1984.

[Attardi and Simi, 1986] G. Attardi and M. Simi. A description-oriented logic for building knowledge bases. In *Proceedings of the IEEE, Vol. 74, No. 10*, October 1986.

[Bergamaschi and Sartori, 1992] S. Bergamaschi and C. Sartori. On Taxonomic Reasoning in Conceptual Design. *ACM Transactions on Database Systems*, 17(3):385–421, 1992.

[Blanco *et al.*, 1994a] J.M. Blanco, A. Illarramendi, and A. Goñi. Building a federated database system: an approach using a knowledge based system. *International Journal on Intelligent and Cooperative Information Systems*, 3(4), December 1994.

[Blanco *et al.*, 1994b] J.M. Blanco, A. Illarramendi, A. Goñi, and J. Bermúdez. Advantages of using a terminological system for integrating databases. In *Proc. of the International Workshop on Description Logics. Bonn*, 1994.

[Brachman *et al.*, 1993] R.J. Brachman, P.G.. Selfridge, L.G. Terveen, B. Altman, A. Borgida, F. Halper, T. Kirk, A. Lazar, D.L. McGuinnes, and L.A. Resnick. Integrated support for data archeology. *International Journal of Intelligent and Cooperative Information Systems*, 2(2):159–185, 1993.

[Hewitt *et al.*, 1980] C. Hewitt, G. Attardi, and M. Simi. Knowledge embedding in the description system omega. In *Proceedings of First National Annual Conference on Artificial Intelligence. Stanford University*, August 1980.

[Illarramendi *et al.*, 1991] A. Illarramendi, J.M. Blanco, and A. Goñi. A uniform approach to design a federated information base using BACK. In *Proc. of the Terminological Logic Users Workshop. Berlin*, 1991.

[Sheth *et al.*, 1993] 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.

[Spaccapietra *et al.*, 1992] S. Spaccapietra, C. Parent, and Y. Dupont. Model independent assertions for integration of heterogeneous schemas. *The VLDB Journal*, 1(1):81–126, 1992.

Caching in Multidatabase Systems based on DL

Alfredo Goñi and **Arantza Illarramendi** and **José Miguel Blanco**
Facultad de Informática, Universidad del País Vasco.
Apdo. 649, 20.080 San Sebastián. SPAIN
e-mail: jibgosaa@si.ehu.es

1 Introduction

MultiDataBase Systems (MDBS) have been proposed as a solution to work with different pre-existing autonomous databases. There exist many different approaches for building a MDBS, namely the Entity-Relationship model approach [Larson *et al.*,1989; Spaccapietra *et al.*,1992], the Object-Oriented approach [Ahmed *et al.*,1991; Czejdo and Taylor,1991], and the Knowledge Representation Systems (KRS) approach [Collet *et al.*,1991; Sheth *et al.*,1993]. In our case we have built a MDBS that integrates several heterogeneous relational databases[1] by using a KRS based on Description Logics (DL system). In [Blanco *et al.*,1994b; Blanco *et al.*,1994a] we show the advantages of using a DL system for building a Relational Multidatabase System.

Three different types of problems are involved when building a MDBS: *translation* of the underlying database schemata into schemata expressed in a canonical model, *integration* of the translated schemata into an integrated schema and *query processing* of the user-formulated queries over the integrated schema by accessing the underlying databases. Although there has been a lot of research about the problems of translation and integration of schemata to obtain integrated ones the problem of query processing against these integrated schemata has not been treated so much. In our case, the integrated schema is represented by a knowledge base built upon the different relational database schemata, and the extension of the knowledge base (the instances of the classes and attribute values) is in fact stored in the underlying databases. When a query is formulated over that knowledge base the answer must be obtained by accessing the different databases.

Client/Server architectures have been shown as appropriated for building and supporting Multidatabase systems. Using this type of architecture a Client application can be defined that deals with the integrated schema and several Server applications, one for each database that participates in the Multidatabase system. In this context, it is worth having some data cached in the real extension of the knowledge base in order to avoid accessing the underlying databases each time a user formulates a query. Communication cost involved in transferring intermediate results among the nodes and the final reconstruction of the answer can be avoided.

Unfortunately, it is not possible to have all the data cached for several reasons:

1. Due to the autonomy of the underlying relational databases, their extensions can be updated very often and so the cached data in the real extension of the knowledge base would become inconsistent with the extension stored in the underlying databases.
2. The size of the cache memory would be obviously huge because it would be the sum of the size of several databases. This would produce space problems, most of the times it is not possible to have such a huge cache memory, and time response problems, because frequently asked queries could be answered slowly if some not so frequently asked queries were cached.

There are some related works where a DL system is used in connection to information sources but only a few of them talk about query processing aspects. In particular, in [Devanbu,1993] Devanbu explains how translators from DL queries to database queries can be built. Borgida and Brachman [Borgida and Brachman,1993] present an efficient translator from DL queries to SQL queries and also presents some problems when loading data into the DL knowledge base. In the two previous cases only one database is connected to the DL knowledge base and the whole DL knowledge base is loaded at the beginning of the session. Therefore, there is not a caching problem, because everything is cached since the beginning. In [Arens *et al.*,1993] Arens et al. show the SIMS system, that integrates data from several information sources (databases and Loom knowledge bases). From the query processing point of view they select the appropriate information sources, create plans to answer the query and reformulate these plans by exploiting knowledge about the domain and the information sources. They also point in a recent paper [Arens and Knoblock,1994] that the retrieved data can be cached and give several principles that describe the interesting

[1] Although they all use the same data model, semantic heterogeneity still can remain.

data to cache but they do not propose concrete solutions for the problems that appear.

The aim of this paper is to show how some features of DL systems can be useful when including a caching optimization mechanism for query processing in a Multidatabase system. To our knowledge, this work is the first to address caching within the context of MDBSs. In the rest of the paper we present the cache definition and strategies, and then the cache optimization during the query processing.

2 Cache definition and strategies

When defining a caching optimization mechanism there are several aspects to be considered: first of all, the types of objects to be cached must be selected and then, it is necessary to decide the optimal set of objects that are worth caching. As some explicitly cached objects may implicitly cache other objects this has to be taken into account in order to select that optimal set. And finally, the strategy for caching has to be defined.

2.1 What types of objects can be cached

When working with a DL system there are two different types of values associated with every instance: the *object identifier (OID)*, that is the unique value that identifies it, and the particular values taken by its attributes. The question now is which data should be cached: OIDs or values? If only OIDs are cached, less space is needed and probably fewer problems arise with the consistency of the cache, but queries that refer to value instances, the most common ones, will not be able to be answered accessing only to the cache.

Therefore it is possible to cache the OIDs of the instances of a class description (hereafter to *cache a class description* to which a class name is given) and to cache the attribute values for each instance of a class description (hereafter to *cache an attribute* for a class). Notice that this is different than working with object-oriented database systems where only instances of entire classes can be cached. Moreover, dealing with DL systems, to cache class descriptions or attributes is the same as to cache queries as it can be seen in figure 1 where the equivalence between objects and queries is shown.

object to cache	corresponding query
class description	*getall class description*
attribute of a class description	*[rf(attribute)] for getall class description*

Figure 1: Equivalence between objects and queries.

2.2 Optimal set of queries to cache

It is important to decide which queries are worth caching. Candidate queries to cache are: 1) *frequently asked* queries; 2) *non-volatile*, that means that are not frequently updated; 3) with a *high cost* of retrieving from the underlying databases; and 4) with a *small size* in order to occupy less in the cache.

In [Goñi *et al.*,1995] we show a method that calculates the optimal set of queries to cache, that is, the queries that provide the best benefit for a limited size of cache. It makes use of several parameters such as probability of asking for each query, probability of updating the extension of a query, response time for a query in the underlying databases and in the cache and time for caching a query. These parameters (or the process that works with them) have to take into account that some queries are included in others and that the explicit caching of some queries can implicitly cache another ones.

Working with DL systems, a query q is implicitly cached if the classes in the **Most Immediate Superclasses** *(MIS) of the class description of q are cached and all the attributes that appear in the MIS and the projected attributes in the query are cached. An attribute A for a class C is implicitly cached if A is cached for a super class of C and the class C is cached.*

Implicitly cached queries are queries that can be answered using other explicitly cached queries and do not occupy extra space in the cache. When there are some explicitly cached queries and a new one becomes explicitly cached then other queries can be implicitly cached.

For example: suppossing that there exists a knowledge base with the classes *course*, *person* (with attribute *name*), where *person* has two subclasses *teacher* (with attributes *teaches, teaches_to* and *title*) and *student* (with attribute *studies*). And there are also four *defined* classes (*teaching_assistant, super_student, super_teaching_assistant* and *lucky_teacher*). The knowledge base definition is shown in figure 2.

CLASSES
person :< *anything*
student :< *person*
teacher :< *person*
course :< *anything*
teaching_assistant := *teacher and student*
super_student := *student and atleast(10,studies)*
super_teaching_assistant := *teacher and student and atleast(10,studies)*
lucky_teacher := *teacher and atmost(0,teaches_to)*
ATTRIBUTES
name :< *domain(person) and range(string)*
title :< *domain(teacher) and range(string)*
teaches :< *domain(teacher) and range(course)*
teaches_to :< *domain(teacher) and range(student)*
studies :< *domain(student) and range(course)*

Figure 2: Knowledge base

- The caching of a class description may implicitly cache another class.

  For example, suppose that *rf(studies) for getall student* is cached, when *getall teaching_assistant* be-

comes cached then *getall super_teaching_assistant* will be implicitly cached because any instance of *super_teaching_assistant* is an instance of *student* and an instance of *teacher*, that is, an instance of *teaching_assistant*, with at least ten values in the attribute *studies*. Notice that *teacher* is not cached but the MIS *teaching_assistant* is.

- The caching of a class description may implicitly cache an attribute.

  If the attribute *name* is cached for the class *person* (*rf(name) for getall name*), when the query *getall student* becomes cached then *name* will also be implicitly cached for *student* (*rf(name) for getall student*).

- The caching of an attribute may implicitly cache a class.

  If the attribute *studies* is cached for the class *student*, (*rf(studies) for getall student*) then the query *getall super_student* is implicitly cached because any instance of *super_student* is an instance of *student* with at least ten values in the attribute *studies*.

- The caching of an attribute for a class may implicitly cache that attribute for another class.

  If the queries *getall person* and *getall student* are cached when the attribute *name* becomes cached for the class *person* (*rf(name) for getall person*), then *name* will also be implicitly cached for *student* (*rf(name) for getall student*) because if the names are known for all the persons and the students can be identified from those persons, then the names for the students are also known.

Therefore we can conclude that

- when *getall class_description* is explicitly cached then queries that have *class_description* as a super class may become implicitly cached, and queries *rf(attribute) for getall class_description* may become cached if *rf(attribute) for getall super_class_for_class_description* is cached.
- when *rf(attribute) for getall class_description* is explicitly cached then queries that have a restriction for that *attribute* may become implicitly cached, and queries *rf(attribute) for getall sub_class_description* may become cached if *getall sub_class_description* is cached.

2.3 Static and dynamic caching

The process that calculates the queries worth to be cached has to take into account many parameters and it cannot be executed each time a query is made. Furthermore, it is possible that this process decides not to cache the last made queries because their probabilities of asking are not great or because they are very volatile. However, the work with the knowledge base is made in sessions where one loads it, queries something and ends the session. It is quite possible that the most recently query made is done again in the same session (although in all sessions is not usually asked). To solve this problem, we use two different replacement policies for the cache: the *static* and the *dynamic*. The *static* strategy consists on caching the set of optimal queries as said in section 2.2 and the *dynamic* strategy consists on caching the last formulated queries and deallocate, when space is needed, the *least recently used* queries. The *static* strategy is used between sessions (at the end of the session or in off-peak hours) and the *dynamic* strategy is used during the query processing within a session.

3 Cache optimization during the query processing

During the query processing task it is necessary to detect if the query can be answered with the data stored in the cache (in our case the real extension of the integrated knowledge base), that is, if the query *is contained* in the cache. As queries are descriptions of data, it has to be proved that any data that verifies the description is in the cache. In general, to verify if a query is in the cache it is not easy and it depends on the query language and on the representation of the cached data.

3.1 Identification of the cached parts of the query

When working with a DL system, the classification mechanism of classes can be used to verify if queries are cached. If a query class is subsumed by the cached classes then it is true that the instances of the query class are in the cache. However, that does not mean that they can be identified and that the query can be answered directly from the cache.

For example, suppose that all the instances of the class *person* are cached and that the next query is formulated: obtain all the persons with at least five children (*getall person and atleast(5,children)*). In fact, it is true that all the instances of *person and atleast(5,children)* are in the cache because all the instances of *person* are cached. However, it is not possible to answer the query unless the attribute *children* is also cached because it is not possible to distinguish which persons have at least 5 children if the children are not known (another possibility would be that the class *person and atleast(5,children)* were cached).

We can say that the query *[rf(r_1),...,rf(r_N)] for getall* C_1 *and ... and* C_M is cached *if all the class names that appear in* C_i *and all the attributes that appear in* C_i *and* $r_1,...,r_N$ *are cached.* But this is a too strong restriction because although two classes were not cached, the intersection of both could be cached and the same query would be also cached. In fact, a query is cached *if the MIS of the class description of the query* C_1 *and ... and* C_M *are cached and also the projected attributes* $r_1,...,r_N$.

3.2 Obtaining of a set of DL queries to be retrieved

It is obvious that if the query is cached then it is answered from the cache, but if the query is not completely cached then it has to be answered by accessing the underlying databases. Therefore, in the query there are some *cached* parts and some *non-cached* parts. The point here is: which parts have to be retrieved from the underlying databases and introduced in the knowledge base in order to get the answer to the original query?

- not all the class names and attributes have to be retrieved from the databases because parts of the query may already be cached
- not only the non-cached parts of the query have to be retrieved because it may be more costly. Retrieving only the non-cached parts implies to bring more data (because DL queries are conjunctive queries) and also the final computation in the knowledge base is more expensive. However, retrieving only the non-cached parts can avoid accessing to more than one database and can also be useful to answer future queries.

Therefore, there is a trade-off here between retrieving only the non-cached parts with a greater communication cost and more computation in the knowledge base or retrieving the non-cached and some cached parts that produce less communication cost, less computation in the knowledge base but more computation in the databases (above all if more databases have to be accessed).

We will show with an example the different possibilities of DL queries to cache in order to answer the original query. Suppose that the next query is made:

[rf(name)] for getall student and teacher and atleast(10,studies) and atmost(1,teaches)

In the figure 3 the classes, attributes and attribute restrictions needed to answer the query can be seen as a tree. In the first level of the tree, the nodes are the MIS for the query and the projected attributes. Every node that corresponds to a defined class is expanded with its MIS and so on. The underlined classes and attributes are cached.

Query
class: teaching_assistant
class: student
class: <u>teacher</u>
class: super_student
class: student
cardinality restriction: atleast(10,studies)
cardinality restriction: atmost(1,teaches)
attribute to project: <u>name</u>

Figure 3: Tree corresponding to the example query

For the previous query the different possibilities of DL queries to retrieve and cache from the underlying databases are:

1. to cache the query, equivalent to the initial one, formed by the conjunction of the MIS;
   [rf(name)] for getall teaching_assistant and super_student and atmost(1,teaches)
2. to cache the conjunction of only the non-cached parts of the query;
   getall student and atleast(10,studies) and atmost(1,teaches)
3. to cache the values for the attributes of the restrictions;
   [rf(studies),rf(teaches)] for getall student
   The next two queries can also be retrieved because they can be executed in parallel in both databases:
   [rf(studies)] for getall student
   [rf(teaches)] for getall teacher
   But as *teaching_assistant* has a support in both databases, the queries to be retrieve can be:
   [rf(studies)] for getall teaching_assistant
   [rf(teaches)] for getall teaching_assistant

The strategy used to get the set of DL queries to cache consists on two phases:

- To transform the query formed by the MIS and the projections of attributes into another one
  1. MIS and projected attributes that are already cached are removed from the query. However, some cached MIS are kept in the query in order to reduce the size of the answer, if it is too big, or if there are other non-cached parts in the same database node where the cached part is.
  2. *Defined* MIS with alternative *mapping information*[2] are kept in the query.
  3. *Defined* MIS with no alternative mapping informations are substituted by their corresponding non-redundant MIS. To these new MISs the steps 1,2, 3 are applied.
  4. *Defined* MIS with alternative mapping informations may be substituted by their corresponding non-redundant MIS (and applying 1,2, 3), above all if some of them are already cached, and there are possibilities of asking for some of them and space in the cache.
  5. Attribute restrictions may be substituted by the projection of the corresponding attributes if there are possibilities of asking for these attribute values and space in the cache.
- To split it into queries to be executed in parallel if it is possible.

[2] When dealing with FDBS it is necessary to define a linking information between the integrated schema and the underlying databases. We call *mapping information* to this linking information.

1. The query can be split in subqueries such that each one of the subqueries are in the same database. In this case, the split subqueries can be executed in parallel.
2. If there are possibilities of asking for part of the query, this part can be separated from the query.

Some of the previous transformations imply to bring and cache more data than the requested in the query. This follows the dynamic strategy, because it tries to cache the queries made in the session. It supposes that there are more possibilities of asking for these data. Anyway, it is possible to *cooperate* with the user and ask him if he is interested in other related data to the query. For example if the user queries *getall teacher and atleast(3,teaches)* then the system can ask the user:

`Do you want to know the values of attribute` *teaches*?

If the answer is *yes* then the query *[rf(teaches)] for getall teacher* is split and cached.

References

[Ahmed *et al.*, 1991] R. Ahmed, P. Smedt, W. Du, W. Kent, M. Ketabchi, and W.A. Litwin. The Pegasus heterogeneous multidatabase system. *IEEE Computer*, 24, December 1991.

[Arens and Knoblock, 1994] Y. Arens and C. A. Knoblock. Intelligent caching: Selecting, representing, and reusing data in an information server. In *Proceedings of the Third International Conference on Information and Knowledge Management CIKM*, 1994.

[Arens *et al.*, 1993] Y. Arens, C.Y. Chee, C. Hsu, and C.A. Knoblock. Retrieving and integrating data from multiple information sources. *International Journal of Intelligent and Cooperative Information Systems*, 2(2):127–158, 1993.

[Blanco *et al.*, 1994a] J.M. Blanco, A. Illarramendi, and A. Goñi. Building a federated database system: an approach using a knowledge based system. *International Journal on Intelligent and Cooperative Information Systems*, 3(4), December 1994.

[Blanco *et al.*, 1994b] J.M. Blanco, A. Illarramendi, A. Goñi, and J. Bermúdez. Advantages of using a terminological system for integrating databases. In *Proc. of the International Workshop on Description Logics. Bonn. Germany*, 1994.

[Borgida and Brachman, 1993] A. Borgida and R. J. Brachman. Loading data into description reasoners. In *Proceedings of the ACM SIGMOD Conference*, 1993.

[Collet *et al.*, 1991] C. Collet, M. N. Huhns, and W. Shen. Resource integration using a large knowledge base in CARNOT. *IEEE Computer*, December 1991.

[Czejdo and Taylor, 1991] B. Czejdo and M. Taylor. Integration of database systems using an object-oriented approach. In *First International Workshop on Interoperability in Multidatabase Systems*, April 1991.

[Devanbu, 1993] P.T. Devanbu. Translating description logics to information server queries. In *Proceedings of the ISMM International Conference on Information and Knowledge Management CIKM*, 1993.

[Goñi *et al.*, 1995] A. Goñi, A. Illarramendi, and E. Mena. Semantic query optimization and data caching for a multidatabase system. In *Proceedings of the Basque International Workshop on Information Technology*. IEEE Computer Society Press, July 1995. To celebrate in San Sebastian, Spain.

[Larson *et al.*, 1989] J. A. Larson, S. B. Navathe, and R. Elmasri. A theory of attribute equivalence in databases with application to schema integration. *IEEE TOSE*, SE-15(4), April 1989.

[Sheth *et al.*, 1993] 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.

[Spaccapietra *et al.*, 1992] S. Spaccapietra, C. Parent, and Y. Dupont. Model independent assertions for integration of heterogeneous schemas. *VLDB*, 1:81–126, 1992.

Cooperative Recognition of Interdatabase Dependencies

M. Klusch
Institut für Informatik und Praktische Mathematik
Christian-Albrechts-Universität Kiel, Olshausenstr. 40, 24118 Kiel, Germany
E-Mail: *mkl@informatik.uni-kiel.d400.de*

(Extended Abstract)

Abstract

A novel approach towards the recognition of interdatabase dependencies (IDD) using a federative agent system FCSI is presented. The architecture of the FCSI is designed as a set of coalition-based cooperative, intelligent agents each of them uniquely assigned to one autonomous local database system. The *FCSI aims for a cooperative solution for the problem of searching for semantically related information while strictly respecting the autonomy requirements of each individual database system.* For this purpose first a *terminologically represented local domain information model on top of the local conceptual database schema* is built at each agent by processing scripts specified by user's intention on externally available semantic aspects or views of some local *schema objects*. These objects are then appropiately linked into the local information ontology by the local FCSI agent. Remote *terminological classification* then serves as a basis for the *recognition of intentional IDD*s between objects of different schemas with respect to some or all of their sofar intentionally related semantic aspects. Projections on respective associated *aspect valuations at schema and state level* then determine agent-directed, *context-based data sharing* and result in restricted proposals for *interdatabase schema assertions*. Methods for *utilitarian coalition building among the rational agents of the FCSI* are used in order to cooperatively search for semantically related data. The decentralized calculation of each agent's utility bases on their local productions resulting from the execution of own and received tasks to find such dependencies between local terminological information models. There is no prior need and even no possibility to browse through non-local schema structures in order to find some possibly relevant data. In this paper the current status of ongoing research on the FCSI is reported.

1 Introduction

The approach of the FCSI is motivated by the idea of following the recently introduced paradigm of intelligent cooperative information systems ICIS [20] for solving one essential problem in the research area of interoperability of database systems: *a context-based recognition of interdatabase dependencies (IDD) between heterogeneous, autonomous database systems.* An IDD describes the relationship between related data of different database schemas by an integrity constraint [8]. Any approach of declarative or functional specification of IDDs like in [25],[7],[8] presumes somehow gained knowledge about where to find which kind of semantically related data. The related problem of tackling semantic heterogeneity[23] is aggravated by the need to respect in particular the association autonomy[24] of all respective database systems. In other words, the problem is how to find in a completely decentralized environment some semantically related non-local schema data having no prior possibility to browse through all respectively exportable schema structures. Criticism[5] on federated database systems FDBS[24] state in particular their lack of support with respect to this *object discovery problem*[10] and thus the recognition and maintenance of IDDs [20]. The FCSI is the first approach towards a federative system[18] which aims for such an intelligent support of a context-based recognition of interrelated data in autonomous databases. For this purpose the FCSI uses in particular methods from the different research areas of terminological knowledge representation and reasoning [19] as well as distributed artificial intelligence (DAI)[18].

Local construction of a terminological information model on top of the local database schema and appropiate linking of some externally available schema objects into the local ontology enables the agent to find semantically related data already at information type, i.e. terminological level. This is done by sending some aspect terms each describing one semantic aspect of a schema object terminologically and their formal classification into the local terminological information models by the receiving agents. Proposed interschema assertions are compositions of locally attached state constraints on related schema objects with respect to the type of detected ter-

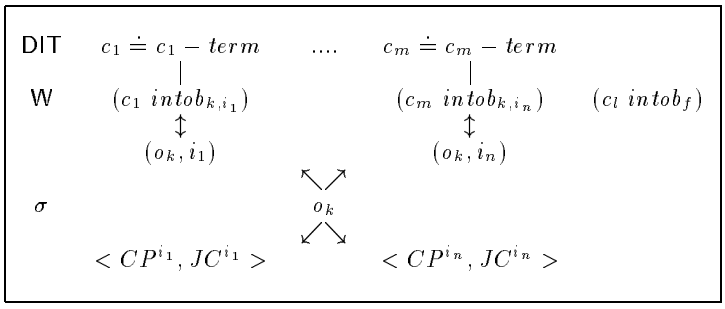
minological relation between both schema objects. Such recognition of some interdatabase dependencies and respective data sharing is done without any efforts in partial or global schema integration. In order to organize a *cooperative search for semantically related data* methods for *utilitarian coalition building among the autonomous agents of the FCSI* are currently investigated. Thus, no global information agent [3] or central mediator agent [4] exists. For implementation of FCSI agents we currently develop an interactive development environment for the specification and simulation of agent systems IDEAS-1 on a set of networked SUN-workstations.
The remainder of this paper is organized as follows. In section 2 a short overview on the functionality and architecture of each FCSI agent is given. Section 3 briefly concludes and gives a outlook on future research on the FCSI.

2 FCSI : Functionality and Architecture

As already reported in [17],[15] according to a set of user-specified *intentional scripts* on some exportable own schema object structures a local *domain information terminology* DIT and its actually instantiated schema aspect world W on top of the conceptual database schema is incrementally built at each agent. For this purpose an *information terminological formalism* ITF as well as a *schema aspect assertional formalism* SAF as its conservative extension[19] is used. This KL-ONE based terminological description language provides most usual term-forming operators like conjunction, number restriction and value restriction as well as atomic concept negation [1]. For reasons of space limitation for a comprehensive introduction to hybrid terminological knowledge representation and reasoning we refer the reader to [19]. The main steps for the construction of a local informtaion model DIT/W are as follows.

- **by user**:
  for each semantic aspect i_x of available schema object $o_k \in SObj$ specify one so-called *intentional aspect script* with two following kinds of entries
  1. *terminological aspect description*:
     - term introductions of relevant information reference concepts c (initial)
     - set of appropiate, consistent assertions about object o_k considered as an instance of c; [2]
  2. *aspect valuation over database schema* σ:
     - (part of) relevant schema object structure: CP^{i_x}
     - at database state level (DML): JC^{i_x} [3]
- **by FCSI Agent**: *Construction of Local Information Model and Schema Object Linking*
  1. Read all given aspect scripts;
  2. **Build** or extend the local terminology DIT:
     - Collect all term introductions of concepts and roles
  3. **Determine** directed, acyclic information concept hierarchy:
     - Compute Term subsumption hierarchy;
  4. **Build** actual aspect world W of reference objects $intob \in RObj$:
     - Create for each schema aspect (o_k, i_x) one reference object $intob_{k,i_x}$
     - Collect all assertions, substitute occurrences of o_k by $intob_{k,i_x}$
  5. **Check** consistency of assertions in W wrt. the terminology DIT;
  6. **Store** one computed *Aspect Term* for each Schema Aspect (o_k, i_x):
     - build conjunction of expanded terms by constraint propagation on all given assertions for unique reference object $intob_{k,i_x}$ in W, i.e. terminological abstraction of the reference object wrt. DIT/W.

Linking one schema object o_k into the agent's local terminological knowledge base[4] is then formally done by interpreting the respective reference objects as one aspect of o_k.



EXAMPLE 1: see Appendix A.1.

[1] We currently use the terminological language NTF/AF proposed in [19] as ITF/SAF. Several alternatives for this choice like e.g. ALC [27][11] or ALCF(D) [1] are currently under consideration wrt. some efficient implementation of respective algorithms for hybrid terminological reasoning within each agent. Term subsumption is decidable for NTF/AF[19] as well as for ALC and its several derivatives[27]. As in most practical terminological systems for reasons of efficiency actually used subsumption algorithm is polynomial but incomplete. A formal description of the respective linking of schema objects into the terminological information model can be found in [14]. It enables in particular a flexible modeling of distinct semantic aspects of a schema object the user has in mind for external representation. The expressivity of ITF/SAF restricts the more natural-language based description of semantic aspects of schema objects the user intends to represent. Besides, it depends on the user how close he models the DIT/W wrt. the given schema by the use of same names or pure data structure copy[2]. In order to achieve homogeneous interschema assertions we currently presume an EER data model[12] as canonical data model for each local schema[24], although this assumption is not necessary for the first phase of finding related data at terminological information level.

[2] Terms of named information concepts are formulated in ITF, consistent term restrictions for asserted instances in SAF.

[3] The justification constraint JC^{i_x} is a *state constraint, function or self denotation* actually specified by user with respect to an aspect i_x of o_k he has in mind. Detailed examples are in [14].

Similar to the classical view definitions this allows to attach the same schema object to several information concepts where each of them is relevant for a terminological description of one of the object's particular semantic aspect. Roughly speaking, this enables to negotiate about each other's conceptual schema at information type level [5] *without* actually having the need as well as the possibility to access the respective, fixed schema data structures itself. It is possible for each agent user to change the actual terminological representation of some aspect of an schema object without the need to change the valuation at schema or state level and vice versa.
It is now possible for the agents to automatically detect a set of terminological relations between two objects o_1,o_2 in different schemas with respect to some or all of their aspects, the so-called *intentional interdatabase dependencies* i-IDD.

Let $tsub(\ t_2, t_1\ )$ compute *term-subsumption* $t_2 \succeq t_1$, $intabst(o_1, i_x)$ denotes the computed *aspect term* of (o_1, i_x) and $intset_1$ set of *aspect identifier* of schema object o_1, then :

$\mathbf{p-intsub(o_1, o_2, M, N)} :\Leftrightarrow$
$(\ \forall i_x \in M \subseteq intset_1 \exists i_y \in N \subseteq intset_2$:
$tsub(intabst(o_2, i_y), intabst(o_1, i_x))\ ) \wedge$
$(\ \forall i_y \in N\ \exists i_x \in M$:
$tsub(intabst(o_2, i_y), intabst(o_1, i_x))\ )$

$\mathbf{c-intsub(o_1, o_2, M, N)} :\Leftrightarrow$ p-intsub with M $= intset_1$
$\mathbf{inteq(o_1, o_2)} :\Leftrightarrow c-inteq(o_1, o_2, intset_1, intset_2)$
$\mathbf{intdis(o_1, o_2)} :\Leftrightarrow c-intdis(o_1, o_2, intset_1, intset_2)$
$\mathbf{intsub(o_1, o_2)} :\Leftrightarrow c-intsub(o_1, o_2, intset_1, intset_2)$

For example, $\mathbf{p-intsub(o_1, o_2, M, N)}$ means that schema object o_1 is partially, intensionally subsumed by o_2 exactly wrt. the (sub-)sets M, N of their semantic aspects. The complete set of these terminological relations can be found in [14]. In accordance with [21], to enable mutual understanding of received foreign terms a set $cpc_{i,j}$ of corresponding local primitive components is used between communicating agents a_i, a_j to avoid linguistic ambiguity at the lowest level.
The *recognition process of terminological dependencies i-IDDs* executed at each agent locally bases then on classifying a received aspect term into the local terminological information model, finding relevant reference objects and then projecting down to the attached local schema data [17].
EXAMPLE 2: see Appendix A.2.

Agent's ability to propose some *interdatabase schema assertion IDSA* is then realized by using integration rules for composing the state constraints JC on attached schema objects with respect to the terminological relation:
if $p-inteq(o_k, o_j, \{i_x\}, \{i_y\})$ **then**
propose $[JC^{i_x} \Leftrightarrow JC^{i_y}]$;
if $p-intsub(o_k, o_j, \{i_x\}, \{i_y\})$ **then**
propose$[JC^{i_x} \Rightarrow JC^{i_y}\ ]$; a.o.

Example for a composition by propose:

Let $p-intsub(o_k, o_j, \{i_x\}, \{i_y\})$ be recognized with

$o_k = E_1 \in E_{\sigma_1},\ o_j = E_2 \in E_{\sigma_2}\ of\ EER-Schemas \sigma_1, \sigma_2$;
$CP^{i_x} \subseteq attr(E_1), \quad CP^{i_y} \subseteq attr(E_2)$;
$JC^{i_x} := \forall e1_{E_1} : < ic_1 - qual_{e_1} >$,
$JC^{i_y} := \forall e2_{E_2} : < ic_2 - qual_{e_2} >$

propose: $\forall e1_{E_1}, e2_{E_2} :\ (e1.key(E_1) = e2.key(E_2)) \Rightarrow$
$(\ < ic_1 - qual_{e_1} > \Rightarrow < ic_2 - qual_{e_2} >\ )$

Each set of IDSA proposals gives hints for possible object domain mappings and can be extended as agent's dependency knowledge grows. Rules for IDSA proposals are further investigated in a more detail in [14].
EXAMPLE 3: see Appendix A.3

Context-based data sharing is now possible by any request of the agent a_j on a remote schema object o_2 to a_i yet known to be related wrt. some aspects. The receiver of an agent's data query like `request-for` structure part `from` schema object `wrt` aspect will first check the respective aspect identifier, select the corresponding state constraint and then compile the contextual query into a local database query.
EXAMPLE 4: see Appendix A.4

The purpose of each FCSI agent is to support the user in discovering the available information space with respect to its own local domain of interest DIT. This can be followed up by the agent through detecting some i-IDDs in cooperation with other agents for satisfying the respective local FIND-tasks. Execution of REQUEST-FOR-tasks, i.e. agent data queries, is possible only after some terminological relations are found and only permitted between fixed coalition members.
The *modular structure of a FCSI agent* is given in Fig. 1.

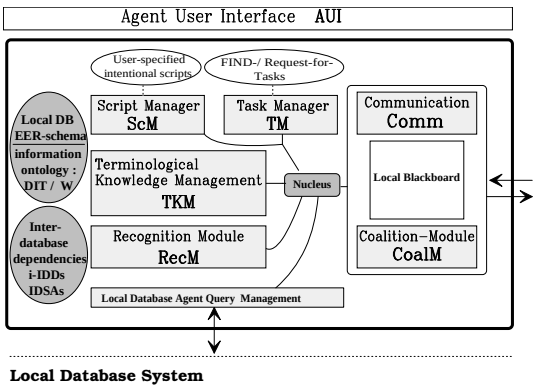


Figure 1: Modular structure of a FCSI agent

The *Agent User Interface* AUI enables the local user for task specification as well as to access all internal modules of an agent. A central *nucleus* NM is responsible for internal task management, coalition negotiation and action scheduling with respect to the execution of some task received from local user or other agents. All user-specified intentional scripts on schema object aspects are

maintained by the *Script Manager* SM. The terminological knowledge DIT/W on the local schema is then constructed and maintained by the *TKM module*. A *Recognition module* RM infers i-IDDs by joining the TKM and projecting on respective valuations stored at the SM. It is also responsible for maintaining these detected i-IDD in its *belief base* which includes in particular all logically deducted i-IDD facts (PROLOG facts). Any request on some particular or the existence of any i-IDD with other agents are handled by the RM (using PROLOG). IDSA proposals are reported by a RM submodule. Cooperation dialogue can be modeled by using a set of cooperation primitives, objects and constraints[18] in compliance with KQML[9]. The *coalition module* CoalM is responsible for determining the agent's interest in participation in some coalition with other FCSI agents.
FCSI coalitions are group of agents which are mutually committed to execute some of their tasks for a limited period of time under certain negotiated constraints: Each agent tries to find and to coalesce with other agents which are able to satisfy the posed task to find related data with respect to his own terminological local information model. Members of a fixed coalition are committed to maintain any of the respectively known inter-database dependency just within a coalition by providing in particular immediate notification on any relevant local modification of domain aspect representation and executing local database queries generated from received REQUEST-FOR-tasks.
Each possible coalition obtains a certain utility, its *coalition value*, which is then fairly *shared* among its members regarding their *marginal contribution* to this coalition. In the FCSI it is currently assumed that the agents have common knowledge of this coalition value function of the respective coalition game and agree on the utility division method. Since each agent is individual self-motivated, thus rational, it decides by itself about the kind of joining such a coalition considering its own *expected share* of the respectively divided coalition utility, i.e. the computed agent's *Shapley-value* [13][22] (see Appendix A.5).

The *decentralized calculation of each agent's utility* bases on their *local productions* resulting from the execution of own and received tasks to find some specific dependencies. One such agent utility function is defined by the amount of *task interactions* constituted by detected terminological object relations i-IDD satisfying received as well as local own FIND-task goals. Thus, for this *task interaction coalition type* each rational FCSI agent tries to locate and then get into coalition with other FCSI agents which would maximize its own utility in satisfying its FIND-tasks as much as possible.
EXAMPLE 5: see Appendix A.6

We currently investigate several different types of FCSI agent utility functions and the relationships between the induced types of FCSI coalitions. Research on the FCSI mainly aims for the description of such coalition-based cooperation for this application.

3 Conclusion

The proposed approach of the FCSI enables the user to discover some intentional relevant data without the need to browse through all available schema structures first without any help. In particular it is even not possible to get acces to the local schema or state level before any utilitarian coalition commitment with other rational agents is fixed. Context-based recognition is exclusively done by the FCSI agents at information type level, i.e. by formal terminological classification of some considered aspect term independent from its actually attached and sofar protected data structures. This is similar to the idea of incrementally building and using some shared ontology for contextual interchange[4] respecting association autonomy[24]. Thus, possible data sharing as well as proposing IDSAs is determined by the necessarily prior success and kind of such terminological aspect classification and restricted on the respective aspect valuations specified by user. The agent provides the user with all locally deducted i-IDDs and reports respective possible IDSAs. These can be used for final decisions in specification of IDDs by human[25]b.
There has been only little related research on using formal terminological classification for the object discovery problem like in [6]. Recent works on system approaches which have influenced the FCSI approach are in particular [5],[4] and [10]. Partially related works are [6],[2] and [3] [4].
Ongoing research on the FCSI includes the following main topics:
- formal FCSI agent description[28] and possible implementation with IDEAS-1
- further investigations on applying utilitarian coalition building for cooperation within the FCSI[13]

Acknowledgements: I would like to thank Prof. D. Klusch and Prof. P. Kandzia for supporting this work and giving many helpfully hints and advices.

References

[1] Baader,F., Hanschke,P., 1992,"Extensions of concept languages for a mechanical engineering application", in: LNAI 671

[2] Beck, H.W., et al., 1989,"Classification as a query processing technique in the CANDIDE SDM",IEEE Computer

[3] Barbuceanu,M., Fox,M.S., 1994,"The information agent: an infrastructure for collaboration in the integrated enterprise", Proc. CKBS-94, Keele(UK)

[4] Behrendt,W., et al., 1993,"Using an intelligent agent to mediate multibase information access",Proc. CKBS-93, Keele(UK)

[5] Bouguettaya,A., 1992,"A dynamic framework for interoperability in large MDB", Ph.D. thesis

[4] For reasons of space limitation for a detailed discussion of related works cf. [14].

[6] Catarci,T., Lenzerini,M.,"Interschema knowledge in cooperative IS", Proc. ICIS-93, Rotterdam

[7] Ceri, S., Widom, J., 1992,"Managing semantic heterogeneity with production rules and persistent queues", Politecnico Milano TR 92-078

[8] Elmagarmid, E., Zhang, A., 1992,"Enforceable interdatabase constraints in combining multiple autonomous databases", Purdue Tech. Rep. CSD-TR-92-008

[9] Genesereth,M. et al.,"Specification of the KQML Agent-Communication Language", Draft Spec. Rep. 6/1993, DARPA Knowledge Sharing Initiative EIWG

[10] Hammer,J., et al., 1993,"Object discovery and unification in FDBS", in Proc. RIDE-93, Wien

[11] Hollunder,B., Nutt,W., 1990,"Subsumption algorithms for concept languages", DFKI-Res.Rep. RR-90-04

[12] Kandzia,P., Klein,H.-J., 1993,"Theoretische Grundlagen relationaler Datenbanken", BI

[13] Ketchpel, S., 1993,"Coalition formation among autonomous agents", Proc. MAAMAW-93

[14] Klusch, M., "Ein föderatives Agentensystem FCSI zur Erkennung von Interdatenbankabhängigkeiten", Ph.D. thesis, (in preparation)

[15] Klusch, M., 1994,"Towards a Federative System FCSI for a context-based recognition of plausible Interdatabase Dependencies", Proc. 6. GI-Workshop on 'Foundations of Databases', Bad Helmstedt

[16] Klusch, M., 1994,"Using a cooperative agent system for a context-based recognition of interdatabase dependencies", Proc. CIKM-94 Workshop on 'Intelligent Information Agents', Gaithersburg (USA)

[17] Klusch, M., 1995,"Towards a Federative System FCSI for a context-based discovery of Interdatabase Dependencies", Proc. ETCE-95, 'Knowledge-based Systems in Engineering Applications', ASME PD-Vol. 67, Houston (USA)

[18] Mueller, J. (Hrsg.), 1993, "Verteilte Künstliche Intelligenz", BI

[19] Nebel, B., 1990, "Reasoning and revision in hybrid representation systems", LNAI 422, Springer

[20] Papazoglou,M.P. et al.,1992,"An organizational framework for intelligent cooperative IS", IJICIS 1(1)

[21] Sabah, G., 1993, "Knowledge Representation and Natural Language Understanding", AICOM 6(3/4)

[22] Shechory, O., Kraus, S., 1994, "Coalition formation among autonomous agents (preliminary report)"

[23] Sheth,A., Kashyap,V., 1992,"So far schematically yet so near semantically", Proc. IFIP TC2/WG2.6

[24] Sheth,A., Larson,J.A., 1990,"Federated database systems for managing distributed, heterogeneous and autonomous DBS", ACM CS 22(3)

[25] Sheth, A., et al., 1991,"Specifying interdatabase dependencies in a MDB environment",
IEEE Computer (see also Bellcore TM-STS-018609/1)

[26] Sheth, A., et al., 1991,"On applying classification to schema integration", Proc. IEEE 1.Wshp Interop. MDBS, Kyoto(Japan)

[27] Smolka,G., Schmidt-Schauß,M., 1991,"Attributive concept description with complements", AI 48

[28] Woolridge, M., 1993, "MYWORLD: an agent-oriented testbed for DAI", Proc. CKBS-93, Keele(UK)

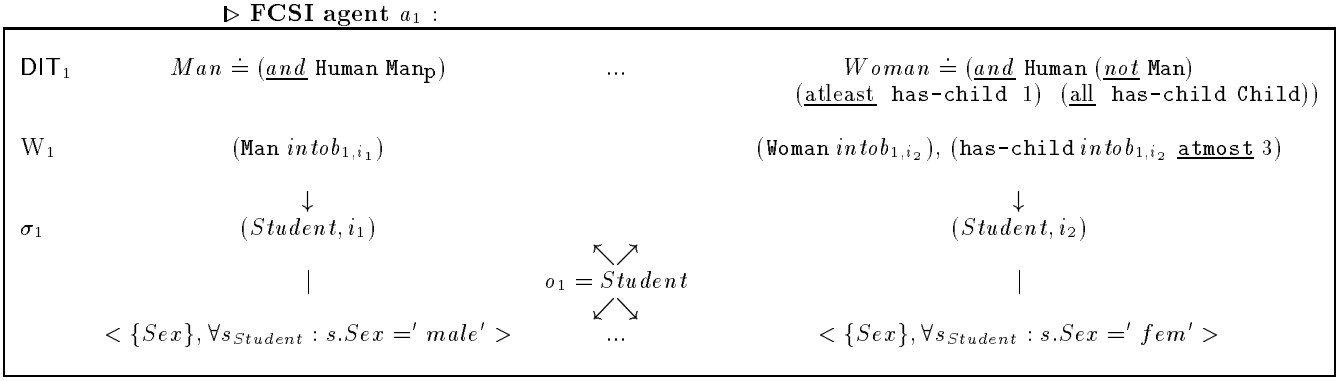
Appendix

A Simple Examples

A.1: EXAMPLE 1:

Parts of a simple local information models for two FCSI agents are shown below. In reasons of space limitation the presentation of the comprehensive local DITs is omitted.

▷ **FCSI agent** a_1 :

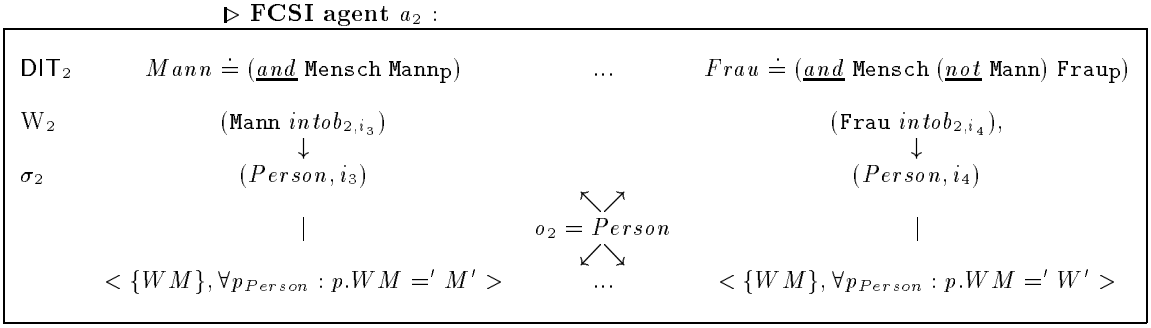


Part of respective (finite) interpretation domain for the local information model:
$intob_{1,i_1} \in RObj_1 \mapsto (Student, i_1) \in SObj_1 \times Int_1$; $intob_{1,i_2} \in RObj_1 \mapsto (Student, i_2) \in SObj_1 \times Int_1$ etc.

Then, e.g. the **aspect term** for 'female students' is

(and Humanp Womanp (not Manp)(all has − childp Childp) (atleast has − childp 1)(atmost has − childp 3))

▷ **FCSI agent** a_2 :



The **aspect term** for 'female persons' is here
(and Menschp Fraup (not Mannp)). □

A.2: EXAMPLE 2:

Assume FCSI agent a_2 receives the aspect term $intabst(Student, \{i_2\})$ from FCSI agent a_1 Let the relevant subset of corresponding primitive components in $CPC_{1,2}$ for both agents be as follows :
for primitive concept components:
$(\mathtt{Human_p}, \mathtt{Mensch_p})$, $(\mathtt{Man_p}, \mathtt{Mann_p})$, $(\mathtt{Woman_p}, \mathtt{Frau_p})$, $(\mathtt{Child_p}, \mathtt{Kind_p})$
for primitive role components:
$(\mathtt{has-child_p}, \mathtt{hat-Kind_p})$, *etc.*
Then, for example one of the following three cases can be detected by agent a_2:
$\{i_2\} \subset intset_1$, $\{i_4\} \subseteq intset_2$: $\mathbf{p-intsub}(Student, Person, \{i_2\}, \{i_4\})$, or
$\{i_2\} = intset_1$: $\mathbf{c-intsub}(Student, Person, \{i_2\}, \{i_4\})$, or
$\{i_2\} = intset_1$, $\{i_4\} = intset_2$: $\mathbf{intsub}(Student, Person)$. □

A.3: EXAMPLE 3:

Suppose that agent a_2 has detected the terminological IDD $\mathbf{p-intsub}(Student, Person, \{i_2\}, \{i_4\})$ (cf. A.2).
Then, the corresponding IDSA proposal is :
$\forall s_{Student},\ p_{Person} : (s.key(Student) = p.key(Person)) \Rightarrow$
$(\ s.Sex =' female' \Rightarrow p.WM =' W'\ )$ □

A.4: EXAMPLE 4:

▷ FCSI-Agent a_2:
(o_j, i_y) in relation to (o_k, i_x) owned by a_1;
$o_k = E_1 \in E_{\sigma_1},\ A \in \mathtt{avail}(E_1, i_x, a_2)$:
request-for A **from** E_1 **wrt** i_x

▷ FCSI-Agent a_1:
(E_1, i_x) valuation includes $JC^{i_x} = \forall e1_{E_1} : < ic-qual_{e1} >$
retrieve $e1.A$ from E_1 where $< ic - qual_{e1} >$;
(EER-DML)

Now, summarizing the continued simple example for both FCSI agents wrt. contextual data requests:

FCSI Agent a_1:

FIND-Task on $(Student, i_2)$

FCSI Agent a_2:

FIND-Task on $(Person, i_4)$

```
process tasks: produce terminological object relation i-IDD
coalition formation: determine production utility and coalition offer, bilateral negotiation
coalition commitment: coalition information availability and maintenance
```

avail$(Person, i_4, a_2)$: incl. $Name \in attr(Person)$

REQUEST-FOR-Task (context-based agent query): *'Names of Persons related to female Students'* $(i_2 : i_4)$
request-for $Name$ **from** $Person$ **wrt** i_4

- check $(Person, i_4)$: get $JC^{i_4} = (p.WM =' W')$;
- compile into local EER-DML query:
  retrieve $p.Name$ from $Person$ where $p.WM =' W'$

A.5: Definitions for EXAMPLE 5:

Let $\mathcal{A}$ set of n FCSI agents, $a_i \in \mathcal{A}$, $\mathcal{C} \subseteq \mathcal{A}$

Agent Utility Function	$U^{type}_{agent_{id}}(p(t^{a_k}_{tid_{x,a_j}}))$, with production $p(t^{a_k}_{tid_{x,a_j}})$
Coalition Value	$v(\mathcal{C})$: $\mathcal{P}(\mathcal{A}) \mapsto \mathcal{R}^+$, $v(\mathcal{C}) := \sum_{a_k \in \mathcal{C},\ p \in Prod_{a_k}} U_k(p)$
Self-Value of Agent a_i	$v(\{a_i\})$
Marginal Contribution of Agent a_i to Coalition $\mathcal{C}_i, a_i \notin \mathcal{C}_i$	$v(\mathcal{C}_i \cup \{a_i\}) \ - \ v(\mathcal{C}_i)$
Shapley-Value of agent a_i	$sv_C(a_i) := \frac{1}{n!} \sum_\pi (\ v(\mathcal{C}^\pi_i \cup \{a_i\}) \ - \ v(\mathcal{C}^\pi_i)\ )$
Restriction on *pairs of agent entities* for coalition formation:	$sv_{\{a_i,a_j\}}(a_i) = \frac{1}{2} v(\{a_i\} \ + \ \frac{1}{2}(v(\{a_j,a_i\}) \ - \ v(\{a_i\}))$
Individual Rationality	$sv_C(a_i) \ \geq \ v(\{a_i\})$

FIND-*task interaction*	$t^{a_i}_{tid_{1,a_j}} \ \underline{^{<intrel>(o_k,o_j,M,N)}} \ t^{a_i}_{tid_2} :\Leftrightarrow$
	i-IDD $<intrel>$ satisfies the Task-Goal Part tg of <u>both</u> FIND-Tasks
$\mathtt{C}_{ti}$-*utility function* U^{ti}_k	$U^{ti}_k(p(t^{a_k}_{tid_{x,a_j}})) :=\mid \{t^{a_k}_{tid_{x,a_j}} \ \underline{^{<ir>(o_k,o_j,M,N)}} \ t^{a_k}_{tid_y}\} \mid \in N_0$
$\mathtt{C}_{ti}$-*coalition value*	$v_{ti}(\mathcal{C}) = \sum_{a_k \in \mathcal{C}, p \in \mathcal{P}_{a_k}} U^{ti}_k(p)$

A.6: EXAMPLE 5:

Consider now the FCSI agents $\{a_1, a_2\} = \mathcal{A}$ as in the examples above, and suppose both aspect terms $intabst(Student, i_2)$, $intabst(Person, i_4)$ as task terms of mutually exchanged and thus respectively received FIND-tasks $t^{a_1}_{y,a_2}$, $t^{a_2}_{x,a_1}$.
Then, according to their respective local knowledge about each other, both agents are able to determine FIND-*task interactions*, e.g. $t^{a_1}_{y,a_2} \ \underline{^{p-intsub(Student,Person,\{i_2\},\{i_4\})}} \ t^{a_1}_x$ by agent a_1.
Concerning the decentralized coalition value calculations, assume that $v_{ti}(\{a_1\}) = 0$ and $v_{ti}(\{a_2\}) = 3$, i.e. only agent a_2 can satisfy 3 of its own FIND-tasks exclusively by itself through considering all reflexive task-interactions induced by dependencies relating local objects. This yields $v_{ti}(\{a_1, a_2\}) = v_{ti}(\{a_1\}) + v_{ti}(\{a_2\}) + U^{ti}_1(\{p - intsub(Student, Person, \{i_2\}, \{i_4\})\}) + U^{ti}_2(\{p - intsub(Student, Person, \{i_2\}, \{i_4\})\}) = 0 + 3 + 1 + 1 = 5$, thus for the marginal contribution of a_1 to $\{a_2\}$: $5 - 3 = 2$ and in turn for a_2: 5 - 0 = 5, which leads to the agents' Shapley-values $sv_{\{a_1,a_2\}}(a_1) = 1$, $sv_{\{a_1,a_2\}}(a_2) = 4$. Since their individual rationality is fulfilled and since no better offer from other agents exists, both agents try to coalesce with each other. They are then mutually committed to get access to all respective schema aspect valuations by some now executable REQUEST-FOR tasks. □

Logical and Computational Properties of the Description Logic MIRTL

P. Buongarzoni, C. Meghini, R. Salis, F. Sebastiani and U. Straccia
Istituto di Elaborazione dell'Informazione
Consiglio Nazionale delle Ricerche
Via S. Maria, 46 - 56126 Pisa (Italy)
E-mail: {*lastname*}@iei.pi.cnr.it

1 Introduction

In recent years a number of positive (i.e. tractability and decidability) results have been found concerning the computational complexity of Description Logics (DLs) [Buchheit *et al.*,1993; Donini *et al.*,1991; Donini *et al.*,1992a; Donini *et al.*,1992b; Schmidt-Schauß and Smolka,1991; Sebastiani and Straccia,1991]. Unfortunately, also negative results have appeared, e.g. showing that some DLs (e.g. NIKL [Patel-Schneider,1989] and KL-ONE [Schmidt-Schauß,1989]) are undecidable. This work contributes to this latter literature by showing a negative result for another DL (called MIRTL) which had not been previously studied from the standpoint of computational complexity, and which is not directly related to the ones already shown undecidable; in more standard DL terminology, MIRTL is the $\mathcal{ALEN}$ logic plus the $\mathcal{I}$ operator for inverse roles and the $\mathcal{O}_1$ operator for singleton concepts[1].

We have recently been investigating the use of MIRTL for modelling multimedia information retrieval (see [Meghini *et al.*,1993; Sebastiani,1994]). Given the requirements imposed by this application domain, and given the well-known negative results on the computational complexity of more powerful DLs, we had deemed MIRTL the best compromise between expressivity and tractability for our purposes. Somehow encouraged by the fact that the standard reasoning problems in the related logics $\mathcal{ALCNR}$ [Buchheit *et al.*,1993], $\mathcal{ALCO}$ [Schaerf,1994] and $\mathcal{ALNI}$ (also known as $\mathcal{PL}_1$) [Donini *et al.*,1991] are all decidable, and considering that the well-known algorithms based on constraint propagation for reasoning on them tend to be easily customizable to a chosen set of operators, we had thought that developing a sound and complete algorithm for MIRTL could be reasonably straightforward.

Unfortunately, while trying to develop such an algorithm, we discovered that MIRTL does not have the *finite model property*: i.e. there are satisfiable MIRTL concepts (and assertions, and KBs) which are satisfiable only in interpretations of infinite cardinality. Although this result is not one of full-blown undecidability[2], it however casts a shadow on the possibility of making practical use of such a logic. In fact, since the constraint propagation algorithms by now standard in the field of DLs prove the satisfiability of a concept by building a finite model of the concept, these algorithms are not applicable to MIRTL unless one renounces to guaranteed termination, or unless one builds some non-trivial loop-detecting control structure into them.

Once we abandon the idea of checking the satisfiability of a concept by building a finite model for it, the problem arises whether an alternative method exists or not. We have thus tried to find an upper bound to the complement of the satisfiability problem, i.e. to find a threshold under which possible inconsistencies should necessarily show up and above which no new inconsistency could emerge any more. This work reports on some negative results we have obtained in this direction, and which might constitute a prelude to a true undecidability result[3].

2 MIRTL admits infinitary concepts

The language of MIRTL includes primitive concepts (A), negation of primitive concepts ($\neg A$), concept conjunction ($C \sqcap D$), universal quantification ($\forall R.C$), qualified existential quantification ($\exists R.C$), number restrictions (($\geq n\ R$) and ($\leq n\ R$)), inversion of roles (R^{-1}) and singleton concepts ($\{a\}$). We will also use the notation ($= n\ R$) in place of ($\geq n\ R$)$\sqcap$($\leq n\ R$), and the notation $f(R).C$ in place of ($=1\ R$)$\sqcap(\forall R.C)$. Finally, MIRTL allows assertions $C[a]$ and $R[a,b]$, where C is a concept, R is a role and a, b are individual constants; $C[a]$ states that a is an instance of C, whereas $R[a,b]$ states that

[1] This latter operator is a restricted form of the $\mathcal{O}$ operator (also known as **one-of**); the extension of a singleton concept $\{a\}$ is just the singleton containing the individual denoted by a.

[2] There are in fact decidable logics that do not have the finite model property; see e.g. [Hughes and Cresswell,1984, page 154] or [Vardi and Wolper,1986].

[3] Until recently we actually thought we had an undecidability proof for MIRTL, based on a reduction of the halting problem for a model of computation equivalent to Turing Machines; the proof was then shown to contain a mistake by Diego Calvanese, a mistake that we have not been able to fix yet.

$\langle a, b\rangle$ is an instance of R. The semantics of these expressions is standard (see e.g. [Donini *et al.*,1992a]), so it will be omitted here.

Consider now the following MIRTL concept:

$$
\begin{array}{ll}
 & \{z\} \sqcap (\leq 0\ S^{-1}) \ \sqcap \\
(1) & f(S).(f(G^{-1}).\{z\}) \ \sqcap \\
 & \forall G.(f(S).((=\ 1\ S^{-1}) \ \sqcap \ f(G^{-1}).\{z\})
\end{array}
$$

The effect of concept (1) is to state some properties of the natural number zero (z); for instance, it states that z has no predecessor (S^{-1}) and a unique successor (S), and that all the numbers greater than z have a unique successor, which in turn is greater than z and has a unique predecessor.

Theorem 2.1 *Concept (1) admits only infinite models.*

Concept (1) is satisfiable, as it admits the following infinite model:

$$
\begin{array}{l}
\mathcal{D} = \{0, 1, 2, 3, \ldots\} \\
z^{\mathcal{I}} = \{0\} \\
S^{\mathcal{I}} = \{\langle 0, 1\rangle, \langle 1, 2\rangle, \langle 2, 3\rangle, \ldots\} \\
G^{\mathcal{I}} = \{\langle 0, 1\rangle, \langle 0, 2\rangle, \langle 0, 3\rangle, \ldots\}
\end{array}
$$

It can also be shown that if an interpretation $\mathcal{I}$ is a model of concept (1), then $\mathcal{I}$ must be defined on an infinite domain $\{z^{\mathcal{I}}, d_1, \ldots, d_n, \ldots\}$ such that the extension of G contains $\{\langle z^{\mathcal{I}}, d_1\rangle, \ldots, \langle z^{\mathcal{I}}, d_n\rangle, \ldots\}$ and the extension of S contains $\{\langle z^{\mathcal{I}}, d_1\rangle, \langle d_1, d_2\rangle, \ldots, \langle d_{n-1}, d_n\rangle, \ldots\}$.

This shows the existence of MIRTL concepts which are satisfied only in interpretations with infinite domain; we will call them *infinitary concepts*. Similar concepts had already been shown to exist for other DLs, albeit substantially more expressive [Schild,1991; Calvanese *et al.*,1994].

The presence of infinitary concepts has the practical consequence that the standard methods based on constraint propagation (e.g. the methods discussed in [Schmidt-Schauß and Smolka,1991; Buchheit *et al.*,1993]) do not work properly; since these methods attempt to build a model of the concept they want to prove consistent, once applied to infinitary concepts they loop forever, unless they are endowed with a control structure able to detect the construction of an infinitely self-replicating structure (which every MIRTL infinitary concept, from what our experiments have shown, seems to contain).

The existence of MIRTL infinitary concepts has a computational counterpart in what might be called *self-reactivating constraints*: as a consequence of the interaction among MIRTL propagation rules (in particular, the rules for universal quantification, role inversion and singleton), some constraints may be such that the rules that process them generate the same activating conditions that obtained before the application of the rule (thus inducing an infinite loop).

After having noticed this, in our search for a decidability result for MIRTL, we had hoped to find, nonetheless, an upper bound on the number of steps necessary to find a clash deriving from an inconsistent MIRTL concept. If we had found such an upper bound, say k, decidability would have been established, as a MIRTL concept could be declared satisfiable after the k-th step had failed to produce a clash. Unfortunately, this strategy proved ineffective: for every $f(n)$ (where n is the size of the concept) that we had, in repeated attempts, conjectured to be such an upper bound, we were able to discovered an unsatisfiable MIRTL concept which generated a clash well after $f(n)$ steps. As we will see in the next section, this has happened for functions f of increasing order of magnitude.

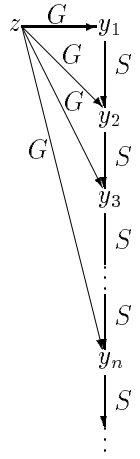


Figure 1: (Partial) graphical representation of the constraints generated by Concept (1).

3 Looking for an upper bound

Once we abandon the idea of checking the satisfiability of a concept by building a model for it, the problem arises whether an alternative method exists or not. To this end, we have studied the computational behaviour of various categories of infinitary concepts. For instance, if we graphically represent a constraint system as a directed graph, in which nodes represent objects (individual constants or variables) and the set of concepts constraining them, and edges represent binary assertions between them (the edge is oriented from the first element of the assertion to the second), concept (1) generates a graph that contains an infinite subgraph like the one shown in Figure 1).

By analyzing the structure of the constraint sets that infinitary concepts generate, we have noticed that the generated graph has the following structural property: it is composed by a "kernel" subgraph, in which all the constants appearing in the concept and some variables occur, to which other subgraphs are connected that repeat *ad infinitum* portions of the kernel subgraph. This shows that the constraint propagation process, after a

finite number of de-activations of the same ∀-constraint, completes the construction of the kernel subgraph and starts building an infinite number of replicas of portions of it.

After observing this, we have tried to individuate the step of the constraint propagation process at which the building of the kernel subgraph is completed and the building of the replicas start. In fact, once this step were individuated, the problem would be solved: in fact, it would suffice to check if the kernel subgraph (plus a small number of replicas) contained an inconsistency, as, if this were not the case, also the replicas would not contain any inconsistency, and the concept could be deemed satisfiable without any further rule application.

In other words, what we have been looking for is an upper bound to the complement of the satisfiability problem (co-**CS**), i.e. a threshold under which possible inconsistencies should necessarily emerge and above which no new inconsistency could possibly emerge. We have tried to find such an upper bound as an upper bound on the number of de-activations of the same constraint, expressed in terms of the size d of the initial concept. In fact, as we have already argued, the cause of non-termination of the computation is the re-activation of a constraint. Hence, we have made the hypothesis that, by limiting this number, one could limit the generation *ad infinitum* of replicas of portions of the kernel subgraph. We then started an empirical study in order to identify the maximum number of activations of the same constraint beyond which the procedure diverges and builds an infinite model.

3.1 Concepts denoting natural numbers (or: the d limit)

In Section 2 we have seen that the procedure of constraint propagation diverges when applied to concept 1. By examining the constraint system generated after $h < d$ de-activations of the self-reactivating constraint

$$(\forall G.(f(S).((=\ 1\ S^{-1})\ \sqcap\ f(G^{-1}).\{z\})))[z]$$

one can verify that no clash occurs in it. Within d activations, inconsistencies due to arbitrarily nested concepts like the following, are discovered:

$$\begin{array}{ll} & ((\forall\ R.((\exists\ P.(\exists\ R^{-1}.\{i\}))\ \sqcap \\ (2) & (\forall\ P^{-1}.(\forall\ P^{-1}.(\forall\ P^{-1}.(\forall\ P^{-1}.C')))))\ \sqcap \\ & (\exists\ R)\ \sqcap\ \{i\}) \end{array}$$

The peculiarity of this case comes from the depth of the syntactic tree of concept 2; the subconcept C' is to be found at depth level 7. In order for C' to constrain an individual, it is necessary that the longest branch of the syntactic tree is completely explored. At this point, C' constrains the individual constant i, and from the development of $C'[i]$ a clash may arise (e.g. if $C' = (\leq\ 0\ R)$), or the infinite generation of variables may be blocked ($C' = (\leq\ n\ R)$), or still other things may happen.

3.2 Concepts denoting the multiples of a natural number (or: the d^2 limit)

The d limit has resisted numerous empirical tests, until we have been able to find a MIRTL concept denoting the set of the multiples of a natural number. For instance, the set of multiples of 2 may be represented by the following MIRTL concept:

$$\begin{array}{lll} (0)\ C_3 = & (\{z\}\ \sqcap & \\ (1) & (\leq\ 0\ E^{-1})\ \sqcap & \\ (2) & (\leq\ 0\ O^{-1})\ \sqcap & \\ (3) & (\leq\ 0\ E)\ \sqcap & \\ (4) & (=\ 1\ O)\ \sqcap & \\ (5) & (\exists\ M.\{t\})\ \sqcap & \\ (6) & (\exists\ G.(\exists\ O^{-1}.\{z\}))\ \sqcap & \\ (7)(7') & (\forall\ G.((\forall\ O^{-1}.(\forall\ O.((=\ 1\ O^{-1})\ \sqcap & \\ & \qquad (\leq\ 0\ O)\ \sqcap & \\ & \qquad (\leq\ 0\ E^{-1})\ \sqcap & \\ & \qquad (=\ 1\ E)\ \sqcap & \\ & \qquad (\forall\ E.((\exists\ G^{-1}.\{z\})\ \sqcap & \\ & \qquad\qquad (\exists\ M.\{t\})))))))\ \sqcap & \\ (7'') & \qquad (\forall\ E^{-1}.(\forall\ E.((=\ 1\ E^{-1})\ \sqcap & \\ & \qquad (\leq\ 0\ E)\ \sqcap & \\ & \qquad (\leq\ 0\ O^{-1})\ \sqcap & \\ & \qquad (=\ 1\ O)\ \sqcap & \\ & \qquad (\forall\ O.(\exists\ G^{-1}.\{z\})))))))) & \end{array}$$

If we interpret the roles O, E, M e G as the "odd successor", "even successor", "multiple-of" and "bound above by", one may see that C_3 denotes the set of even numbers; in fact, starting from the individual constant z and generating variables y_1, y_2, ..., only variables with an even subscript are put in relation with the the individual constant t. In fact, the first sub-concepts state that the natural number 0 (subconcept 0) is neither the even successor of a number (1), nor its odd successor (2); also, 0 does not have any even successor (3), has exactly an odd successor (4), is a multiple of 2 (6). Subconcept (7) (i.e. the one which generates the self-reactivating constraint) states that the odd successors of their predecessor have an even successor which is both a multiple of two and a positive number, while the even successors of their predecessors have an odd successor which is positive. In this way, a set of constraints is generated which is partially represented in Figure 2.

Each relation of type M is generated after two activations of the constraint $(\forall\ G\ \ldots)$. In a similar way, it would be possible to describe the multiples of 3, 4, etc..

Note that in the self-reactivating constraint (7) the disjunction of two concepts is simulated by means of two occurrences of the ∀ operator; in fact, of the two subconcepts:

(7') $(\forall\ O^{-1}.(\forall\ O.(\ldots)))$
(7") $(\forall\ E^{-1}.(\forall\ E.(\ldots)))$

only one is activated when they are applied to the same variable. The activation depends on the subscript of the variable: variables with even subscript k represent the even successor of the $k-1$ variable; variables with

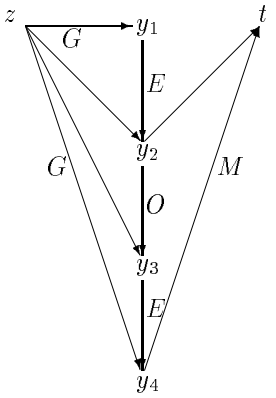


Figure 2: Graphical representation of the set of constraints denoting the multiples of 2.

odd subscript k represent the odd successor of the same variable.

In order to describe the multiples of the numbers greater than 2 it is sufficient to modify concept C_3 by changing roles E and O into S_0 and S_1, and adding as many roles S_j and as many subconcepts $(\forall\ S_j^{-1}.(\forall\ S_j \ldots .)))$ as the number whose successors we want to represent (S_j stands for *"successor modulo j"*).

This concept, which is in itself satisfiable, can be used in order to write unsatisfiable concepts whose first clash occurs after d activations of a constraint. In fact, if the individual constant t were constrained by concept $(\leq\ k\ M^{-1})$, for some k, the clash would be generated only after $n(k+1)-1$ de-activations of the self-reactivating constraint, where n is the number of subconcepts of type $(\forall\ S_j^{-1}.(\forall\ S_j \ldots .)))$ of the self-reactivating constraint. As the numbers n and k have the same order of magnitude of the dimension d of the original concept, the first clash will appear in the constraint system before $O(d^2)$ steps but after $O(d)$ steps.

3.3 Concepts denoting the first n powers of a number (or: the d^d limit)

If we substitute, within concept 1 the self-reactivating subconcept:

$$(\forall G.(f(S).((=\ 1\ S^{-1})\ \sqcap\ f(G^{-1}).\{z\})$$

by

$$\begin{array}{l}(\forall\ G.f(S).((=\ 1\ S^{-1})\ \sqcap \\ \qquad f(G^{-1}).\{z\}\ \sqcap \\ \qquad (\exists\ R_1.((\leq\ n\ R_1^{-1})\ \sqcap \\ \qquad\quad (\exists\ R_2.((\leq\ n\ R_2^{-1})\ \sqcap \\ \qquad\quad\quad (\exists\ R_3.((\leq\ n\ R_3^{-1})\ \sqcap \\ \qquad\quad\quad\quad \{t\})))))))\end{array}$$

we obtain an unsatisfiable concept, whose first clash appears within the constraint system after d^2 activations of the self-reactivating constraint. In fact, during the application of the propagation rules, the subconcept $(\exists\ R_1$ $\ldots)$ generates within the graph of Figure 3 a path of length 3 ending with the individual constant t. When a

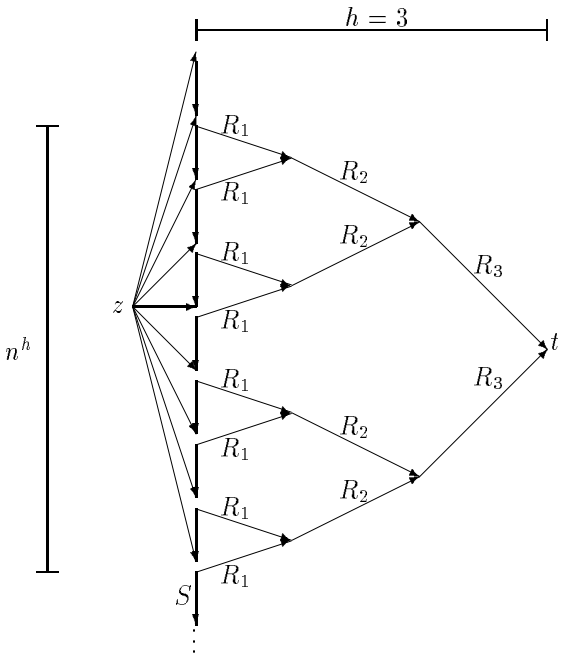


Figure 3: Graphical representation of the constraints denoting the first power of 2.

path for every variable y_j is generated, a tree is created with root t and depth 3. As the individual constant t is constrained by concept $(\leq\ n\ R_3^{-1})$, when variable y_{n^3+1} is generated (at the n^3+1-th disactivation of the $(\forall\ G \ldots)$ constraint, the first clash appears within the constraint system. Adding a role to path R_1, R_2, ...corresponds to incrementing by one the depth of the tree, and consequentially incrementing by one the value h of the exponent; instead, modifying the value n of the number restrictions corresponds to modify the width of the tree, and consequentially modifying the base n. Obviously, both h and n are strictly smaller than d, but still remain of the same order of magnitude; hence, the number of de-activations of a constraint remains exponential in the dimension d of the original concept C.

It is also possible to combine these two latter sources of complexity (multiples of a number (d^2) and powers of a number (d^d)), by generating the role R_1 only for the multiples of a number; this brings the maximum number of activations to $d \cdot d^d$.

4 Conclusions

In this work we have shown that MIRTL does not enjoy the finite model property, and have discussed some consequences of this fact. In particular, we have argued that, even if MIRTL should be proven decidable (which is still, to the best of our knowledge, an open problem), reasoning on it by means of constraint propagation algorithms is probably going to be computationally onerous, as MIRTL concepts may be written which generate clashes only after a computationally unfeasible number of steps.

Acknowledgements

We are grateful to Diego Calvanese for his detailed comments to an earlier draft (see Footnote 3), and for giving us interesting pointers to the literature.

References

[Buchheit *et al.*, 1993] Martin Buchheit, Francesco M. Donini, and Andrea Schaerf. Decidable reasoning in terminological knowledge representation systems. *Journal of Artificial Intelligence Research*, 1:109–138, 1993.

[Calvanese *et al.*, 1994] Diego Calvanese, Maurizio Lenzerini, and Daniele Nardi. A unified framework for class-based representation formalisms. In *Proceedings of KR-94, 5th International Conference on Principles of Knowledge Representation and Reasoning*, pages 109–120, Bonn, FRG, 1994.

[Donini *et al.*, 1991] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Werner Nutt. Tractable concept languages. In *Proceedings of IJCAI-91, 12th International Joint Conference on Artificial Intelligence*, pages 458–463, Sidney, Australia, 1991.

[Donini *et al.*, 1992a] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, Werner Nutt, and Andrea Schaerf. Adding epistemic operators to concept languages. In *Proceedings of KR-92, 3nd International Conference on Principles of Knowledge Representation and Reasoning*, pages 342–353, Cambridge, MA, 1992.

[Donini *et al.*, 1992b] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Andrea Schaerf. From subsumption to instance checking. Technical Report 15.92, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", Roma, Italy, 1992.

[Hughes and Cresswell, 1984] George E. Hughes and Maxwell J. Cresswell. *A companion to modal logic.* Methuen, London, UK, 1984.

[Meghini *et al.*, 1993] Carlo Meghini, Fabrizio Sebastiani, Umberto Straccia, and Costantino Thanos. A model of information retrieval based on a terminological logic. In *Proceedings of SIGIR-93, 16th ACM International Conference on Research and Development in Information Retrieval*, pages 298–307, Pittsburgh, PA, 1993.

[Patel-Schneider, 1989] Peter F. Patel-Schneider. Undecidability of subsumption in NIKL. *Artificial Intelligence*, 39:263–272, 1989.

[Schaerf, 1994] Andrea Schaerf. Reasoning with individuals in concept languages. *Data and Knowledge Engineering*, 13:141–176, 1994.

[Schild, 1991] Klaus Schild. A correspondence theory for terminological logics: preliminary report. In *Proceedings of IJCAI-91, 12th International Joint Conference on Artificial Intelligence*, pages 466–471, Sidney, Australia, 1991.

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

[Schmidt-Schauß, 1989] Manfred Schmidt-Schauß. Subsumption in KL-ONE is undecidable. In *Proceedings of KR-89, 1st International Conference on Principles of Knowledge Representation and Reasoning*, pages 421–431, Toronto, Ontario, 1989.

[Sebastiani and Straccia, 1991] Fabrizio Sebastiani and Umberto Straccia. A computationally tractable terminological logic. In *Proceedings of SCAI-91, 3rd Scandinavian Conference on Artificial Intelligence*, pages 307–315, Roskilde, Denmark, 1991.

[Sebastiani, 1994] Fabrizio Sebastiani. A probabilistic terminological logic for modelling information retrieval. In *Proceedings of SIGIR-94, 17th ACM International Conference on Research and Development in Information Retrieval*, pages 122–130, Dublin, IRL, 1994.

[Vardi and Wolper, 1986] Moshe Vardi and P. Wolper. Automata-theoretic techniques for modal logics of programs. *Journal of Computer and Systems Science*, 32:183–221, 1986.

Making $\mathcal{CATS}$ out of kittens: description logics with aggregates

Giuseppe De Giacomo and Maurizio Lenzerini
Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, 00198 Roma, Italia
{degiacomo,lenzerini}@assi.dis.uniroma1.it

Abstract

Based on the research done in the last decade, attempts have been made to propose description logics as unifying formalisms for the various class-based representation languages used in different areas. These attempts have made apparent that sound, complete, and decidable description logics still suffer from several limitations, regarding modeling classes of aggregate objects, expressing general inclusion axioms, and the ability of navigating links between classes. In this paper, we propose a powerful description logic overcoming the above limitations and we show that its reasoning tasks are decidable in worst case exponential time.

1 Introduction

Description logics are AI formalisms that allow one to represent domain knowledge by focusing on classes of objects [Brachman,1977] and their relationships [Woods,1975], and offering specialized inferences on the class structure.

The research developed in the last decade offers a quite complete picture of several issues related to the expressive power of the logics and the computational complexity of the reasoning tasks (see [Woods and Schmolze,1992]). Based on the outcome of this research, attempts have been made to propose description logics as unifying formalisms for the various class-based representation languages used in different areas, such as semantic networks, feature logics, conceptual and object-oriented database models, type systems, and other formalisms used in software engineering [Bergamaschi and Sartori,1992; Piza *et al.*,1992; Borgida,1992; Calvanese *et al.*,1994; Schreiber *et al.*,1993]. However, these attempts have made apparent that description logics equipped with sound, complete, and terminating reasoning procedures still suffer from several limitations that are not acceptable when representing complex domains in the different fields mentioned above. Here is a list of the most important limitations.

- The domain of interpretation is flat, in the sense that the logics consider the world as constituted by elementary objects (grouped in concepts) and binary relations between them. One consequence of this property is that N-ary relations are not supported (an exception is the logic proposed in [Schmolze,1989], for which no complete decision procedure was proposed). In fact, N-ary relations have been shown to be important in several contexts (see [Catarci and Lenzerini,1993]), especially in databases and natural language. For example, 'exam' is correctly modeled as a ternary relation over 'student', 'professor' and 'course'. Note that supporting N-ary relations means that the logic offers suitable mechanisms for their definition and characterization. For example, one has to ensure that no pair of 'exam' instances connect the same triple of objects; also, one may want to assert that students linked to graduate courses by the relation exam are graduate students. These kinds of properties cannot be represented by simply modeling the N-ary relation in terms of N binary relations.

- Usually, general inclusion axioms are not supported. Although inclusion axioms are essential when we want to assert properties of classes and relations, as required in complex domains, most of the research on description logics either deals with class descriptions only, or impose severe restrictions, such as acyclicity, on axioms. Exceptions are, for example, [Nebel,1991; Baader,1991; Schild,1991; De Giacomo and Lenzerini,1994; Buchheit *et al.*,1993]. An important outcome of this research is that reasoning with axioms is computationally hard, even for the simplest description logics (weaker than $\mathcal{FL}^-$). All these works, however, limit their attention to axioms on concepts, and do not consider the problem of expressing inclusion axioms on relations.

- Relationships between classes are generally described by means of poor representation mechanisms. In fact, when trying to use description logics for capturing representation formalisms used in different fields, one realizes that at least three features are essential: the ability of *navigating* relationships (say of a semantic network or an entity-relationship schema) in both directions; the ability of stating cardinality constraints of general forms on relationships; the possibility of conceiving relationships as sets, thus applying set theoretic operators on

them (including the notorious role value map [Woods and Schmolze,1992]).

The aim of the present work is to devise a description logic, called $\mathcal{CATS}$, that finally addresses the above issues. The basic ingredients of $\mathcal{CATS}$ are classes and links. In contrast to traditional description logics, *classes* are abstractions not only for a set of individuals (corresponding to the usual notion of concept, called simple class here), but also for sets that have aggregates as instances (called aggregate classes). There are two types of aggregates: property aggregates and instance aggregates. A *property aggregate* is an abstraction for an object that is considered as an aggregation of other objects, one for each attribute belonging to a specified set [Smith and Smith,1977]. A typical example of such an aggregate is a date, which is seen as an aggregation of three objects, one for the attribute day, one for the attribute month, and one for the attribute year. Another example of property aggregate is an exam, which again is seen as an aggregation of three objects (one professor, one student and one course). This makes clear that N-ary relations can be modeled as classes whose instances are aggregates. An *instance aggregate* is an abstraction of a group of other objects belonging to a certain class [Brodie and Ridjanovic,1984]. A typical example of such an aggregate is a team, which can be seen as a group of players. Like any other description logics, $\mathcal{CATS}$ allows one to form *complex classes* by applying suitable constructors to both simple and aggregate classes. Notably, $\mathcal{CATS}$ includes a form of role value map, and the most general form of number restrictions (called qualified).

Links are abstractions for atomic, basic, and complex relationships between classes. An *atomic link* (denoted simply by a name, and also called attribute) is the most elementary mean for establishing a relationship between classes. A *basic* link is formed by applying certain constructors (like inverse, union, intersection and difference) to atomic links. A *complex* link is formed by applying more complex constructors (like chaining, transitive closure, and identity) to basic links.

A knowledge base in $\mathcal{CATS}$ is simply a set of inclusion axioms. We point out that $\mathcal{CATS}$ allows inclusion assertions to be stated on classes of all kinds (simple, aggregate and complex), and on basic links, with no limitation (for example on cycles). A particular care is put in devising $\mathcal{CATS}$ so that its reasoning tasks remain decidable and even with the same computational complexity as the simplest description logics where inclusion axioms are allowed. Indeed, making use of the results in [De Giacomo and Lenzerini,1994], we have proved that computing logical implication (and satisfiability) in $\mathcal{CATS}$, is both EXPTIME-hard and decidable with exponential time in the worst case.

2 The description logic $\mathcal{CATS}$

As we said above, the language of $\mathcal{CATS}$ supports classes and links. Classes are partitioned into simple classes and aggregate classes, which are further distinguished in property aggregate and instance aggregate classes. Links are partitioned into atomic (also called attributes), basic, and complex.

Let a nonempty finite alphabet $\mathcal{A}$ of atomic classes (classes denoted simply by a name, no matter if simple or aggregate), and a nonempty finite alphabet $\mathcal{U}$ of attributes be available. We use A for a generic element of $\mathcal{A}$, U (possibly with subscript) for a generic element of $\mathcal{U}$, C (possibly with subscript) for a generic class, b (possibly with subscript) for a generic basic link, and L (possibly with subscript) for a generic complex link. The language of $\mathcal{CATS}$ has the following syntax ($n, k \geq 1$):

$$
\begin{array}{lcl}
C & ::= & A \mid \tau(U_1, \ldots, U_n) \mid \chi(C, U_1, \ldots, U_n) \mid \sigma(C) \mid \\
 & & C_1 \sqcap C_2 \mid \neg C \mid \forall L.C \mid (\leq k\, b.C) \mid (\leq k\, b^-.C) \mid \\
 & & (b_1 \subseteq b_2) \mid (b_1^- \subseteq b_2^-) \\
b & ::= & U \mid \ni \mid b_1 \cup b_2 \mid b_1 \setminus b_2 \\
L & ::= & b \mid L_1 \circ L_2 \mid L_1 \cup L_2 \mid L^* \mid L^- \mid id(C)
\end{array}
$$

We use a (possibly with subscript) for b and b^-, and we adopt the following abbreviations: $\top \doteq A \sqcup \neg A$, $\bot \doteq \neg\top$, $\tau \doteq \tau(U_1) \sqcup \cdots \sqcup \tau(U_m)$ (where $\{U_1, \ldots, U_m\} = \mathcal{U}$), $\sigma \doteq \sigma(\top)$, $C_1 \sqcup C_2 \doteq \neg(\neg C_1 \sqcap \neg C_2)$, $\exists L.C \doteq \neg\forall L.\neg C$, $\emptyset \doteq \ni \setminus \ni$, $(\geq k\, a.C) \doteq \neg(\leq k+1\, a.C)$, $a_1 \cap a_2 \doteq a_1 \setminus (a_1 \setminus a_2)$, and $(a_1 = a_2) \doteq (a_1 \subseteq a_2) \sqcap (a_2 \subseteq a_1)$. Parentheses are used to disambiguate expressions.

The semantics for the language of $\mathcal{CATS}$ is based on an *interpretation* $\mathcal{I} = (\mathcal{O}^\mathcal{I}, \cdot^\mathcal{I})$, where $\mathcal{O}^\mathcal{I}$ is the *universe* of the interpretation, and $\cdot^\mathcal{I}$ is the *interpretation function* over such a universe. Differently from the usual notion of interpretation, $\mathcal{O}^\mathcal{I}$ is a nonempty set of *polymorphic objects*, which means that every object in $\mathcal{O}^\mathcal{I}$ has none, one, or both of the following two forms:

1. The form of *tuple*: when an object has this form, it can be considered as a property aggregation, which is formally defined as a partial function from $\mathcal{U}$ to $\mathcal{O}^\mathcal{I}$. We use the term tuple to denote an object in $\mathcal{O}^\mathcal{I}$ that has the form of tuple, and we write $\langle U_1 : o_1, \ldots, U_n : o_n \rangle$[1] to denote any tuple t such that, for each $i \in \{1, \ldots, n\}$, $t(U_i)$ is defined and equal to o_i (which is called the U_i-component of t). Note that the tuple t may have other components as well, besides the U_i-components.
2. The form of *set*: when an object o has this form, it can be considered as an instance aggregate, which is formally defined as a nonempty finite collection of objects in $\mathcal{O}^\mathcal{I}$, with the following proviso: the view of o as a set is unique, in the sense that there is only one finite collection of objects of which o can be considered an aggregation, and no other object o' is the aggregation of the same collection. We use the term set to denote an object in $\mathcal{O}^\mathcal{I}$ that has the form of set, and we write $\{\!|o_1, \ldots, o_n|\!\}$ to denote the collection whose members are exactly $o_1, \ldots, o_n$.

Objects having none of these forms are called elementary objects - i.e., individuals with no structure.

[1] This notation makes it clear that a tuple is indeed a function assigning one element of $\mathcal{O}^\mathcal{I}$ to some of the elements of $\mathcal{U}$.

The interpretation function $\cdot^{\mathcal{I}}$ is defined as follows:

- It assigns to $\ni$ a subset of $\mathcal{O}^{\mathcal{I}} \times \mathcal{O}^{\mathcal{I}}$ such that for each $\{\!|\ldots, o, \ldots|\!\} \in \mathcal{O}^{\mathcal{I}}$, we have that $(\{\!|\ldots, o, \ldots|\!\}, o) \in \ni^{\mathcal{I}}$.
- It assigns to every attribute U a subset of $\mathcal{O}^{\mathcal{I}} \times \mathcal{O}^{\mathcal{I}}$ such that, for each $\langle \ldots, U : o, \ldots \rangle \in \mathcal{O}^{\mathcal{I}}$, $(\langle \ldots, U : o, \ldots \rangle, o) \in U^{\mathcal{I}}$, and there is no $o' \in \mathcal{O}^{\mathcal{I}}$ different from o such that $(\langle \ldots, U : o, \ldots \rangle, o') \in U^{\mathcal{I}}$. Note that this implies that every U in a tuple is functional for the tuple.
- It assigns to every basic link a subset of $\mathcal{O}^{\mathcal{I}} \times \mathcal{O}^{\mathcal{I}}$ such that the following conditions are satisfied:

$$\begin{array}{lcl} (b_1 \cup b_2)^{\mathcal{I}} & = & b_1^{\mathcal{I}} \cup b_2^{\mathcal{I}} \\ (b_1 \setminus b_2)^{\mathcal{I}} & = & b_1^{\mathcal{I}} - b_2^{\mathcal{I}} \\ (b^-)^{\mathcal{I}} & = & \{(o, o') \mid (o', o) \in b^{\mathcal{I}}\}. \end{array}$$

- It assigns to every complex link a subset of $\mathcal{O}^{\mathcal{I}} \times \mathcal{O}^{\mathcal{I}}$ such that the usual conditions for $\circ$, $\cup$, *, $^-$, and id are satisfied:

$$\begin{array}{l} (L_1 \cup L_2)^{\mathcal{I}} = L_1^{\mathcal{I}} \cup L_2^{\mathcal{I}} \\ (L_1 \circ L_2)^{\mathcal{I}} = L_1^{\mathcal{I}} \circ L_2^{\mathcal{I}} \\ (L^*)^{\mathcal{I}} = (L^{\mathcal{I}})^* \\ (L^-)^{\mathcal{I}} = \{(o, o') \in \mathcal{O}^{\mathcal{I}} \times \mathcal{O}^{\mathcal{I}} \mid (o', o) \in R^{\mathcal{I}}\} \\ id(C)^{\mathcal{I}} = \{(o, o) \in \mathcal{O}^{\mathcal{I}} \times \mathcal{O}^{\mathcal{I}} \mid o \in C^{\mathcal{I}}\}. \end{array}$$

- It assigns to every class a subset of $\mathcal{O}^{\mathcal{I}}$ in such a way that the following conditions are satisfied ($\sharp\{\}$ denotes the cardinality of a set): - $A^{\mathcal{I}} \subseteq \mathcal{O}^{\mathcal{I}}$
  - $\tau(U_1, \ldots, U_n)^{\mathcal{I}} = \{\langle U_1 : o_1, \ldots, U_n : o_n \rangle \in \mathcal{O}^{\mathcal{I}} \mid o_1, \ldots, o_n \in \mathcal{O}^{\mathcal{I}}\}$
  - $\chi(C, U_1, \ldots, U_n)^{\mathcal{I}} = S \subseteq \tau(U_1, \ldots, U_n) \cap C^{\mathcal{I}}$ and no distinct $s, s' \in S$ have the same $U_1, \ldots, U_n$-components
  - $\sigma(C)^{\mathcal{I}} = \{\{\!|o_1, \ldots, o_n|\!\} \in \mathcal{O}^{\mathcal{I}} \mid o_1, \ldots, o_n \in C^{\mathcal{I}}\}$
  - $(C_1 \sqcap C_2)^{\mathcal{I}} = C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$
  - $(\neg C)^{\mathcal{I}} = \mathcal{O}^{\mathcal{I}} - C^{\mathcal{I}}$
  - $(\forall L.C)^{\mathcal{I}} = \{o \in \mathcal{O}^{\mathcal{I}} \mid \forall o'.(o, o') \in R^{\mathcal{I}} \supset o' \in L^{\mathcal{I}}\}$
  - $(\leq k\, a.C)^{\mathcal{I}} = \{o \in \mathcal{O}^{\mathcal{I}} \mid \sharp\{(o, o') \in a^{\mathcal{I}} \wedge o' \in C^{\mathcal{I}}\} \leq k\}$
  - $(a_1 \subseteq a_2)^{\mathcal{I}} = \{o \in \mathcal{O}^{\mathcal{I}} \mid \{o' \mid (o, o') \in a_1^{\mathcal{I}}\} \subseteq \{o' \mid (o, o') \in a_2^{\mathcal{I}}\}\}$.

A $\mathcal{CATS}$ TBox $\mathcal{K}$ is a finite set of inclusion assertions of the form $C_1 \sqsubseteq C_2$, where C_1 and C_2 are classes in $\mathcal{CATS}$ (we write $C_1 \equiv C_2$ for $C_1 \sqsubseteq C_2, C_2 \sqsubseteq C_1$). As usual, an interpretation $\mathcal{I}$ is a model of $C_1 \sqsubseteq C_2$ if $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$, and $\mathcal{K} \models C_1 \sqsubseteq C_2$ (read as $\mathcal{K}$ logically implies $C_1 \sqsubseteq C_2$), if each model of all assertions in $\mathcal{K}$ is also a model of $C_1 \sqsubseteq C_2$. As mentioned, we have the following result.

Theorem 1 *Logical implication in $\mathcal{CATS}$ is EXPTIME-complete.*

3 Discussion

Let us discuss the most important modeling capabilities of $\mathcal{CATS}$ by means of one example.

$\top \sqsubseteq (\texttt{father} \cap \texttt{mother} \subseteq \emptyset) \sqcap$
$\quad (\texttt{father} \cap \texttt{children} \subseteq \emptyset) \sqcap$
$\quad (\texttt{children} \cap \texttt{mother} \subseteq \emptyset)$
$\top \sqsubseteq \forall \texttt{father}^- \cup \texttt{mother}^- \cup \texttt{children}^-.\texttt{Family}$
$\texttt{Date} \equiv \chi(\texttt{Date}, \texttt{day}, \texttt{month}, \texttt{year})$
$\exists \texttt{date}^-.\top \sqsubseteq \texttt{Date}$
$\exists \texttt{day}^-.\top \sqsubseteq \texttt{Day}$
$\exists \texttt{month}^-.\top \sqsubseteq \texttt{Month}$
$\exists \texttt{year}^-.\top \sqsubseteq \texttt{Year}$
$\exists \texttt{city}^-.\top \sqsubseteq \texttt{City}$
$\texttt{Day} \sqcup \texttt{Month} \sqcup \texttt{Year} \sqsubseteq \neg\tau \sqcap \neg\sigma$
$\texttt{Mayor} \equiv \exists \texttt{mayor}^-.\top$
$\exists \texttt{mayor}.\top \sqsubseteq \texttt{City}$
$\texttt{City} \sqsubseteq \tau(\texttt{name}, \texttt{state}, \texttt{country}, \texttt{mayor}) \sqcap$
$\quad \chi(\texttt{City}, \texttt{name}, \texttt{state}, \texttt{country}) \sqcap \chi(\texttt{City}, \texttt{mayor})$
$\texttt{Family} \sqsubseteq \sigma(\texttt{Person}) \sqcap \tau(\texttt{father}, \texttt{mother}, \texttt{date}, \texttt{city}) \sqcap$
$\quad \chi(\texttt{Family}, \texttt{father}, \texttt{mother}, \texttt{date}) \sqcap$
$\quad (\ni = \texttt{father} \cup \texttt{mother} \cup \texttt{children})$
$\texttt{StillFamily} \sqsubseteq \texttt{Family} \sqcap \chi(\texttt{StillFamily}, \texttt{father}, \texttt{mother})$
$\texttt{PhdFamily} \equiv (\geq 3\ \ni.\texttt{PhdPerson}) \sqcap (\leq 1\ \ni.\neg\texttt{PhdPerson})$
$\texttt{Person} \sqsubseteq (\exists \texttt{children}^-.\top) \sqcap (\leq 1\, \texttt{children}^-.\top)$
$\texttt{ChildOfMayor} \equiv \exists \texttt{children}^- \circ \texttt{father}.\texttt{Mayor}$
$\texttt{VeryPhd} \equiv \forall(\texttt{children}^- \circ (\texttt{father} \cup \texttt{mother}))^*.\texttt{PhdPerson}$

Figure 1: Families, persons, and cities

Figure 1 shows a TBox $\mathcal{K}$ modeling a world with persons, families and cities. The following observations help understanding the expressive power of $\mathcal{CATS}$.

- Objects are polymorphic. For example, every instance of **Family** (representing families resulting from a marriage) can be seen both as a set of persons, and as a tuple with attributes **father**, **mother**, **date** (of marriage) and **city** (of marriage). Note, however, that assertions can be used to impose that the instances of a certain class (**Day**, **Month** and **Year** in our example) can only be seen as elementary objects.
- Inclusion assertions on classes are used with no limitation. In particular, they can be stated for all kinds of classes, and cycles are allowed in the TBox. Notably, inclusion assertions can also be stated for basic links: indeed, $\top \sqsubseteq (b_1 \subseteq b_2)$ forces b_1 to a subset of b_2 in every model of $\mathcal{K}$. Inclusion assertions of this kind are used in the example to specify the properties of the attributes **father**, **mother** and **children**.
- N-ary relations are supported. Any instance of **Family** can indeed be considered as a relation with four arguments. The χ constructor is used to define keys for (N-ary) relations: for example, the fact that every instance of **Family** is an instance of χ(**Family**, **father**, **mother**, **date**) implies that the three attributes form a key for the class. On the other hand, **StillFamily**, representing families whose father and mother are still married, has a more specialized key, constituted by the attributes **father** and **mother**. Observe that several keys can

be defined for a class (see `City`).

- Qualified number restrictions and role value maps on basic links can be used without any limitation. Indeed, $(\ni = \texttt{father} \cup \texttt{mother} \cup \texttt{children})$ is a role value map on basic links.
- Complex links can be used for modeling interesting relationships. For example, the relationship `hasfather` between a person and her/his father is captured in $\mathcal{K}$ by $\texttt{children}^- \circ \texttt{father}$ (similarly for `hasmother`). Also, `ancestor` is captured by $(\texttt{hasfather} \cup \texttt{hasmother}) \circ (\texttt{hasfather} \cup \texttt{hasmother})^*$ (see the definition of `VeryPhd`).

As an example of inference that can be draw from $\mathcal{K}$, observe that:

$$\mathcal{K} \models \exists \texttt{children}^-.\top \sqsubseteq \texttt{Person} \sqcap \exists \ni^- .\exists \texttt{father}.\top.$$

Indeed, note that every instance of $\exists \texttt{children}^-.\top$ is also an instance of $\exists \texttt{father}^- \cup \texttt{mother}^- \cup \texttt{children}^-.\top$ and therefore is an instance of `Family`. This means that $\mathcal{K} \models \exists \texttt{children}^-.\top \sqsubseteq \exists \texttt{children}^-.\texttt{Family}$. Observe that $\mathcal{K} \models \texttt{Family} \sqsubseteq (\texttt{children} \subseteq \ni)$, and, since $\mathcal{K} \models \texttt{Family} \sqsubseteq \forall \ni .\texttt{Person}$ (because $\mathcal{K} \models \texttt{Family} \sqsubseteq \sigma(\texttt{Person})$), we have that $\mathcal{K} \models \exists \texttt{children}^-.\top \sqsubseteq \exists \texttt{children}^-.(\forall \texttt{children}.\texttt{Person})$, which implies that $\mathcal{K} \models \exists \texttt{children}^-.\top \sqsubseteq \texttt{Person}$. The fact that $\mathcal{K} \models \exists \texttt{children}^-.\top \sqsubseteq \exists \ni^- .\exists \texttt{father}.\top$ easily follows from the fact that every `Family` is a tuple with attribute `father`.

4 Conclusions

It is our opinion that the work described in this paper makes description logics accomplish the necessary leap in order to be well equipped for the new challenging applications they are faced with. Our first investigations show that $\mathcal{CAT}\mathcal{S}$ can indeed capture and extend most class-based representation formalisms used in different areas as AI, databases, software engineering, etc.. One main issue still remains to be addressed, namely, the possibility of adding to $\mathcal{CAT}\mathcal{S}$ suitable constructs for expressing finiteness of nested aggregates, and, correspondingly, suitable techniques for reasoning in finite models (in the style of [Calvanese *et al.*,1994]). This will be the subject of further research.

References

[Baader, 1991] F. Baader. Augmenting concept languages by transitive closure of roles: An alternative to terminological cycles. In *Proc. of IJCAI-91*, Sydney, Australia, 1991.

[Bergamaschi and Sartori, 1992] S. Bergamaschi and C. Sartori. On taxonomic reasoning in conceptual design. *ACM Trans. on Database Systems*, 17(3):385–422, 1992.

[Borgida, 1992] A. Borgida. From type systems to knowledge representation: Natural semantics specifications for description logics. *J. of Intelligent and Cooperative Information Systems*, 1(1):93–126, 1992.

[Brachman, 1977] R. J. Brachman. What's in a concept: Structural foundations for semantic networks. *International Journal of Man-Machine Studies*, 9(2):127–152, 1977.

[Brodie and Ridjanovic, 1984] M. L. Brodie and D. Ridjanovic. On the design and specification of database transactions. In *On Conceptual Modelling*, pages 277–306. Springer-Verlag, 1984.

[Buchheit *et al.*, 1993] M. Buchheit, F. M. Donini, and A. Schaerf. Decidable reasoning in terminological knowledge representation systems. *J. of Artificial Intelligence Research*, 1:109–138, 1993.

[Calvanese *et al.*, 1994] D. Calvanese, M. Lenzerini, and D. Nardi. A unified framework for class based representation formalisms. In *Proc. of KR-94*, pages 109–120, Bonn, 1994. Morgan Kaufmann, Los Altos.

[Catarci and Lenzerini, 1993] T. Catarci and M. Lenzerini. Representing and using interschema knowledge in cooperative information systems. *J. of Intelligent and Cooperative Information Systems*, 2(4):375–398, 1993.

[De Giacomo and Lenzerini, 1994] G. De Giacomo and M. Lenzerini. Boosting the correspondence between description logics and propositional dynamic logics. In *Proc. of AAAI-94*, pages 205–212. AAAI Press/The MIT Press, 1994.

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

[Piza *et al.*, 1992] B. Piza, K.-D. Schewe, and J. W. Schmidt. Term subsumption with type constructors. In *Proc. of CIKM-92*, pages 449–456, Baltimore, 1992.

[Schild, 1991] K. Schild. A correspondence theory for terminological logics: Preliminary report. In *Proc. of IJCAI-91*, pages 466–471, Sydney, 1991.

[Schmolze, 1989] J. G. Schmolze. Terminological knowledge representation systems supporting n-ary terms. In *Proc. of KR-89*, pages 432–443. Morgan Kaufmann, Los Altos, 1989.

[Schreiber *et al.*, 1993] G. Schreiber, B. Wielinga, and J. Breuker. *KADS: A principled approach to knowledge-based system development.* Academic Press, 1993.

[Smith and Smith, 1977] J. M. Smith and D. C. P. Smith. Database abstractions: Aggregation and generalization. *ACM Transactions on Database Systems*, 2(2):105–133, 1977.

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

[Woods, 1975] W. A. Woods. What's in a link: Foundations for semantic networks. In *Representation and Understanding: Studies in Cognitive Science*, pages 35–82. Academic Press, 1975.

Symbolic Arithmetical Reasoning with Qualified Number Restrictions

Hans Jürgen Ohlbach, Renate A. Schmidt and Ullrich Hustadt

Max-Planck-Institut für Informatik
Im Stadtwald, 66123 Saarbrücken, Germany
Email: {ohlbach, schmidt, hustadt}@mpi-sb.mpg.de

Abstract

Many inference systems used for concept description logics are constraint systems that employ tableaux methods. These have the disadvantage that for reasoning with qualified number restrictions n new constant symbols are generated for each concept of the form $(\geq n\ R\ C)$. In this paper we present an alternative method that avoids the generation of constants and uses a restricted form of symbolic arithmetic considerably different from the tableaux method. The method we use is introduced in Ohlbach, Schmidt and Hustadt 1995 for reasoning with graded modalities. We exploit the exact correspondence between the concept description language $\mathcal{ALCN}$ and the multi-modal version of the graded modal logic $\overline{\mathbf{K}}$ and show how the method can be applied to $\mathcal{ALCN}$ as well.

This paper is a condensed version of Ohlbach *et al.* 1995. We omit proofs and much of the technical details, but we include some examples.

1 The description logic $\mathcal{ALCN}^+$

The description logic $\mathcal{ALCN}$ defined in Hollunder and Baader 1991 extends the logic $\mathcal{ALC}$ of Schmidt-Schauß and Smolka 1991 with numerical quantifier constructs. The logic $\mathcal{ALC}$ can be viewed as a restricted form of a multi-modal logic, called $\mathbf{K}_{(m)}$ [Schild,1991]. The language of $\mathcal{ALC}$ includes the operations $\sqcap$ (conjunction), $\sqcup$ (disjunction), $\neg$ (negation), $\exists$ (existential role quantification) and $\forall$ (universal role quantification), and it includes two designated primitive concepts $\top$ (top) and $\bot$ (bottom). The Boolean operations correspond to the propositional connectives in modal logic, and role quantifier expressions of the general form $\exists R:C$ and $\forall R:C$ correspond to the modal formulae $\langle R\rangle C$ and $[R]C$, respectively. The numerical quantifier constructs in the language of $\mathcal{ALCN}$ have the general form

$$(\geq n\ R\ C) \quad \text{and} \quad (\leq n\ R\ C)$$

(with $n \geq 0$) and have modal correspondents as well [van der Hoek and de Rijke,1992]. $(\geq n\ R\ C)$ and $(\leq n\ R\ C)$ define sets of elements which have, respectively, at least n and at most n successors by R in C. For example,

$$(1) \qquad \mathit{city} \;=\; \mathit{place} \sqcap (\geq 100\,001\ \mathit{inhabited\text{-}by\ people})$$

defines the concept *city* to be a place with more than 100 000 inhabitants.

The language we consider is slightly more expressive than $\mathcal{ALCN}$. Our version, referred to as $\mathcal{ALCN}^+$, has no restrictions on the inclusion statements. On the left hand side of inclusions arbitrary concepts may occur, whereas in $\mathcal{ALCN}$ only atomic concepts can occur on the left hand side. Also, terminological cycles are allowed. $\mathcal{ALCN}^+$ coincides with the language $\mathcal{ALCNR}$ considered in Buchheit *et al.* 1993, except that role conjunction is not provided for.

Formally, the syntax of $\mathcal{ALCN}^+$ is defined as follows. The signature of the terminological language of $\mathcal{ALCN}^+$ consists of a set Σ_R of *role names* and a disjoint set Σ_C of *concept names*. From role names $Q \in \Sigma_R$ and concept names $A \in \Sigma_C$ compound concept terms C are formed according to the following rules:

$$\begin{aligned} C, D \;\longrightarrow\; & A \mid \neg C \mid C \sqcap D \mid C \sqcup D \mid \\ & \exists R:C \mid \forall R:C \mid \\ & (\geq n\ R\ C) \mid (\leq n\ R\ C) \mid \\ & C \sqsubseteq D \mid C = D. \end{aligned}$$

n is a non-negative integer. Most authors define the symbols $\sqsubseteq$ and $=$ to be sentential symbols. We define them to be connectives just as $\sqcap$ and $\sqcup$ are. Note, we consider terminological sentences of the form $C \sqsubseteq D$ and $C = D$ to be concept terms. In $\mathcal{ALCN}$ terminological sentences are constrained to be of the form $A \sqsubseteq C$ and $A = C$, where A are concept names. A *T-Box* is defined as a set of concept terms.

The semantics of $\mathcal{ALCN}^+$ is specified by an interpretation $\mathcal{I} = (U, V)$ with U a non-empty set U (the domain of interpretation) and a signature assignment V. The signature assignment maps role names to binary relations on U and it maps concept names to subsets of U. The interpretation of concept terms C and D specified

by:

$$\begin{aligned} C^{\mathcal{I}} &= V(C) \qquad \text{if } C \text{ is a concept name} \\ (C \sqsubseteq D)^{\mathcal{I}} &= (U \setminus C^{\mathcal{I}}) \cup D^{\mathcal{I}} \\ (C = D)^{\mathcal{I}} &= (U \setminus (C^{\mathcal{I}} \cup D^{\mathcal{I}})) \cup (C^{\mathcal{I}} \cap D^{\mathcal{I}}) \end{aligned}$$

and as usual for the remaining operations. Atomic concept names in a T-Box T are interpreted as the entire domain and are all equivalent to the top concept $\top$. $\top$ is the largest element in the subsumption ordering. The complement of $\top$ is $\bot$ and represents the empty set.

An interpretation $\mathcal{I} = (U, V)$ with $C^{\mathcal{I}} = U$ for all concept terms C in the T-Box T is a *model* of T. A concept term C is *universal* iff $C^{\mathcal{I}} = U$ for *all* interpretations $\mathcal{I}$. C is *empty* or *incoherent* iff $C^{\mathcal{I}} = \emptyset$ for *all* interpretations $\mathcal{I}$. The entailment relation $\models$ between concept terms is defined by: $C \models D$ iff $D^{\mathcal{I}} = U$ for every interpretation $\mathcal{I}$ of C. Then $C \models D$ iff $C \sqsubseteq D$ is universal iff $C \sqcap \neg D$ is empty.

We treat sets $\{C_1, \ldots, C_n\}$ of concept terms in the same way as the conjunction $C_1 \sqcap \ldots \sqcap C_n$. Thus, a given T-Box T will be treated as the conjunction of its elements.

In contrast to other terminological languages the language $\mathcal{ALCN}^+$ includes no role-forming operators. Roles that occur are all atomic. To simplify our presentation, without loss of generality we assume there is one atomic role R.

2 The graded modal logic $\overline{\mathbf{K}}$

The corresponding modal logic of $\mathcal{ALCN}^+$ is the multi-modal version of the graded modal logic $\overline{\mathbf{K}}$ (defined for example in van der Hoek 1992). Graded modalities are modal operators indexed with cardinal numbers which fix the number of worlds in which a formula is true. The formula $\Diamond_n\varphi$ (with n a non-negative integer) is true in a world iff there are more than n accessible worlds in which the formula φ is also true. The dual formula $\Box_n\varphi$, given by $\neg\Diamond_n\neg\varphi$, is then true in a world iff there are at most n accessible worlds in which $\neg\varphi$ is true. Another form of graded modal formula is $\Diamond!_n\varphi$. $\Diamond!_0\varphi$ abbreviates $\Box_0\neg\varphi$ and $\Diamond!_n\varphi$ abbreviates $\Diamond_{n-1}\varphi \wedge \neg\Diamond_n\varphi$ for $n > 0$. $\Diamond!_n\varphi$ is true in a word iff φ is true in exactly n accessible worlds. More formally, the semantics of the graded modal operators is defined in terms of one accessibility relation, say R, by:

$$\begin{aligned} &\mathcal{M}, x \models_{\overline{\mathrm{K}}} \Diamond_n\varphi \\ &\quad \text{iff} \quad |\{y \mid R(x,y) \;\&\; \mathcal{M}, y \models_{\overline{\mathrm{K}}} \varphi\}| > n \\ &\mathcal{M}, x \models_{\overline{\mathrm{K}}} \Box_n\varphi \\ &\quad \text{iff} \quad |\{y \mid R(x,y) \;\&\; \mathcal{M}, y \models_{\overline{\mathrm{K}}} \neg\varphi\}| \leq n \\ &\mathcal{M}, x \models_{\overline{\mathrm{K}}} \Diamond!_n\varphi \\ &\quad \text{iff} \quad |\{y \mid R(x,y) \;\&\; \mathcal{M}, y \models_{\overline{\mathrm{K}}} \varphi\}| = n. \end{aligned}$$

$\mathcal{M}$ denotes a model and x, y denote possible worlds. For any set X, $|X|$ denotes the cardinality of X. The modal version of $(\geq n\ R\ C)$ is $\Diamond_{n-1}C$ and the modal version of $(\leq n\ R\ C)$ is $\Box_n\neg C$. We think of the diamond and box operators being associated with the accessibility relation that defines the role R.

The logic $\overline{\mathbf{K}}$ has a Hilbert-style presentation $\Gamma_{\overline{\mathbf{K}}}$ that is sound and complete with respect to its natural possible worlds semantics [Fine,1972; de Caro,1988]. $\Gamma_{\overline{\mathbf{K}}}$ is defined by the following axioms

A1 the axioms of propositional logic

A2 $\vdash_{\overline{\mathrm{K}}} \Diamond_{n+1}\varphi \rightarrow \Diamond_n\varphi$

A3 $\vdash_{\overline{\mathrm{K}}} \Box_0(\varphi \rightarrow \psi) \rightarrow (\Diamond_n\varphi \rightarrow \Diamond_n\psi)$

A4 $\vdash_{\overline{\mathrm{K}}} \Box_0\neg(\varphi \wedge \psi) \rightarrow ((\Diamond!_n\varphi \wedge \Diamond!_m\psi) \rightarrow \Diamond!_{n+m}(\varphi \vee \psi))$

together with the uniform substitution rule, Modus Ponens, and the necessitation rule for $\Box_0$:

US if φ is a theorem so is every substitution instance of φ

MP if $\vdash_{\overline{\mathrm{K}}} \varphi$ and $\vdash_{\overline{\mathrm{K}}} \varphi \rightarrow \psi$ then $\vdash_{\overline{\mathrm{K}}} \psi$

N if $\vdash_{\overline{\mathrm{K}}} \varphi$ then $\vdash_{\overline{\mathrm{K}}} \Box_0\varphi$.

Van der Hoek 1992 shows $\overline{\mathbf{K}}$ is decidable.

3 $\mathcal{ALCN}^+$ and $\overline{\mathbf{K}}$

We now show that $\mathcal{ALCN}^+$ can be embedded in $\overline{\mathbf{K}}$. Define a mapping τ from $\mathcal{ALCN}^+$ to $\overline{\mathbf{K}}$ by:

$$\begin{aligned} \tau(C) &= C \qquad \text{if } C \text{ is an atomic concept} \\ \tau(\neg C) &= \neg\tau(C) \\ \tau(C \sqcap D) &= \tau(C) \wedge \tau(D) \\ \tau(C \sqcup D) &= \tau(C) \vee \tau(D) \\ \tau(C \sqsubseteq D) &= \tau(C) \rightarrow \tau(D) \\ \tau(C = D) &= \tau(C) \leftrightarrow \tau(D) \\ \tau(\exists R\!:\!C) &= \Diamond_0\tau(C) \\ \tau(\forall R\!:\!C) &= \Box_0\tau(C) \\ \tau(\geq n\ R\ C) &= \Diamond_{n-1}\tau(C) \\ \tau(\leq n\ R\ C) &= \Box_n\neg\tau(C) \end{aligned}$$

Theorem 1 *The translation τ is sound and complete: For any concept C, C is universal iff $\tau(C)$ is a tautology, i.e. $\models_{\overline{\mathrm{K}}} \tau(C)$.*

Proof. Let $\mathcal{I} = (U, V)$ be any interpretation of a T-Box of $\mathcal{ALCN}^+$. Let $\mathcal{M}$ be the modal model $(U, R^{\mathcal{I}}, V)$. By induction on the structure of C prove, for every $x \in U$: $x \in C^{\mathcal{I}}$ iff $\mathcal{M}, x \models_{\overline{\mathrm{K}}} \tau(C)$. The proof is routine and we omit the details. □

Since $\overline{\mathbf{K}}$ is sound and complete, $\Gamma_{\overline{\mathbf{K}}}$ provides an axiomatization for $\mathcal{ALCN}^+$ with one role (that is both sound and complete). We have:

Theorem 2 *For any concept C, C is universal iff its translation $\tau(C)$ is provable in $\Gamma_{\overline{\mathbf{K}}}$.*

A sound and complete axiomatisation for $\mathcal{ALCN}^+$ with arbitrarily many roles is given by the multi-modal version of $\Gamma_{\overline{\mathbf{K}}}$ in which each modal operator is indexed with an R.

4 Reasoning for $\mathcal{ALCN}^+$

Tableaux systems like the constraint system described in Hollunder and Baader 1991, for description logics with numerical quantifier constructs, are based on the set-theoretic semantics and generate for each numerical quantifier $\geq n$ (or $\Diamond_{n-1}$) n new constant symbols. For large n as in the sample definition (1) this is infeasible. On the other hand, the axiomatization of $\overline{\mathbf{K}}$ is formulated with arithmetical terms, and in principle, this allows for invoking arithmetical computations, thus, avoiding the manipulation of explicitly generated constants. However, Hilbert systems have other disadvantages that makes them unsuitable to form the basis for automated reasoning.

Ohlbach *et al.* 1995 show that the axiomatization $\Gamma_{\overline{\mathbf{K}}}$ of $\overline{\mathbf{K}}$ can be transformed into a first-order theory that exactly captures $\overline{\mathbf{K}}$. The first-order theory is defined by the set of clauses P1–P12 given in Figure 1 below.

Theorem 1 *Any graded modal formula φ is provable in $\overline{\mathbf{K}}$ iff its first-order translation* $\mathrm{FO}(\varphi)$ *is a logical consequence of the first-order theory iff* $\neg\mathrm{FO}(\varphi)$ *in clause form is refutable from* P1–P12*.*

P1–P12 are obtained in two reduction steps.

The first reduction embeds $\overline{\mathbf{K}}$ in an intermediary multi-modal logic, called $\overline{\mathbf{K}}_E$, with a standard Kripke semantics.

The second reduction uses an optimization of the functional translation method of multi-modal logics into sorted predicate logic [Ohlbach and Schmidt,1995].

The first reduction is necessary, because the functional translation method is formulated for multi-modal logics and $\overline{\mathbf{K}}$ is not a multi-modal logic in the usual sense that each modality is associated with a different binary relation (in $\overline{\mathbf{K}}$ every modality $\Diamond_n$ is associated with the same relation R).

5 The logics $\overline{\mathbf{K}}_E$ and $\overline{\mathbf{K}}$

$\overline{\mathbf{K}}_E$ has two kinds of modalities:

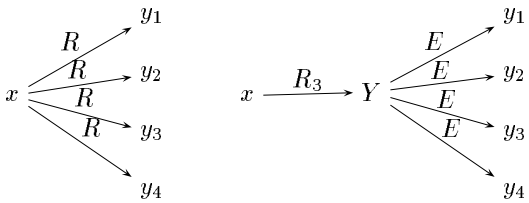
(i) $\langle n\rangle$ and $[n]$, which are characterized in the Kripke frames by infinitely but countably many different relations R_n ($n \in \mathbf{N}_0$), and

(ii) $\Diamond$ and $\Box$, which are characterized by a designated relation E.

Ohlbach et al. translate formulae of $\overline{\mathbf{K}}$ to formulae of $\overline{\mathbf{K}}_E$ according to the following rules (for modal operators):

$$(2)\qquad \begin{array}{lcl} \Pi(\Diamond_n\varphi) & = & \langle n\rangle\Box\Pi(\varphi) \\ \Pi(\Box_n\varphi) & = & [n]\Diamond\Pi(\varphi). \end{array}$$

The intuitive idea underlying the translation of $\Diamond_n\varphi$ is this: If φ is true in a set Y of worlds with more than n elements then we introduce an accessibility relation R_n that connects the actual world and a world w_Y which we can think of as being a representative for the set Y. This defines the $\langle n\rangle$ operator. $\Box\varphi$ and its associated accessibility relation E expresses that φ is true in all the worlds of the set Y. E can be thought of as the reverse membership relation. Thus, $\langle n\rangle\Box\varphi$ encodes 'there is a set with more than n elements (encoded by $\langle n\rangle$) and φ is true for all the elements of this set (encoded by $\Box$)'. For example, consider the formula $\Diamond_3\varphi$. According to the semantic definition $\Diamond_3\varphi$ is true in a world x iff there are at least 4 worlds to which x is R-related. This definition is depicted in the first picture below. The second picture depicts our new alternative view.



The relation R is replaced by the relational composition of the two new relations R_3 and E. In the process we have introduced a new world which we labelled Y as it is meant to represent the set of worlds y_1, y_2, y_3 and y_4.

$\overline{\mathbf{K}}_E$ has a Hilbert-style axiomatisation $\Gamma_{\overline{\mathbf{K}}_E}$. The axioms and rules are:

N1 the axioms of propositional logic and Modus Ponens

N2 the K-axioms for $[n]$ and $\Box$:
$\vdash_{\overline{\mathrm{K}}_E} [n](\varphi\to\psi)\to([n]\varphi\to[n]\psi)$
$\vdash_{\overline{\mathrm{K}}_E} \Box(\varphi\to\psi)\to(\Box\varphi\to\Box\psi)$

N3 the necessitation rules for $[n]$ and $\Box$:
if $\vdash_{\overline{\mathrm{K}}_E} \varphi$ then $\vdash_{\overline{\mathrm{K}}_E} [n]\varphi$
if $\vdash_{\overline{\mathrm{K}}_E} \varphi$ then $\vdash_{\overline{\mathrm{K}}_E} \Box\varphi$

N4 $\vdash_{\overline{\mathrm{K}}_E} [0]\Diamond\varphi\to[n]\Box\varphi$

N5 $\vdash_{\overline{\mathrm{K}}_E} \langle n\rangle\Box\varphi\to\langle n\rangle\Diamond\varphi$

N6 $\vdash_{\overline{\mathrm{K}}_E} [n]\varphi\to[n+1]\varphi$

N7 $\vdash_{\overline{\mathrm{K}}_E} \langle n+m\rangle\Box(\varphi\vee\psi)\to(\langle n\rangle\Box\varphi\vee\langle m\rangle\Box\psi)$

N8 $\vdash_{\overline{\mathrm{K}}_E} \langle n\rangle\Box\varphi\wedge\langle m\rangle\Box\psi\wedge[j]\Diamond\neg(\varphi\wedge\psi)$
$\to\langle n+m+1-j\rangle\Box(\varphi\vee\psi)$.

Here, N8 is in a more general form than that of Ohlbach *et al.* 1995 without changing $\overline{\mathbf{K}}_E$. The resulting set of first-order clauses obtained by the reduction to come is larger and better suited for our purposes, though.

Theorem 1 *The transformation of $\overline{\mathbf{K}}$ to $\overline{\mathbf{K}}_E$ is sound and complete: for any graded modal formula φ, φ is provable from $\Gamma_{\overline{\mathbf{K}}}$ iff its translation* Π *(defined in (2) for modal formulae) is provable from $\Gamma_{\overline{\mathbf{K}}_E}$.*

6 $\overline{\mathbf{K}}_E$ and first-order logic

One of the axioms of $\overline{\mathbf{K}}_E$ is not first-order definable, if we use, in the second reduction step, the standard relational translation of multi-modal logics to first-order logic. Using the functional translation we can avoid this problem [Ohlbach and Schmidt,1995]. Furthermore, the

P1 $de_n(w_*),\ [w_* x_n z] = [w_* f_0^n(w_*, x_n, z) y]$

P2 $AF_m \sqsubseteq AF_n \qquad$ for all $m > n$

P3 $\neg de_n(w_*),\ de_m(w_*) \qquad$ for all $m > n$

P4 $de_{n+m}(w_*),\ [w_* x_{n+m} h1^{nm}(w_*, x_{n+m}, y)] = [w_* h2_n^{nm}(w_*, x_{n+m}, y, z) y]$

P5 $de_{n+m}(w_*),\ [w_* x_{n+m} h1^{nm}(w_*, x_{n+m}, y)] = [w_* h3_n^{nm}(w_*, x_{n+m}, y) z]$

P6 $de_{\max(n,m)}(w_*),\ \neg de_{n+m+1}(w_*),\ [w_* x_n k1^{nm}(w_*, x_n, y_m)] = [w_* y_m k2^{nm}(w_*, x_n, y_m)]$

P7 $de_{\max(n,m)}(w_*),\ [w_* x_n k3^{nm}(w_*, x_n, y_m)] = [w_* y_m k4^{nm}(w_*, x_n, y_m)],$
$[w_* x_n k5^{nm}(w_*, x_n, z)] = [w_* k_{n+m+1}^{nm}(w_*, x_n, y_m) z],$
$[w_* y_m k6^{nm}(w_*, y_m, z)] = [w_* k_{n+m+1}^{nm}(w_*, x_n, y_m) z].$

P8 $de_{\max(n,m)}(w_*),\ \neg de_{n+m+1-j}(w_*),\ [w_* f7_j^{nmj}(w_*, x_n, x_m) u] = [w_* x_n f5^{nmj}(w_*, x_n, u)]$

P9 $de_{\max(n,m)}(w_*),\ \neg de_{n+m+1-j}(w_*),\ [w_* f7_j^{nmj}(w_*, x_n, x_m) u] = [w_* x_m f6^{nmj}(w_*, x_m, u)]$

P10 $de_{\max(n,m)}(w_*),\ \neg de_j(w_*),\ [w_* f1_{n+m+1-j}^{nmj}(w_*, v, x_n) v] = [w_* x_n f2^{nmj}(w_*, v, x_n)],$
$[w_* f3_{n+m+1-j}^{nmj}(w_*, v, x_m) v] = [w_* x_m f4^{nmj}(w_*, v, x_m)]$

P11 $de_{\max(n,m)}(w_*),\ [w_* f1_{n+m+1-j}^{nmj}(w_*, v, x_n) v] = [w_* x_n f2^{nmj}(w_*, v, x_n)],$
$[w_* f3_{n+m+1-j}^{nmj}(w_*, v, x_m) v] = [w_* x_m f4^{nmj}(w_*, v, x_m)],$
$[w_* f7_j^{nmj}(w_*, x_n, x_m) u] = [w_* x_n f5^{nmj}(w_*, x_n, u)]$

P12 $de_{\max(n,m)}(w_*),\ [w_* f1_{n+m+1-j}^{nmj}(w_*, v, x_n) v] = [w_* x_n f2^{nmj}(w_*, v, x_n)],$
$[w_* f3_{n+m+1-j}^{nmj}(w_*, v, x_m) v] = [w_* x_m f4^{nmj}(w_*, v, x_m)],$
$[w_* f7_j^{nmj}(w_*, x_n, x_m) u] = [w_* x_m f6^{nmj}(w_*, x_m, u)]$

Figure 1: The first-order theory that captures $\overline{\mathbf{K}}_E$, $\overline{\mathbf{K}}$ and $\mathcal{ALCN}^+$ with one role.

functional approach has computational advantages over the standard relational translation method.

Recall, the relational translation of modal logics is commonly denoted by ST and uses the Kripke semantics definition, see e.g. van Benthem 1984. For (multi-) modal operators, ST is defined by:

$$\begin{aligned} \mathrm{ST}(\langle R\rangle\varphi, x) &= \exists y\ R(x,y) \wedge \mathrm{ST}(\varphi, y) \\ \mathrm{ST}([R]\varphi, x) &= \forall y\ R(x,y) \rightarrow \mathrm{ST}(\varphi, y) \end{aligned}$$

The functional translation method [Ohlbach,1991] is based on the fact that any binary relation R can be defined by a set AF_R of partial functions, namely:

$$R(x,y) \quad \leftrightarrow \quad \exists \gamma \in AF\ \ y = \gamma(x).$$

The functional translation π_f for (multi-) modal formulae is:

$$\begin{aligned} \pi_f(\langle R\rangle\varphi, x) &= \neg de_R(x) \wedge \exists \gamma{:}AF_R\ \pi_f(\varphi, \downarrow(\gamma, x)) \\ \pi_f([R]\varphi, x) &= \neg de_R(x) \rightarrow \forall \gamma{:}AF_R\ \pi_f(\varphi, \downarrow(\gamma, x)). \end{aligned}$$

The term $\neg de_R(x)$ is meant to capture that x is not a dead-end in the relation R, i.e. R is defined for x. $\downarrow$ is the 'apply' function, so, we can think of the term $\downarrow(\gamma, x)$ as representing $\gamma(x)$. Total (serial) relations can be defined by a set of total functions and have a simpler functional formulation not involving dead-end predicates. The relation E in the semantics of $\overline{\mathbf{K}}_E$ is a total relation. For modal operators, like $\Box$ of $\overline{\mathbf{K}}_E$, which are determined by a total relation the functional translation π_f is given by:

$$\begin{aligned} \pi_f(\langle R\rangle\varphi, x) &= \exists \gamma{:}AF_R\ \pi_f(\varphi, \downarrow(\gamma, x)) \\ \pi_f([R]\varphi, x) &= \forall \gamma{:}AF_R\ \pi_f(\varphi, \downarrow(\gamma, x)). \end{aligned}$$

Take for example this concept

$$(\geq 1\ R\ (\geq 1\ R\ (\geq 4\ R\ \top))).$$

Its modal translation by τ is $\Diamond_0\Diamond_0\Diamond_3\top$. The first reduction by Π (in (2)) yields its $\overline{\mathbf{K}}_E$-version: $\langle 0\rangle\Box\langle 0\rangle\Box\langle 3\rangle\Box\top$. According to the functional translation the second reduction yields, for a world w (in a simplified form):

$$\begin{aligned} & \neg de_0(w) \\ & \wedge \exists \alpha{:}AF_0\ \forall \gamma{:}AF_E\ \neg de_0([\alpha\gamma]w) \\ & \wedge \exists \beta{:}AF_0\ \forall \delta{:}AF_E\ \neg de_3([\alpha\gamma\beta\delta]w). \end{aligned} \tag{3}$$

de_n is an abbreviated notation for de_{R_n}. Note, since E is a total relation it is defined by a set AF_E of total functions, which implies the dead-end predicates de_E are superfluous.

Likewise the axioms $\Gamma_{\overline{\mathbf{K}}_E}$ of $\overline{\mathbf{K}}_E$ can be systematically translated into their functional representation. We omit the details, but see Ohlbach *et al.* 1995. This translation does not yet give us P1–P12 of Figure 1. It yields a set of second-order formulae. To compute the first-order equivalents, we use the elimination algorithm of

second-order quantifiers of Gabbay and Ohlbach 1992 together with an optimization step developed in Ohlbach and Schmidt 1995. The resulting clauses are P1–P12. (Again, we omit the details.)

Ohlbach and Schmidt 1995 prove:

Theorem 1 *The functional translation is sound and complete relative to the completeness of the relational translation.*

The theorem applies to extensions of the normal multi-modal logic $\mathbf{K}_{(m)}$. Let Φ be the additional axioms that define the extension. The theorem says, more formally, *provided the second-order relational translation of the extension Φ is complete with respect to a first-order class of frames, φ is provable in Φ iff $\Pi_f(\Phi) \to \Pi_f(\varphi)$ is a predicate logic theorem.* (Π_f is the functional translation mapping for axioms and rules that uses the translation mapping π_f for modal formulae.)

Ohlbach and Schmidt 1995 also prove a stronger theorem for the functional translation together with an optimization that allows for functional existential and universal quantifiers to be swapped arbitrarily. This rule exploits that one relational frame in general corresponds to many 'functional frames', and there is always one which is rich enough to allow for moving existential quantifiers over universal quantifiers.

Theorem 2 *The functional translation with the quantifier exchange rule is sound and complete relative to the completeness of the relational translation.*

Combining Theorems 1 to 2 it is easy to see that the sequence of translations which we described in Sections 5 and 6 is sound and complete. *For any $\overline{\mathbf{K}}$-formula φ,* we have *φ is a $\overline{\mathbf{K}}$-theorem iff* $(\mathrm{P1}\text{–}\mathrm{P12}) \to \Upsilon(\Pi_f(\varphi))$ *is a predicate logic theorem.* This is the main theorem which we stated earlier in Theorem 1. The operation FO of Theorem 1 is the combination of the $\overline{\mathbf{K}}$ to $\overline{\mathbf{K}}_E$ translation Π and the $\overline{\mathbf{K}}_E$ to predicate logic translation Π_f followed by Υ, the optimized functional translation. Υ is the operation that moves existential functional quantifiers inwards over universal quantifiers. This operation is not mandatory. It is useful for replacing in complete modal logics, modal axioms that alone (without the interaction with other axioms and rules) have no first-order (relational) correspondence property, by weaker (functional) correspondence properties.

Note, the notation used in Figure 1 is very much abbreviated. It is known as the world path notation. Subscripts indicate the sort of a symbol. For example, x_n is a variable of sort AF_n. f_0^n is a Skolem function of sort AF_0. The absence of subscripts as for z and $h1^{nm}$ indicates being of sort AF_E. The variable w_* is of sort AF_R^*, which is the supersort of all sorts AF_n.

P1 has the following reading: if w_* is an arbitrary (path to a) world from which a path leads to a set of worlds with more than n worlds $(de_n(w_*), \ldots)$ then for any path x_n to a set X of worlds with $|X| > n$ and any world z in this set there exists a path $f_0^n(w_*, x_n, z)$ to a non-empty set B and any y in this set is also in X. This implies $\{z\} \subseteq X$.

7 Symbolic arithmetical reasoning for $\mathcal{ALCN}^+$ and graded modal logic

Evidently, because of the exact correspondence between $\mathcal{ALCN}^+$ and the graded modal logic $\overline{\mathbf{K}}$, Theorem 1 remains true if we replace 'graded modal formula' and φ by 'concept' and C, respectively. Consequently we may apply the inference method of Ohlbach *et al.* 1995 for $\mathcal{ALCN}^+$ too. The method uses theory resolution based on P1–P12. This has the advantage that instead of counting constants we use traditional resolution together with symbolic arithmetical reasoning. We demonstrate this by way of two examples.

Example 1 Consider the set of concepts

$$\begin{array}{c}(\geq 1\ R\ (\geq 1\ R\ (\geq 4\ R\ \top))),\\ (\geq 1\ R\ (\geq 1\ R\ (\leq 3\ R\ \top))),\\ (\geq 1\ R\ (\leq 1\ R\ \top)),\\ (\leq 1\ R\ \top).\end{array} \tag{4}$$

The corresponding set of $\overline{\mathbf{K}}$-formulae are

$$\Diamond_0\Diamond_0\Diamond_3\top,\quad \Diamond_0\Diamond_0\Box_3\bot,\quad \Diamond_0\Box_1\bot,\quad \Box_1\bot. \tag{5}$$

The conjunction of (4) is incoherent (or unsatisfiable). This amounts to the same thing as saying, the set (5) is inconsistent.

Above we derived the functional translation (3) for the first concept in (4). The functional translation for the remaining concepts is obtained analogously. The sets reduce to the following set of clauses.

$$\begin{array}{llll} C_1 & \neg de_0([]) & C_5 & de_3([c_0x'd_0y']) \\ C_2 & \neg de_0([a_0x]) & C_6 & de_1([e_0x'']) \\ C_3 & \neg de_3([a_0xb_0y]) & C_7 & de_1([]) \\ C_4 & \neg de_0([c_0x']) & & \end{array}$$

The clauses C_1, C_2 and C_3 represent (3). The empty path [] replaces the world variable w. a_0 and b_0 are the Skolem constant associated with α and β. The variables x and y are associated with γ and δ.

Now, we demonstrate how we can use P1–P12 and refutational theorem proving together with limited symbolic arithmetic to verify the claim that the conjunction of (4) is incoherent. For the refutation we use P1 with $n = 0$ and P6 with $n = 0$ and $m = 0$. P1 can immediately be simplified with clause C_1. The instances are:

P1′ $[f_0^0([], x_0, z)y] = [x_0z]$

P6′ $de_0(w_*),\ \neg de_1(w_*),\ [w_*x_0k1^{00}(w_*, x_0, y_0)] = [w_*y_0k2^{00}(w_*, x_0, y_0)]$.

The result of simultaneously resolving P6′, C_1, and C_7 using the unifier $\{w_* \mapsto []\}$ is

$$C_8 \quad [x_0k1^{00}([], x_0, y_0)] = [y_0k2^{00}([], x_0, y_0)].$$

Paramodulating with C_8 and unifier

$$\{x_0 \mapsto a_0,\ x \mapsto k1^{00}([], a_0, y_0)\},$$

C_3 becomes (this means we do equality replacement with unification in C_3 using the equation C_8)

$$C_9 \quad \neg de_3([y_0 k2^{00}([], a_0, y_0) b_0 y]).$$

This becomes

$$C_{10} \quad \neg de_3([x_0 z b_0 y])$$

when paramodulating with P1′ using the unifier

$$\{y_0 \mapsto f_0^0([], x_0, z),\ y \mapsto k2^{00}([], a_0, y_0)\}.$$

We resolve P6′ and C_4 using unifier

$$\{w_* \mapsto [c_0 x],\ x' \mapsto x\}$$

to get

$$C_{11} \quad \neg de_1([c_0 x]),\ [c_0 x x_0' k1^{00}([c_0 x], x_0', y_0')] = [c_0 x y_0' k2^{00}([c_0 x], x_0', y_0')].$$

Now, use the unifier

$$\{x_0 \mapsto c_0,\ x' \mapsto z \mapsto x, x_0' \mapsto d_0, y_0' \mapsto b_0,\ y' \mapsto k1^{00}([c_0 x], d_0, b_0), y \mapsto k2^{00}([c_0 x], d_0, b_0)\}$$

and apply E-resolution to C_5, C_{10} and C_{11} and get

$$C_{12} \quad \neg de_1([c_0 x]).$$

(This means we resolve between C_5 and C_{10} using an equation in C_{11}.) Resolving this with C_6 using E-resolution with C_8 yields the empty clause. The unifier is

$$\{x_0 \mapsto e_0,\ x'' \mapsto k1^{00}([], e_0 c_0), y_0 \mapsto c_0,\ x'' \mapsto k2^{00}([], e_0 c_0)\}.$$

Note, the computational effort does not depend on the numbers we use. The proof of this example is not significantly different for other (possibly large) values.

Example 2 Suppose the universe consists of at most thirty horses. If there are at least twenty horses that are white and there are at least twenty horses that are black, then there are at least ten zebras. Let W denote the set of white horses and B the set of black horses. Then $W \cap B$ denotes the set of zebras.

A standard tableaux system for the number operators would generate twenty witnesses for W, twenty witnesses for B and then it would need to identify ten of them in order not to exceed the limit of thirty. But there are combinatorically many ways for identifying ten of them.

In our system we prove the conjecture by showing the following set of $\mathcal{ALCN}^+$ concepts is incoherent:

$$(\geq 20\ R\ W),\ \ (\geq 20\ R\ B), (\leq 30\ R\ \bot),\ \ (\leq 9\ R\ \neg(W \sqcap B)).$$

The corresponding set of $\overline{\mathbf{K}}$-formulae are:

$$\Diamond_{19} W,\ \ \Diamond_{19} B,\ \ \Box_{30} \bot,\ \ \Box_9 \neg(W \wedge B).$$

We could choose any other suitable combination of numbers. This would not change the structure of the proof at all. The translation into first-order clause form is:

$$\neg de_{19}([]) \wedge W([a_{19} x]),\ \ \neg de_{19}([]) \wedge B([b_{19} x]), de_{30}([]),\ \ de_9([]) \vee \neg W([y_9 c]) \vee \neg B([y_9 c]).$$

The corresponding set of clauses consists of:

$$\begin{array}{ll} C_1 & \neg de_{19}([]) \\ C_2 & W([a_{19} x]) \\ C_3 & B([b_{19} y]) \\ C_4 & de_{30}([]) \\ C_5 & de_9([]),\ \neg W([y_9 c]),\ \neg B([y_9 c]) \end{array}$$

Resolve C_5 with P3 and C_1 and eliminate the $de_9([])$ literal from C_5 leaving:

$$C_5' \quad \neg W([y_9 c]),\ \neg B([y_9 c])$$

We resolve the instance of P9 with $n = m = 19$, $j = 9$, namely

$$\text{P9}' \quad de_{19}([]),\ \neg de_{30}([]), [f7_9^{19\,19\,9}([], x_{19}, x_{19}') u] = [x_{19} f6^{19\,19\,9}([], x_{19}', u)],$$

with C_1 and C_4 and obtain

$$C_6 \quad [f7_9^{19\,19\,9}([], x_{19}, x_{19}') u] = [x_{19}' f6^{19\,19\,9}([], x_{19}', u)].$$

Applying the unifier

$$\{y_9 \mapsto f7_9^{19\,19\,9}([], x_{19}, x_{19}'), u \mapsto c\},$$

we can use this in a paramodulation step with C_5' resulting in

$$C_7 \quad \neg W([x_{19}' f6^{19\,19\,9}([], x_{19}', c)]), \neg B([f7_9^{19\,19\,9}([], x_{19}, x_{19}') c])$$

Unify in C_2 and C_7 with

$$\{x_{19}' \mapsto a_{19}, x \mapsto f6^{19\,19\,9}([], x_{19}', c)\}.$$

Resolving C_2 and C_7 yields

$$C_8 \quad \neg B([f7_9^{19\,19\,9}([], x_{19}, a_{19}) c]).$$

Now we use the following instance of P8:

$$\text{P8}' \quad de_{19}(w_*),\ \neg de_{30}(w_*), [w_* f7_9^{19\,19\,9}(w_*, x_{19}, x_{19}') u] = [w_* x_{19} f5^{19\,19\,9}(w_*, x_{19}, u)]$$

This can be reduced with C_1 and C_4 to the equation

$$C_9 \quad [f7_9^{19\,19\,9}([], x_{19}, x_{19}') u] = [x_{19} f5^{19\,19\,9}([], x_{19}, u)],$$

which we can now use in a paramodulation step with C_8. We get

$$C_{10} \quad \neg B([x_{19} f5^{19\,19\,9}([], x_{19}, c)]).$$

The empty clause is obtained if we resolve C_{10} with C_3 using the appropriate unifier.

8 Conclusion

In description logics with qualified number restrictions it is possible to express properties of finite sets. The usual constraint inference algorithms similar to those described in Schmidt-Schauß and Smolka 1991 and Hollunder and Baader 1991 generate for all sets used in the proof at least as many constants (witnesses) as the cardinality of each set. Even for moderate values a vast number of witnesses are generated which are processed by case distinctions in the proof. For weaker description languages considered by Donini *et al.* 1994 and Calvanese *et al.* 1994 reasoning with unqualified number restrictions by case distinction can be avoided.

In this paper we have considered an alternative method for reasoning with both qualified and unqualified number restriction which does not have the overhead of evaluating case distinctions. Instead the method uses limited arithmetical reasoning. The method was developed for graded modal logics and can be readily applied to all description languages that are included in $\mathcal{ALCN}^+$. We showed that there is a close correspondence between $\mathcal{ALCN}$ and the graded modal logic $\overline{\mathbf{K}}$. In fact, there is an exact correspondence between the terminological operators and the modal operators.

We conclude with two remarks. (i) In the examples only one role R was used. The approach can be applied to description logics with multiple roles (without role definitions), too. (ii) We have restricted our attention to TBox reasoning. This corresponds directly to that in modal logic. We haven't accounted for A-Box reasoning about concrete instantiations of concepts/sets and roles/relations. The functional translation applied to A-Box terms generates many equations. It is not immediate how these can be treated efficiently. It remains to be investigated how the method can be extended to handle ABox reasoning as well.

References

[Buchheit *et al.*, 1993] M. Buchheit, F. M. Donini, and A. Schaerf. Decidable reasoning in terminological knowledge representation systems. *J. Artificial Intelligence Research*, 1:109–138, 1993.

[Calvanese *et al.*, 1994] D. Calvanese, M. Lenzerini, and D. Nardi. A unified framework for class-based representation formalisms. In *Proc. of KR'94*, pages 109–120, 1994.

[de Caro, 1988] F. de Caro. Graded modalities II. *Studia Logica*, 47:1–10, 1988.

[Donini *et al.*, 1994] F. M. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. Deduction in concept languages: From subsumption to instance checking. *J. Logic Computat.*, 4(4):423–452, 1994.

[Fine, 1972] K. Fine. In so many possible worlds. *Notre Dame Journal of Formal Logic*, 13(4):516–520, 1972.

[Gabbay and Ohlbach, 1992] D. M. Gabbay and H. J. Ohlbach. Quantifier elimination in second-order predicate logic. *South African Computer Journal*, 7:35–43, 1992. Also in *Proc. of KR'92*, pp. 425–436.

[Hollunder and Baader, 1991] B. Hollunder and F. Baader. Qualifying number restrictions in concept languages. In *Proc. of KR'91*, pages 335–346, 1991.

[Ohlbach and Schmidt, 1995] H. J. Ohlbach and R. A. Schmidt. Functional translation and second-order frame properties of modal logics. Tech. Rep. MPI-I-95-2-002, MPI Informatik, Saarbrücken, January 1995.

[Ohlbach *et al.*, 1995] H. J. Ohlbach, R. A. Schmidt, and U. Hustadt. Translating graded modalities into predicate logic. Manuscript, MPI Informatik, Saarbrücken. Forthcoming in H. Wansing (ed), *Proof Theory for Modal Logic.*, January 1995.

[Ohlbach, 1991] H. J. Ohlbach. Semantics based translation methods for modal logics. *J. Logic Computat.*, 1(5):691–746, 1991.

[Schild, 1991] K. Schild. A correspondence theory for terminological logics: Preliminary report. In *Proc. of IJCAI'91*, pages 466–471, 1991.

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

[van Benthem, 1984] J. van Benthem. Correspondence theory. In D. Gabbay and F. Guenther, editors, *Handbook of Philosophical Logic*, volume II, pages 167–247. Reidel, Dordrecht, 1984.

[van der Hoek and de Rijke, 1992] W. van der Hoek and M. de Rijke. Counting objects in generalized quantifier theory, modal logic, and knowledge representation. Tech. Rep. IR-307, Vrije Univ. Amsterdam, 1992.

[van der Hoek, 1992] W. van der Hoek. On the semantics of graded modalities. *J. Applied Non-Classical Logics*, 2(1):81–123, 1992.

Research and Applications in Description-Logic-Based Knowledge Representation

Peter F. Patel-Schneider, Deborah L. McGuinness
Merryll Kim Herman, Lori Alperin Resnick
Elia S. Weixelbaum

AT&T Bell Laboratories
Murray Hill, New Jersey

Since the fall of 1994, we have been working on a new version of CLASSIC [Resnick *et al.*,1993; Brachman *et al.*,1991], called NEOCLASSIC. We feel that building a new version of CLASSIC is a vital step in our continuing research on description logics, and in the use of description-logic-based systems in applications at AT&T, such as the PROSE configurator [Wright *et al.*,1993], even though we have made our research version of CLASSIC quite efficient [Patel-Schneider *et al.*,1991; Heinsohn *et al.*,1992], and already have a separate development version of CLASSIC [Weixelbaum,1991].

Our reimplementation continues the road from theoretical work in description logics to practical description-logic-based knowledge representation systems that has been described by Brachman [1992]. We strongly feel that if description logic is to be a successful knowledge representation technology, it has to be used in more applications. Further, many of the interesting recent developments in CLASSIC (e. g., increased expressive power, explanation [McGuinness and Borgida,1995], meta-information, rule extensions) have been affected by the use of CLASSIC in applications, so applications have a vital role to play in the further theoretical work on description logics.

The main reason for our reimplementation is to encourage greater interaction between knowledge representation research in description logics and application work using description-logic-based knowledge representation systems. Until now, we have fostered this interaction by having two versions of CLASSIC, a research version written in LISP [Resnick *et al.*,1993] and a development version written in C [Weixelbaum,1991]. This split between research and development versions of CLASSIC was useful because it allowed for a version of CLASSIC that was acceptable to developers, while also providing a research version that could be extended easily for experimental purposes.

The development version, however, was not as powerful as the research version, and could not be changed as easily as the research version. This meant that advances to the research version have not transferred to the development version as quickly as we had hoped. Therefore we are switching to a single version (NEOCLASSIC) that will be used for both research and applications. With a single version the requirement for reimplementation will be eliminated, and quick transfer of changes and additions from research version to applications will be possible.[1]

It is also a non-trivial task to translate knowledge bases between the two versions of CLASSIC, particularly because CLASSIC allows for test functions and computed rules, written as functions in the underlying programming language, vital aspects of CLASSIC that allows it to be used where it would otherwise not be sufficiently powerful. Currently we have tools do this translation, including conversion of a set of test functions that are used in our major applications, but these are by no means general tools. As different kinds of test functions are used, we have to rewrite them in LISP.

Also, it is next-to-impossible to take full applications and run them on the research version, because we cannot move the non-CLASSIC part of the application to the research platform. Thus we cannot show off advances in the research version to developers, and cannot experiment with complete applications to see what advances in representation or in tools would make them better.

With a single core version of CLASSIC we can easily transfer both application knowledge bases and complete applications to the research variant. We will then be able to perform experiments, instrumenting the behavior of applications, trying out new representation ideas, and prototyping new tools. The transfer of the results of these research activities back to applications is also made easier because there will only be one version of NEOCLASSIC.

Our experience indicates that many good research ideas come from applications, so we feel that this two-way flow of information will result in new research, as well as better applications. This is the *dominant* reason for the common version of CLASSIC.

Because NEOCLASSIC will be directly used in applica-

[1] Of course, there will still be a lag between the time when new features first show up in the research version and the time when they are approved for use in applications. We expect that this lag will be much shorter than now and the process will certainly be much easier than at present.

tions it has certain conditions placed on it that are not required of a research-only tool.

First, a development tool has to be acceptable to development organizations. This requires using an acceptable, mainstream programming language, such as C or C++. (This language requirement was one of the main reasons for producing a separate development version of CLASSIC in the first place.) For our purposes C++ has a number of advantages over C, mostly having to do with encapsulation and its type hierarchy, so we have chosen C++ as the implementation language of NEOCLASSIC.[2]

Second, a development tool has to be embeddable in applications. Our current major applications treat CLASSIC as a small part of a large system. Both the research and the development versions of CLASSIC have good interfaces to other programs, but NEOCLASSIC will have a better one, allowing programs access to the inner workings of the description logic. This will be done in a way that does not violate data encapsulation—NEOCLASSIC is a knowledge representation system, not a data structure storage system.

Programs will be able to access more of the internal objects of a NEOCLASSIC knowledge base, as C++ objects with appropriate public member functions. Of course, C++ is not type-safe, so users can puncture the encapsulation provided by the C++ objects, but this requires special action (i.e., the use of casts) is a violation of the contract that we will give to users.

Programs will also be able to monitor more of the internal workings of NEOCLASSIC, and be informed of changes to the knowledge base. This will be done in two ways: 1/ via an explanation interface that provides justifications of the inferences that NEOCLASSIC performs, and 2/ via "hooks" that provide ways for implementers to call their own functions at appropriate times. For example, hooks will allow for the creation of graphical user interfaces to NEOCLASSIC knowledge bases, both by us and by application programmers.

Further, a development tool has to have good performance, be better tested than the average research tool, and be available on several platforms. This is not a liability as far as we are concerned—we want to have a knowledge representation tool that will not break, that will not produce wrong or too-slow answers, and that can be used by a large number of people.

We will perform a full suite of tests on NEOCLASSIC, to ensure that the implementation is up to application standards. We expect that NEOCLASSIC will be faster than the LISP version of CLASSIC and comparable to the C version, although this is not guaranteed. NEOCLASSIC will run under both UNIX and WINDOWS.

[2] C++ is not the *ideal* language for our purposes. The ideal language would be strongly typed, like ML, and would also allow CLASSIC concepts to be part of its type system. Further, the ideal language would have support for rapid prototyping, such as that provided in many LISP environments. However, C++ is *acceptable* and is the language of choice at AT&T.

We are not neglecting the research aspects of NEOCLASSIC itself, however.

NEOCLASSIC is designed to be more extensible than the development version of CLASSIC. We are redesigning the core of the system so that it is easier to modify the non-core portions of the system and add other modules (like explanation). We plan on making a number of extensions to the description logic underlying NEOCLASSIC, to the point of moving away from near-completeness in some areas. For example, we expect to add a part-of hierarchy [Speel and Patel-Schneider,1994] and defaults [Nutt and Patel-Schneider,1994] to NEOCLASSIC in the near future.

NEOCLASSIC will also allow for back-end interfaces to databases and persistent storage. We plan on providing a simple persistent variant of NEOCLASSIC for use in applications as well as trying out various connections between description logics and databases.

We thus expect our research to be aided by NEOCLASSIC, in particular from the closer relationship it will foster between research and applications. We expect that the building and use of NEOCLASSIC will provide further answers to the following questions, most of which we have already pursued via the development and use of the two versions of CLASSIC.

- What characterizes a good implementation of a description logic system?
- What do applications need beyond a basic description logic system—explanation, user interfaces, programming interfaces, notification of activities?
- What can description logic research learn from description logic applications?
- How much emphasis should a description logic system place on the embedding of description logic applications in larger application systems?
- Is it possible to facilitate more widespread usage of description logics by integrating description logic systems with other AI or programming formalisms, such as rules or object-oriented languages?

References

[Baader *et al.*, 1994] Franz Baader, Maurizio Lenzerini, Werner Nutt, and Peter F. Patel-Schneider, editors. *Working Notes of the 1994 Description Logic Workshop*, Bonn, Germany, May 1994.

[Brachman *et al.*, 1991] 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 Sowa, editor, *Principles of Semantic Networks: Explorations in the representation of knowledge*, pages 401–456. Morgan-Kaufmann, San Mateo, California, 1991.

[Brachman, 1992] 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, Cambridge, Massachusetts, October 1992. Morgan Kaufmann.

[Heinsohn *et al.*, 1992] Jochen Heinsohn, Daniel Kudenko, Bernhard Nebel, and Hans-Jürgen Profitlich. An empirical analysis of terminological representation systems. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 767–773, San Jose, California, July 1992. American Association for Artificial Intelligence.

[McGuinness and Borgida, 1995] Deborah L. McGuinness and Alex Borgida. Explaining subsumption in description logics. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, Montreal, Canada, August 1995. International Joint Committee on Artificial Intelligence.

[Nutt and Patel-Schneider, 1994] Werner Nutt and Peter F. Patel-Schneider. Useful defaults in description logics. In Baader et al. [1994].

[Patel-Schneider *et al.*, 1991] 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. *SIGART Bulletin*, 2(3):108–113, June 1991.

[Resnick *et al.*, 1993] 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: Version 2.1. AI Principles Research Department, AT&T Bell Laboratories, 1993.

[Speel and Patel-Schneider, 1994] Piet-Hein Speel and Peter F. Patel-Schneider. CLASSIC extended with physical whole-part relations. In Baader et al. [1994].

[Weixelbaum, 1991] Elia S. Weixelbaum. C-Classic reference manual release 1.0. AT&T Bell Laboratories, 1991.

[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 Conference*, pages 183–193, Washington, D. C., July 1993. American Association for Artificial Intelligence.

Task Acquisition with a Description Logic Reasoner

Martin Buchheit Hans-Jürgen Bürckert Bernhard Hollunder
Armin Laux Werner Nutt Marek Wójcik

German Research Center for Artificial Intelligence - DFKI GmbH
Stuhlsatzenhausweg 3, 66123 Saarbrücken, Germany
e-mail: tacos@dfki.uni-sb.de, fax: (+49 681) 302 5341

1 Introduction

In many areas of Computer Science and Artificial Intelligence, like Programming, Databases, and Knowledge Based Systems, formalisms for modeling an application domain share a similar view of the world. They perceive a universe as consisting of objects, which are grouped into hierarchically organized classes and linked by attributes. The classes and attributes are further specified by integrity constraints and rules that can be expressed in some fragment or variant of predicate logic.

In particular, in knowledge acquisition research, which has the goal of developing knowledge representation schemes that are capable of capturing the conceptual structures of human experts, such an object-centered approach to domain modeling has proven useful. Meanwhile, a number of tools have been built that support knowledge acquisition in this paradigm (see [Gaines and Shaw,1993; Gil,1994]).

As a result, at the core of many knowledge based systems there is an object-centered domain model. It might be implemented on different kinds of platforms such as object-oriented databases, knowledge representation systems based on Descriptions Logics (DLs), like LOOM [MacGregor,1991], CLASSIC [Patel-Schneider *et al.*,1991] and KRIS [Baader and Hollunder,1991a], or even knowledge acquisition tools that allow one to execute the elicited knowledge, like KSSn [Gaines and Shaw,1993] or EXPECT [Gil,1994].

The advantage of rich domain models is that expert knowledge can be represented adequately. However, a user who wants a system to solve a particular task (e.g., booking a ticket or organizing a removal) has to know the system's domain model well in order to specify his problem appropriately. Thus, a novice user of a knowledge based system is charged with the burden of making himself familiar with the underlying model and to figure out what kind of input is expected by the system.

In this paper we present the system TACOS (= Task ACquisition for Object-centered Systems), which utilizes its domain model to interactively acquire information from the user. This acquisition process is iterated until the system can decide whether or not the task is completely specified. In particular, TACOS guides the user through an object-centered domain model and gives support in querying it and in populating it with relevant objects.

A tool like TACOS can either function as a front-end to a knowledge based system or serve as a platform for prototyping simple such systems. Because of its support for describing objects in terms of a rich object-centered domain model it can be employed for filling and maintaining a large fact base, but also for specifying a particular task. In this paper, we concentrate on the second usage because all services of TACOS come in useful.

This paper is organized as follows. In the next section we give an overview of the system. Sections 3 to 5 describe the main building blocks. Section 6 discusses further application domains and Section 7 concludes.

2 A Bird's Eye View of TACOS

The architecture. TACOS consists of three major components (see Figure 1): a domain modeling component, an inference component, and a user interface.

With the *modeling component* one sets up a model of the problem domain. A domain model consists of three parts: (1) an ontology describing the relevant *classes* with their attributes as well as simple integrity constraints that are to hold between them, (2) *assertions* introducing objects and relationships between them, and (3) a set of monotonic *inference rules*, by which additional relationships between objects can be derived. We also refer to the ontology and assertions as the *static part* of a domain model and to the rules as the *dynamic part*. The language of the static part contains the essentials of object-centered modeling formalisms like object-oriented data models, frame systems, or DLs.

Deductions from the statements in the domain model are drawn in the *inference component*, which is realized by the DL based knowledge representation system KRIS [Baader and Hollunder,1991a]. With the help of the inference component, the *user interface* produces menus suggesting assertions which the user can enter without compromising the consistency of the static part.

The functionality. In our view, a task is described by objects that are—not necessarily completely—specified in terms of the domain model. A knowledge based system responds by either (1) completing the specification,

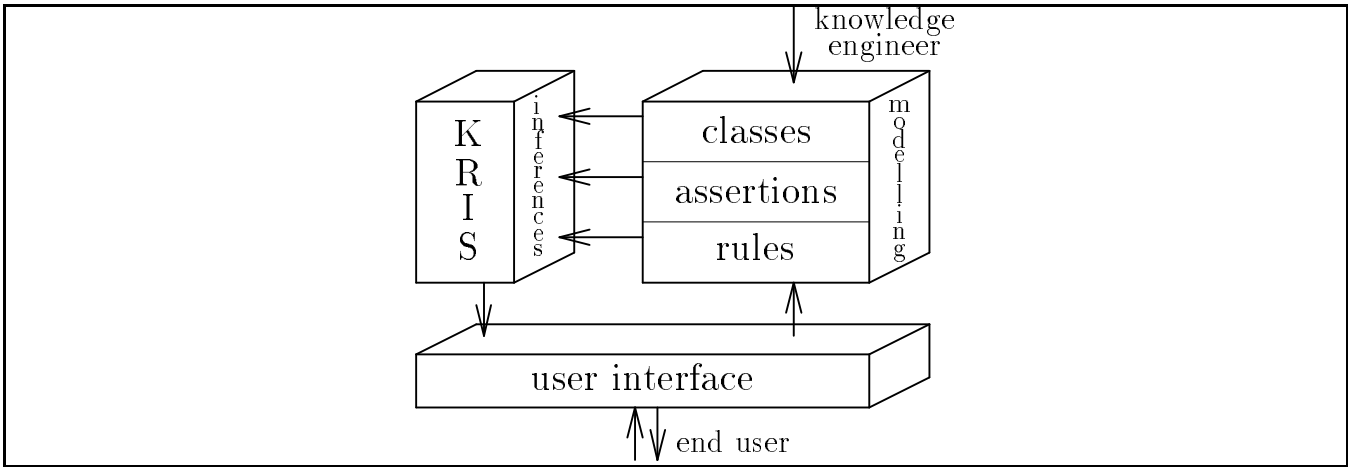


Figure 1: The architecture of TACOS.

(2) generating new objects, or (3) producing output in windows controlled by itself. A user is assisted by TACOS in specifying objects through a menu-based interface. This interface displays the current information about the known objects as specific as possible and suggests possible inputs in its menus. The suggestions encompass only meaningful items in the sense that adding them does not lead to inconsistencies with respect to the static part of the domain model. Obviously, due to the expressivity of the rule language it is impossible to restrict the suggested input further so that also inconsistencies produced by rules are prevented. However, we assume that in general the rules are written in such a way that this case does not occur. If it happens, earlier states of the session can be recovered with a backtracking mechanism.

The whole approach is flexible in that the appearance of the menus and the guidance during the acquisition process are completely derived from the domain model. It is a characteristic of TACOS that only information that is meaningful and consistent with respect to the domain description can be entered. In order to identify such information, the domain model is translated into a KRIS knowledge base. Through its inferences KRIS is able to determine into which classes a given object can still be put and which constraints can be imposed on its attributes without compromising consistency. Moreover, it can find all objects that satisfy a given set of constraints.

For two reasons a system like KRIS is an appropriate inference engine for task acquisition. The first pertains to DL systems in general: Since the purpose of the system is the assistance in specifying a task, it is confronted with incomplete information. Such incompleteness is taken into account by the open world semantics of DLs. The second is more specific: To detect all possible inconsistencies, the reasoning process must not only be sound, but also complete. There are only a few DL systems with this property, and KRIS is one of them.

In the next three sections we will illustrate the languages, the components, and the services of TACOS with a running example taken from a scenario where a user specifies a transport task to be executed by a forwarding agency. We have implemented this scenario in TACOS at DFKI.

3 The Domain Modeling Component

For each part of the modeling component (see Figure 1) TACOS provides a particular language: (1) a class description language, (2) an assertional (object description) language, and (3) a rule language.

With the *class description language* one fixes the structure of a problem domain. Basically, a class describes a set of objects in terms of more general classes and of attributes. Figure 2 contains the description of three classes in our forwarding domain. Classes come in two variants: basic classes and defined classes. The description of a *basic* class (indicated by the keyword Class) imposes *only* necessary conditions for class membership. So, the description of the basic class *task* says that for every task there is exactly one person which is the customer, or formally, if an object is an instance of *task*, then it must have exactly one filler of the attribute *customer*, and the filler must be an instance of the class *person*. The class *transport-task* is a specialization of *task*. Hence, *transport-task* inherits from *task* the attribute *customer*. In addition, a *transport-task* has the attributes *cargo*, *collect-from*, *deliver-to*, and *distance*, which are subject to the following constraints: (1) each *cargo* is a *good*, and there is at least one *cargo*, (2) there is exactly one *collect-from* and one *deliver-to*, which are *location*s, and (3) there is exactly one *distance*, which is an *integer*. The *distance* is computed by the system and cannot be input by the user. The computation is specified by rules (see below).

The class *domestic-transport-task* is a specialization of *transport-task* in that the range restrictions for the inherited attributes are refined: for a *domestic-transport-task* the fillers for the two attributes *collect-from* and *deliver-to* must not only be *location*s, but additionally must be instances of *inland* which is a subclass of *location*. Since *domestic-transport-task* is a *defined* class (indicated by the keyword Defined-Class), the description

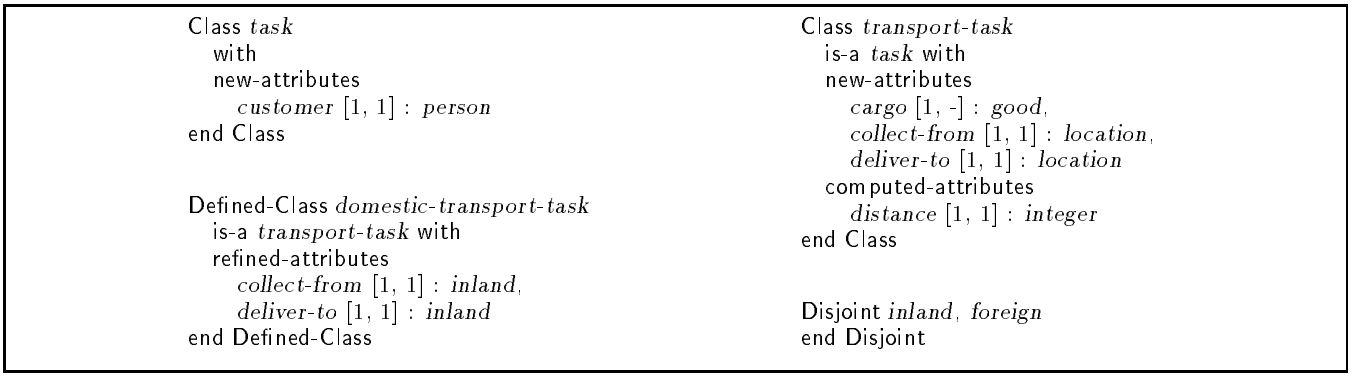


Figure 2: Classes in our forwarding domain.

specifies both necessary *and* sufficient conditions for class membership. This means that any object (1) which belongs to *transport-task*, and (2) whose fillers for *collect-from* and *deliver-to* belong to *inland*, is also an instance of *domestic-transport-task*. Defined classes can be seen as predefined queries or "views" [Buchheit *et al.*,1994], which help in structuring the domain. They can be combined to complex queries and provide macros to be used in the preconditions of rules.

In addition to the means for describing classes discussed so far, one can also define *enumeration classes*, declare a group of basic classes as *disjoint* and specify that one basic class covers a group of other basic classes. Furthermore, there are the built-in classes *string* and *integer*, as well as intervals of *integer*s.

In the *assertional language* one describes a given state of affairs in the problem domain by (1) inserting an object into a class (e.g., declare *my-order* as a *transport-task*) and (2) relating objects through attributes (e.g., make *Berlin* the filler of *deliver-to* for *my-order*). Usually, TACOS will start with an initial knowledge base (KB for short) containing facts about individuals that are relevant for the domain. In our forwarding example we assume that there are facts about cities such as $Bonn : inland$, $Berlin : inland$, $London : foreign$, $(Bonn, 300.000) : inhabitants$, etc.

In the *rule language* one can specify conditions for deriving new assertions, generating fresh objects, or calling functions in the host language (LISP). A rule consists of a *condition part* and a *consequence part* (cf. the rules in Figure 3).

Suppose that *my-order* is a *transport-task*, and that *Bonn* and *Berlin* are the corresponding fillers for *collect-from* and *deliver-to*. Then the first rule can be applied to compute *my-order*'s filler of the attribute *distance* as follows: as the first three conditions are entailed by the static part (if the variables x, y, and z are substituted by *my-order*, *Bonn*, and *Berlin*), the function *compute-distance* is called with the arguments *Bonn* and *Berlin*, and the resulting value, say *598*, is bound to the variable d; thus, the rule fires and the fact that *my-order*'s filler of *distance* is *598* is added to the current set of assertions.

The effect of applying the second rule is the execution of a function, which is called for its side effects. As *my-order*, in our example, is a *transport-task* whose fillers for *collect-from* and *deliver-to* are instances of *inland*, it is recognized as a *domestic-transport-task*.[1] Moreover, the filler of *distance* is not greater than *800*. Thus all preconditions are satisfied and the function *collect-customer-data* is called with argument *my-order*.

The class description language of TACOS captures essentially what can be expressed in common object-centered representation formalisms like frame languages, object-oriented data bases, and DLs. The rule language, however, goes beyond the rules generally available in data bases and DLs. Since it allows for the generation of fresh objects, it is Turing complete even if we ignore the integration with the host language.

4 The Inference Component

The purpose of the domain modeling component is to state facts about a problem domain. To draw conclusions from the represented knowledge we utilize the inference mechanism of KRIS. As a full-fledged DL system, KRIS provides three languages for the representation of knowledge: (1) a terminological language, (2) an assertional language, and (3) a rule language.

In the *terminological language* one can define concepts by means of complex expressions, which are built up from other concepts using constructs such as conjunction, disjunction, negation, range and exists restrictions, and cardinality constraints. While classes can be translated into concepts in a straightforward manner, the converse is not true, as the terminological language of KRIS contains a number of constructs that have no counterpart in TACOS. The *assertional language* and the *rule language* in KRIS are similar to the ones in TACOS, except that the former are based on concepts and the latter on classes. Like a domain model, a KRIS *knowledge base* comprises a static and a dynamic part (static and dynamic KB for short). The *static* KB consists of a set $\mathcal{T}$ of concept declarations and a set $\mathcal{A}$ of assertions, the *dynamic* KB consists of a set of rules.

The static parts of KBs are interpreted under the standard DL semantics, which identifies them with a set of

[1] This fact is crucial, e.g., if the agency can carry out domestic transport tasks only.

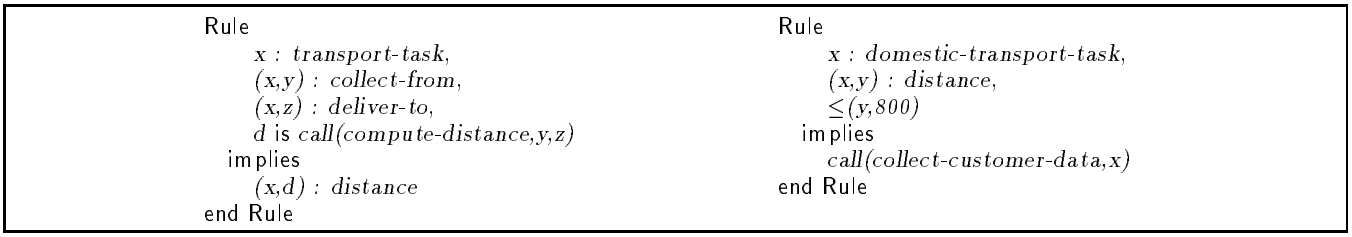


Figure 3: Two rules in our forwarding domain.

first order formulas (see [Nebel,1990]), and inferences are defined with respect to this semantics. Elementary inference problems are to decide, given a static KB, whether one concept subsumes a second, whether an object is an instance of a concept, and whether the KB is consistent at all. More complex reasoning services, like determining all subsumption relationships between the concepts in the KB (classification) and finding all memberships of objects in concepts (realization) are based on the elementary inferences.

When considering inferences, a characteristic comes into play that distinguishes DLs from databases: a KB is not viewed as determining a single structure, but as having—possibly infinitely many—different models: for inferences *all* first order models of the KB are taken into account. For this reason, this semantics is called an "open world semantics" as opposed to a "closed world semantics" admitting only a single model.

For the purpose of task acquisition a closed world semantics would be inappropriate because a task is not completely specified before the very end of a session and various completions are conceivable. In particular, it is not justified to conclude negative information from the absence of information, like a conventional database system would do. For example, the fact that up to a given moment no dangerous cargo has been specified for a transport task does not necessarily imply that all cargo will be safe. Thus, a task acquisition system has to take into account all possible completions in its reasoning process.

In almost all implemented DL systems, inferences are realized by sound but *incomplete* algorithms. This, for example, means that if a consistency checking procedure reports a static KB to be inconsistent, one can rely on this answer. However, if the procedure reports consistency, nothing can be concluded: the KB may or may not be consistent. For the use in TACOS, a DL system with incomplete algorithms is obviously insufficient since it could not prevent a user from entering inconsistent information. However, KRIS is an appropriate choice because its algorithms are sound *and* complete [Baader and Hollunder,1991b].

To achieve the functionality of TACOS, we have extended KRIS to perform additional reasoning services, which are based on elementary inferences, but which usually are not provided by DL systems. Among them are the following, where $\Sigma = \langle \mathcal{T}, \mathcal{A} \rangle$ is a static KB, C a concept occurring in $\mathcal{T}$, R an attribute, and a, b objects:

Possible concepts: C is a *possible concept* for a if $\langle \mathcal{T}, \mathcal{A} \cup \{a : C\} \rangle$ is consistent;

Possible range restrictions: C is a *possible range restriction* of R for a if $\langle \mathcal{T}, \mathcal{A} \cup \{a : \forall R.C\} \rangle$ is consistent;

Possible fillers: b is a *possible filler* of R for a if $\langle \mathcal{T}, \mathcal{A} \cup \{(a, b) : R\} \rangle$ is consistent;

Plausible fillers: b is a *plausible filler* of R for a if for all concepts D having a counterpart in the class language of TACOS we have that $\langle \mathcal{T}, \mathcal{A} \cup \{(a, b) : R\} \rangle \models b : D$ if and only if $\langle \mathcal{T}, \mathcal{A} \rangle \models b : D$.

Possible concepts, range restrictions and fillers for an object are computed when TACOS determines which information a user can enter safely. These inferences are based on consistency checks. The check for plausible fillers, however, is more involved. Intuitively, b is plausible if it already has all the properties—expressible as memberships of b in some D—that a filler of R for a is required to have. Computationally, this check is realized by abstracting the constraints imposed on the fillers of R into a concept and retrieving its instances. Looking up plausible fillers is helpful when a user combines entering information with querying the domain model for interesting objects.

Rules in KRIS are handled analogously to rules in TACOS. The key inference for deciding whether a rule is applicable is the test whether an object is an instance of a concept. A set of rules is applied by executing each rule with all possible instantiations of its variables (see [Hanschke and Hinkelmann,1992]).

5 The User Interface

The user interface assists a user who wants to specify a task in populating and querying a domain model. It has been designed so that it guides the user during the acquisition phase, passes the information acquired from the user to the domain modeling component, and displays appropriate help texts whenever the user needs additional information (e.g., the description of objects or classes).

In the sequel we illustrate the first of these functionalities with an example session in our forwarding scenario. TACOS realizes this functionality by displaying the current information about objects, and by proposing to the user new pieces of information to add. In order to present only information as specific as possible and to offer all possibilities of entering compatible information, the user interface triggers inferences in KRIS.

5.1 Object windows

When data for an object is retrieved from the KB an *object window* showing the information available about the object appears on the screen. An object window consists of an *object level* part and an *attribute level* part. The first contains the name of the object and a list of the (most specific) classes it belongs to; the second contains descriptions of all attributes of the object. Each class, attribute, and object name in the object window is mouse-sensitive. When clicking on it a pull-down menu pops up offering operations that can be applied to that entity. We explain the most important operations continuing the example from our forwarding domain. Suppose, we have started with the object *my-order* of the class *task*. The system displays the following information in an object window.

object:	*my-order*		class(es): *task*		
attr.:	name	range	min	max	filler(s)
	customer	*person*	1	1	—

The object level. On the object level the only data that can be modified is the list of classes the object belongs to. The user can refine a class from the list and extend the list.

Refining a class means replacing it by a more specific one. To this end the user chooses an element from a list of classes offered to him by the system. The list is computed by KRIS, which tests for each concept corresponding to an immediate subclass of the class to be refined whether it is a possible concept for the object in hand (see Section 4). The support of KRIS allows the user to focus on a restricted set of relevant items and guarantees that only consistent information is entered. Assume there are the following subclasses of *task*: *transport-task*, *storing-task* (which are disjoint), and *single-task*, *recurring-task*. If we want to refine the class *task* of *my-order*, all of them are possible and the system offers the four of them. Suppose we choose *transport-task*. As additional attributes are declared on this class the system updates the displayed information:

object:	*my-order*		class(es): *transport-task*		
attr.:	name	range	min	max	filler(s)
	customer	*person*	1	1	—
	cargo	*good*	1	∞	—
	collect-from	*location*	1	1	—
	deliver-to	*location*	1	1	—
	distance	*integer*	1	1	—

Extending the class list consists in adding a class that is independent of those the object is an instance of. Again, the candidates are determined by KRIS, which proceeds in three steps. First it computes the set of possible concepts for the object in hand. Then, this set is filtered to yield those concepts that neither subsume nor are subsumed by a concept corresponding to a class in the current list. Finally, KRIS returns the most general concepts in the reduced set. Assume that in our example we want to extend the list of classes *my-order* belongs to. In this case TACOS offers only *single-task* and *recurring-task*, since *storing-task* is disjoint from *transport-task*, to which *my-order* belongs.

The attribute level. In the attribute level part the following items are displayed for each attribute of an object: the attribute's name, a list of classes restricting its range, the cardinality constraints, i.e., a lower and an upper bound for the number of fillers—indicated by the keywords min and max—and a list of the currently known fillers. Similar to the object level, one can refine a class in the range and extend the range by another class. Additionally, the user is allowed to strengthen the cardinality constraints and to add fillers.

Refining a class in the range of an attribute is similar to refining a class in the object level part. The possible refinements are computed by KRIS based on the test for possible range restrictions. For instance, suppose that in our example we have decided to refine the range of the attribute *collect-from* of *my-order*. Assume further that *location* has the two immediate subclasses *foreign* and *inland*, which are disjoint. Then both are suggested as possible refinements. For the sake of our example, suppose we choose *inland*.

Analogously, *extending* a list of range restrictions resembles extending the list of classes an object is member of. Again, the computation of the candidate classes suggested by TACOS involves the test for possible range restrictions.

Note that neither refining a class in the list of range restrictions nor extending the list results in displaying new attributes for the current object. However, as will be discussed below, it may result in additional attributes of the fillers when the consequences of these operations are computed.

The next operation we want to illustrate is the *introduction of attribute fillers*. As filler of an attribute, the user can either create a new object, i.e., one that does not yet appear in the KB, or he can take an existing object.

In the second case, he can choose between (1) possible fillers, which comprise all objects that can be added as fillers without turning the static KB into an inconsistent state, and (2) plausible fillers, which are the subset of possible fillers already satisfying the constraints imposed on arbitrary fillers of the attribute. They are computed by means of the tests for possible and plausible fillers in KRIS. Coming back to the example, recall that in our KB *Berlin* and *Bonn* are instances of *inland*, and *London* is an instance of *foreign*. Then the only possible (and plausible) fillers of *collect-from* for *my-order* are *Berlin* and *Bonn*, since we already have refined the range of *collect-from* to *inland*. For the attribute *deliver-to*, also *London* is plausible. For our example we choose *Bonn* and *Berlin*.

In the above example, all possible fillers are also plausible. In order to see the difference between the two, assume that there is also an instance *joe* of the class *person* in the KB, and assume further that *person* and *location*

are not disjoint. Then also *joe* would be a possible filler, but not a plausible filler.

Finally, the user can *strengthen the cardinality constraints* imposed on the number of fillers of a given attribute. More precisely, he can increase the lower bound and decrease the upper bound. Again the possible values are offered in a mouse-sensitive menu. In our example, we can change both min and max of *cargo*. Since min and max coincide for any of the remaining attributes, they cannot be changed without running into an inconsistent state. When we attempt to modify them, TACOS will inform us about this fact. Moreover, it will not accept additional fillers for *collect-from* and *deliver-to*, because the cardinality constraints require exactly one.

5.2 The reasoning process

During the acquisition process, new information entered by the user is continuously passed to the domain modeling component. However, the object windows do not immediately display inferred facts. In order to view such derived information, the user has to start an update process. The process consists of two interleaving phases: realization of objects and application of rules. When the update has been completed successfully, the new state of the KB is pushed on a stack containing the states reached so far.

In the *realization phase*, all instance relationships between the objects and the classes occurring in the KB are computed. As a consequence, the system presents the most specific information about each displayed object. In our example, TACOS recognizes that *my-order* not only belongs to *transport-task*, but also to *domestic-transport-task*. Since the only filler of *deliver-to*, namely *Berlin*, is an instance of *inland*, the system concludes that every filler of *deliver-to* belongs to *inland*; in other words, the range of *deliver-to* is specialized to *inland*.

The *application of rules*, as described in Section 3 may cause functions to be called and new assertions to be added. Any newly added assertions may in turn require another classification phase. In our example, the rules in Figure 3 are applicable. As *my-order* is an instance of *transport-task* (although just classified as *domestic-transport-task*) and has fillers of *collect-from* and *deliver-to*, the value computed by *compute-distance(Bonn,Berlin)*, say *598*, is bound to the variable *d* and the first rule fires. Hence the assertion $(my\text{-}order, 598) : distance$ is added to the KB. Now the preconditions of the second rule are fulfilled and the function call in its consequence part is executed. After the update the object window of *my-task* looks as follows.

object:	*my-order*		class(es): *dom.-tr.-task*		
attr.:	name	range	min	max	filler(s)
	customer	*person*	1	1	—
	cargo	*good*	1	∞	—
	collect-from	*inland*	1	1	*Bonn*
	deliver-to	*inland*	1	1	*Berlin*
	distance	*integer*	1	1	*598*

In a similar way as discussed so far, the user will enter during the next steps his personal data and a description of the cargo. If the task complies with the conditions for acceptance, as specified by rules, they can be forwarded to a data base that keeps track of accepted orders.

5.3 Handling inconsistencies

When TACOS is started with an initial KB it computes its consequences by repeated classification and rule application. After these operations it should arrive at a consistent state, since otherwise no meaningful work can be done.

From this moment on, the user is allowed to enter new assertions. As pointed out before, he cannot input arbitrary facts, but must choose from lists of possible items presented by the system. The underlying DL inference component guarantees that adding any of the items offered keeps the static KB consistent. However, this is no more true when also the *dynamic* part is taken into account, because arbitrary assertions can be added through the application of rules.

Therefore, after each reasoning step (see Section 5.2) the inference component checks whether the static KB is consistent. When an inconsistency is detected, the user is informed and the system backtracks to the previous state.

6 Discussion

We have experimented using TACOS with different domain models. One of them describes the possible tasks that can be executed by a shipping company and has served as illustration in this paper.

As another experiment we have translated the Wines knowledge base, originally formulated in CLASSIC [Brachman *et al.*,1991], into TACOS. The Wines KB contains descriptions of meals and wines together with rules that impose constraints on the wines to accompany the meal. Loaded into TACOS, a user can specify his meal while the system gives him advise on the beverage. In this scenario, two services can well be demonstrated: on the one hand, through the various refinement operations, one can stepwise traverse the space of possible meals, on the other hand, constraints on wines are accumulated and combined into a query, which then is used to find a particular wine object that satisfies it.

Although a toy example, the Wines scenario is prototypical for a whole class of applications, whose common characteristic is the interactive composition of a complex object, and where a tool like TACOS can give support. Such an object can be a meal or a transportation task, but it can equally well be a computer or some other complex technical system to be configured.

We also conceive of other areas where the ideas realized in TACOS could be applicable. As schemas of object-oriented data bases tend to be rich in semantic information, there is again the problem for the casual user of getting acquainted with a domain model, if he wants to formulate sensible queries. In such a situation, a system like TACOS could be employed to support a user in

specifying not his inputs, but the objects he wants to retrieve.

We assume that the schema and its integrity constraints can be mapped into a TACOS domain model. Considering the entities described by a set of object windows as variables to be instantiated, one could incrementally formulate a query asking for families of objects. The power of the language would at least be sufficient to formulate conjunctive queries over unary and binary predicates (classes and attributes). Due to the assistance through a reasoner, it is impossible to come up with an inconsistent query, that is, a query that cannot have instances because of the integrity constraints of the database.

7 Conclusion

We have presented TACOS, an information acquisition interface, which receives its power from the inferential capabilities of the DL system KRIS.

Y. Gil has formulated a set of demands on knowledge acquisition tools [Gil,1994]: (1) maximum guidance of the user, (2) robustness, in the sense that erroneous input is tolerated, or better, prevented, and (3) flexibility, in the sense that the system is applicable to a wide range of domains. We think that these requirements give also a good guideline for the development of task acquisition systems.

The TACOS system meets each of them in a specific way: It guides the user by displaying input patterns and actively suggesting items for input. It does not allow information to be entered that is meaningless with respect to the domain model, so that the need for error recovery is reduced. Finally, since its functionality is parameterized by a largely declarative domain model, it can be easily adapted to different applications.

References

[Baader and Hollunder, 1991a] F. Baader and B. Hollunder. $\mathcal{KRIS}$: $\mathcal{K}$nowledge $\mathcal{R}$epresentation and $\mathcal{I}$nference $\mathcal{S}$ystem. *SIGART Bulletin*, 2(3), 1991.

[Baader and Hollunder, 1991b] F. Baader and B. Hollunder. A terminological knowledge representation system with complete inference algorithms. In M. Richter and H. Boley, editors, *Proc. of PDK'91*, Kaiserslautern, Germany, 1991.

[Brachman *et al.*, 1991] 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 J. Sowa, editor, *Principles of Semantic Networks*. Morgan Kaufmann, San Mateo, Calif., 1991.

[Buchheit *et al.*, 1994] M. Buchheit, F. M. Donini, W. Nutt, and A. Schaerf. Refining the structure of terminological systems: Terminology = Schema + Views. In *Proc. of AAAI'94*, Seattle (Washington, USA), 1994.

[Gaines and Shaw, 1993] B. Gaines and M. Shaw. Knowledge acquisition tools based on personal construct psychology. *The Knowledge Engineering Review*, 8(1), 1993.

[Gil, 1994] Y. Gil. Knowledge refinement in a reflective architecture. In *Proc. of AAAI'94*, Seattle (Washington, USA), 1994.

[Hanschke and Hinkelmann, 1992] P. Hanschke and K. Hinkelmann. Combining terminological and rule based reasoning for abstraction processes. In *Proc. of GWAI'92*, Bonn, Germany, 1992.

[MacGregor, 1991] R. MacGregor. Inside the LOOM description classifier. *SIGART Bulletin*, 2(3), 1991.

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

[Patel-Schneider *et al.*, 1991] P. Patel-Schneider, D. L. McGuinness, R. J. Brachman, L. Alperin Resnick, and A. Borgida. The CLASSIC knowledge representation system: Guiding principles and implementation rational. *SIGART Bulletin*, 2(3), 1991.

Part-of Reasoning in Description Logics: A Document Management Application

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

Lin Padgham
Department of Computer Science
Royal Melbourne Institute of Technology
Melbourne, VIC 3000, Australia
linpa@cs.rmit.edu.au

Abstract

The notion of giving support to composite objects in systems is gaining strength in a number of areas including description logic systems. In this paper we examine the usefulness of an earlier proposed approach which introduces part-of reasoning in a description logic in a document management application. We identify key issues in both expressivity and inferencing.

1 Introduction

The notion of composite objects, or objects made up of parts, is one which is gaining strength in a number of areas. Within knowledge representation and reasoning there has been an interest in the description logics community to extend the logics to include the ability to represent and reason about composite objects. There have been a number of different approaches including an extension which allows for treatment of collections as individuals [1] and an integration of a description logic with a rule-based formalism, where properties and inferences about part-whole relations are expressed in the rule-base [2]. The KOLA system [3] also incorporated some aspects of part-whole relations found necessary in a medical application.

In our own previous work [5] we have concentrated on defining a *part-of* hierarchy which is like the *is-a* hierarchy in that it organizes concept descriptions with respect to an important relation, and then uses this organization/relation to define inferences about individuals and concepts. The language component of the theory that we defined has some similarities to the language used by [6] in an extension to CLASSIC.

In this work we examine the usefulness of our approach for use in a particular application. Such examination is intended to guide further work in developing additional expressivity and additional inferencing mechanisms, based on the part-of relation. The application we have chosen is that of a project management system, where system support is required for such things as definition of document types, classification and checking of documents for completeness with respect to definition, inheritance of attributes between parts and wholes, and declaration and management of whole-part relationships. This application is being developed in cooperation with a local company, and the requirements are based on discussions regarding their needs, and documentation describing how things are currently done manually. In this paper we define extensions to the language in [5] which allow us to define templates as desired, and we discuss how the system can be used to make the inferences and answer the queries desired by users. For details and examples we refer to [4].

2 Extending the Language

The language in [5] allows us to do three things w.r.t. parts in the concept definitions. We can define a part name and the concept which the part belonging to that part name must belong to (e.g. (**part** *title-p string*)); we can define how many parts there should be of a certain kind (e.g. (**parts** *1 title-p*)) and we can describe constraints between parts within a composite (e.g. (**pp-constraint** *larger-font title-p abstract-p*)).

In [5] we considered the case where we knew for each part name in the definition of a concept exactly how many parts there were. This was too constraining for the application considered here. Thus we replace the **parts**-construct with two constructs for giving cardinality ranges for parts: **atleastp** and **atmostp**. They are similar to **atmost** and **atleast** for ordinary roles.

The language in [5] allows for differentiation between different kinds of parts by using different labels. Thus we can have (**part** *title-p string*) and (**part** *abstract-p abstract*) in the same definition, indicating that there is both a title part and an abstract part. However, we have no way to differentiate between different instances of the same part label. This was necessary in our application, in particular to support ordering of parts at the individual level - it is not acceptable for the sections of a document to appear in random order. We decided to extend the language by allowing a number attached to the part name in some instances. To allow this ordering to extend over all the parts of a composite in a uniform way, we also introduce the notion of an order constraint between two different kinds of parts. This is written as in (**order-constraint** *section-p reference-p*). This indicates that in the ordering of parts, *section-p* comes before *reference-*

concept ::=
- $\top$
- $\bot$
- *atomic-concept*
- (**and** *concept*$^+$)
- (**all** *role concept*)
- (**atleast** *number role*)
- (**atmost** *number role*)
- (**allp** *part atomic-concept*)
- (**atleastp** *number part-name*)
- (**atmostp** *number part-name*)
- (**pp-constraint** *role part part*)
- (**order-constraint** *part part*)
- (**same-filler** *attr-path attr-path*)
- (**aggregate** *part attribute role*)

role	::=	*identifier*
attribute	::=	*identifier*
atomic-concept	::=	*identifier*
part	::=	*part-name*
		| *part-name:pos-number*
part-name	::=	*identifier*
attr-path	::=	*attribute*
		| *part.attribute*
number	::=	*non-negative-integer*
pos-number	::=	*strictly-positive-integer*

Figure 1: Terminological language: syntax.

p. If there are several sections, indicated as *section-p:1*, *section-p:2*, etc. then these will be ordered by number, and all will come before all *reference-p*.

The third extension we introduce gives the ability to define inheritance information allowing us to infer attribute[1] values of parts based on those of wholes, or vice versa, or attribute values of parts, based on those of other parts. The first construct we introduce is **same-filler** which is similar to *same-as* in CLASSIC, but with restricted path length. It allows us to state such things as (**same-filler** *projectnumber section-p.projectnumber*), indicating that the projectnumber attribute of a section part has the same value as the projectnumber attribute of the composite in which this definition is found. We also introduce the **aggregate**-construct which allows us to aggregate a set of attribute values for a particular kind of parts and declare these to be role fillers of a particular role within the composite object. Thus (**aggregate** *document-p responsible responsibles*) in the definition of a folder indicates that the value of the role responsibles for this composite is an aggregate of the responsible attribute in each of the document parts.

The syntax of the full language[2] we use is defined in figure 1. An interpretation of the language consists of a tuple $< \mathcal{D}, \ll, \varepsilon >$, where $\mathcal{D}$ is the domain of individuals, $\ll$ is a total[3] order on $\mathcal{D}$ and ε the extension function. The semantics for the different constructs are shown in

[1] Attributes are roles which can have exactly one filler.

[2] In [5] the **allp** construct was called **part**.

[3] This constraint can be loosened and will be in practice.

$\varepsilon[(\textbf{and } A\ B)] = \varepsilon[A] \cap \varepsilon[B]$
$\varepsilon[(\textbf{all } r\ A)] =$
$\{\ x \in \mathcal{D} \mid \forall\ y \in \mathcal{D}\colon <x,y> \in \varepsilon[r] \rightarrow y \in \varepsilon[A]\}$
$\varepsilon[(\textbf{atleast } m\ r)] =$
$\{\ x \in \mathcal{D} \mid \sharp\ \{\ y \in \mathcal{D} \mid <x,y> \in \varepsilon[r]\} \geq m\ \}$
$\varepsilon[(\textbf{atmost } m\ r)] =$
$\{\ x \in \mathcal{D} \mid \sharp\ \{\ y \in \mathcal{D} \mid <x,y> \in \varepsilon[r]\} \leq m\ \}$
$\varepsilon[(\textbf{allp } p\ A)] =$
$\{\ x \in \mathcal{D} \mid \forall\ y \in \mathcal{D}\colon\ y \lhd_p x \rightarrow y \in \varepsilon[A]\}$
$\varepsilon[(\textbf{atleastp } m\ n)] =$
$\{\ x \in \mathcal{D} \mid \sharp\ \{\ y \in \mathcal{D} \mid y \lhd_n x\ \} \geq m\ \}$
$\varepsilon[(\textbf{atmostp } m\ n)] =$
$\{\ x \in \mathcal{D} \mid \sharp\ \{\ y \in \mathcal{D} \mid y \lhd_n x\ \} \leq m\ \}$
$\varepsilon[(\textbf{pp-constraint } r\ p1\ p2)] =$
$\{\ x \in \mathcal{D} \mid \forall\ y1,y2 \in \mathcal{D}\colon$
$(y1 \lhd_{p1} x \wedge y2 \lhd_{p2} x) \rightarrow <y1,y2> \in \varepsilon[r]\}$
$\varepsilon[(\textbf{order-constraint } p1\ p2)] =$
$\{\ x \in \mathcal{D} \mid \forall\ y1,y2 \in \mathcal{D}\colon$
$(y1 \lhd_{p1} x \wedge y2 \lhd_{p2} x) \rightarrow y1 \ll y2\ \}$
$\varepsilon[(\textbf{same-filler } p1.a1\ p2.a2)] =$
$\{\ x \in \mathcal{D} \mid a1(p1(x)) = a2(p2(x))\ \}$
$\varepsilon[(\textbf{aggregate } p\ a\ r)] =$
$\{\ x \in \mathcal{D} \mid \forall\ y,z \in \mathcal{D}\colon$
$(y \lhd_p x \wedge <y,z> \in \varepsilon[a]) \rightarrow <x,z> \in \varepsilon[r]\}$

Figure 2: Terminological language: semantics.

figure 2. For convenience we write $y \lhd_n x$ for $<y,x> \in \varepsilon[n]$ where $n \in \mathcal{N}$. We define $y \lhd_{n:m} x$ as $(y \lhd_n x \wedge \sharp\ \{\ z \in \mathcal{D} \mid z \lhd_n x \wedge z \ll y\} = m - 1)$. Subsumption is defined as usual but taking into account the part names and the order relation. A subsumes B iff $\varepsilon[B] \subseteq \varepsilon[A]$ for every interpretation $<\mathcal{D},\ll,\varepsilon>$.

Also the assertional language is modified slightly, but we refer to [4] for details.

3 Inferences and Queries

In addition to examining what the application needed from the language in order to be able to describe its constructs, we have also looked at what questions users wish to ask, or what inferences are useful in this application and have relevance to the notion of part-of or composite objects.

1. 'Is this individual part of that individual?' or more generally 'To which wholes does this individual belong?' Such questions are important for instance to find out which compositions (e.g. documents) change when a particular part (e.g. a section) is updated.

2. 'Does this individual belong to a particular concept?' This question is important in reasoning about why an individual was created and how to use the individual.

3. 'Is this concept a building block for another concept?' or 'What are the possible building blocks for a particular concept?' Answers to these questions provide information about the usefulness of individuals belonging to the first concept (e.g. a section)

to be used as components for composite individuals belonging to the second concept (e.g. a document).

4. 'Build a new individual belonging to a particular composite concept given the parts.' This inference allows us to instantiate composite individuals (e.g. a document) given the parts (e.g. abstract and sections). It allows us to create compositions in a bottom-up way.
5. 'Are there missing parts to build an individual belonging to a particular concept?' and 'Of what kind are these missing parts?' This inference allows us to make use of the templates described by the composite concept definitions (e.g. documents). It gives support for creating compositions in a top-down way.

In [4] we show how the notions of *part-of hierarchy*, *composes* and *compositional inferencing* as defined in [5] can be extended to provide answers to all but the last query. Here we briefly mention the new inference which is needed to answer the last query.

3.1 Completion

A common way of writing documents is by using a template. A document individual is then created following the document concept description, but its parts still have to be instantiated. Important in this method is the fact that at each moment we want to be able to find out what parts are still missing for the document, given that some parts are already written. The information we need is of which kind these missing parts necessarily should be, and what the necessary relationships should be among these parts and between these parts and the already available parts. We also do not want unmotivated new parts.

The notion of *candidate completion* allows us to express this information need formally. A candidate completion is an *Abox* which represents a proposal for an instantiation of the missing parts and thus a possible way to 'complete' the composition.

A candidate completion of a set of individuals and a concept C with respect to an original *Abox* Δ is a pair $<\Sigma,\alpha_\Sigma>$ such that α_Σ is a set of (individual, part name)-pairs and Σ is a smallest *Abox* such that (i) Σ contains the original *Abox* Δ, (ii) α_Σ contains all the new individuals used to complete the composition (and no others) and these are all defined in Σ, and (iii) we can use the new individuals together with the original set of individuals to compose (see [5] for the definition) the given concept C in Σ. Formal details are given in [4].

For some *Abox* Δ, set of individuals, and concept C there may not exist a candidate completion. This can for example happen if some constraints between the given parts are not satisfied. It is also possible that $<\Delta,\emptyset>$ is the only candidate completion, as in the case where the set of original individuals can be used to compose concept C with respect to the original *Abox* Δ. In this case Δ is the smallest *Abox* satisfying the conditions and no new individuals need to be introduced. There may also be many possible candidate completions. In [4] we have defined a preference relation based on the intuition that we prefer to have as general as possible new individuals.

4 Discussion and Conclusion

In this paper we have examined the usefulness of the approach proposed in [5] in a document management application. Although this application is not yet fully implemented, we have identified key issues in both expressivity and inferencing. As implementation progresses we expect to identify further strengths and weaknesses of this approach.

With respect to expressivity we found that the language should be extended in three ways. First, there was a need to have a construct allowing for cardinality ranges of parts. Further, the application required the possibility to order different parts. Finally, value propagation (or inheritance via part-of) was considered necessary. We introduced new constructs to reflect these needs, and indicated how these can be incorporated in the framework of [5].

For many of the important questions and inferences the notions such as compositional inclusion, composes and compositional inferencing introduced in [5] were found to be useful and we extended them to conform with the new language. Further, we defined the notion of candidate completion which is an important inference when creating compositions in a top-down way. It allows us to know at all times which kind of parts are missing for a particular composite individual.

References

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

[2] Hanschke, P., *A Declarative Integration of Terminological, Constraint-based, Data-driven, and Goal-directed Reasoning*, Ph.D. thesis, University of Kaiserslautern, Germany, 1993.

[3] Jang, Y., Patil, R., 'KOLA: A knowledge organization language', *Proceedings of the 13th Symposium on Computer Applications in Medical Care*, pp 71-75, 1989.

[4] Lambrix, P., Padgham, L., 'Analysis of Part-of Reasoning in Description Logics for Use in a Document Management Application', 1995.

[5] Padgham, L., Lambrix, P., 'A Framework for Part-Of Hierarchies in Terminological Logics', Doyle, Sandewall, Torasso, eds, *Principles of Knowledge Representation and Reasoning: Proceedings of the Fourth International Conference - KR 94*, pp 485-496, Bonn, Germany, 1994.

[6] Speel, P.-H., Patel-Schneider, P.F., 'CLASSIC extended with whole-part relations', *Proceedings of the International Workshop on Description Logics*, pp 45-50, Bonn, Germany, 1994.

Description Logic-based Configuration for Consumers

Deborah L. McGuinness and **Lori Alperin Resnick**
AT&T Bell Laboratories
600 Mountain Avenue
Murray Hill, NJ 07974
{dlm,resnick}@research.att.com

Description logic-based configuration applications have been used within AT&T since 1990 to process over two and a half billion dollars worth of orders. While this family of applications[5] has widely acknowledged importance, it is difficult to use for pedagogical purposes since the typical product configured is a highly interconnected, complicated technical piece of equipment like the DACS IV–2000.[1] We have developed a smaller–scale configuration application that has analogous reasoning processes but a more approachable domain—that of building home theater systems. This application provides a platform for explaining how Description Logic-based Systems (DLSs) work, in our case the CLASSIC knowledge representation system[1], and how they can support industrial applications like configuration.

CLASSIC[2] is an object–centered representation and reasoning tool with a formal foundation in description logic. CLASSIC and many DLSs are particularly well suited for applications in areas like configuration that must

1. encode rich class and object descriptions;
2. provide active inference (such as automatic classification of classes and objects into a generalization hierarchy, rule firing and maintenance, inheritance, propagation, etc.);
3. explain the reasoning process;
4. handle an incomplete and incrementally evolving knowledge base; and
5. handle errors in a way that keeps the knowledge base consistent, but also provides useful information to the user.

We will provide some examples in our domain that illustrate each of these areas.

Class and object descriptions: As in any application, we need a domain ontology in which to work. Our home theater application contains a knowledge base including a concept taxonomy and instance descriptions. The knowledge base was created by working with an expert in the domain. The database of instance information was also hand–compiled for this small application but in other applications where we work with changing instance information, we have written automatic translation routines that periodically access databases and then update our knowledge base[2]. The terminological knowledge base contains definitional information concerning classes as well as rules. We worked directly with an expert to obtain these rules, but in our larger applications of this sort[5], system builders begin with preexisting rule specifications and use a rule translator to generate CLASSIC rules. Rules in this application fall into two classes: both hard and fast electrical rules (for example, a receiver must have an A/B switch in order to support secondary main speakers), and "rules of thumb" (for example, home theater systems do not have more than one TV or two VCRs). All products configured by the knowledge base must abide by the hard and fast electrical rules, and products configured following our "guidance" also follow the rules of thumb. If CLASSIC supported defaults, all the rules associated with "guided-stereo-system" would have been defaults.

Active inference: The home theater application uses CLASSIC to provide active inference after the interface has guided the user through a few simple questions. We assume that people want to build audio only, home theater only, or combination audio/video systems and that they already have a price range in mind. The user interface then presents menu choices for the different system types and quality (or price) classes. Once those two choices are made, the application uses CLASSIC to ask follow up questions as appropriate and to produce a complete (abstract) description of a consistent product. For example, if the user chooses a high–quality combination system, then CLASSIC deduces that the target system must have an amplifier, preamplifier, tuner, main, center, and surround speakers, a subwoofer, VCR, and TV, and it presents this information graphically. CLASSIC calculates the deductive closure of the information provided, which usually implies properties of the system as a whole as well as properties of all the individual components. The user can view the completed information on any component just by clicking on the icon. For ex-

[1] DACS IV–2000 is a digital cross–connect system that processes digitized signals for some US standard transmission rates.

[2] CLASSIC is freely available for academic purposes, and commercially available for other purposes. It has been distributed to over 80 universities and is in use in many internal projects within AT&T.

ample, if she clicks on the TV, she sees, among other things, that the list price must be at least $1000.

Explanation: CLASSIC can justify all of its beliefs. (For a more complete description of our explanation design, see [3]). In the example above, if the user asks how the TV acquired its price restriction, she learns that a rule fired which says that high–quality systems must have high–quality components, which for TVs enforces a minimum price of $1000. The explanation facility can also answer other questions such as why one object does or does not "subsume" (is or is not more general than) another object, why a rule fired on an object, or why an error occurred.

Incomplete and evolving knowledge bases: CLASSIC allows refinement of (and changes to) a system specification. For example, a user could add new components (e.g., a turntable), chosen from a panel of icons. She might also "instantiate" a component description by choosing a particular make and model. The interface will only generate choices that appear to be consistent with the information that CLASSIC has derived about the component (by using the specification of the component as a query to the database of individuals). The user may also delete a requirement on the system, in which case any deductions that were made as a result of this requirement are removed from the specification. When the user has finished refining the system to her satisfaction, she can ask the application to complete the specification for her. The application will then choose consistent makes and models for all the components she has left unspecified. She can then view a parts list, after which she might want to ship the order off to the factory.

If the user is not familiar with different types of stereo equipment, she may wish to trust our expert, and build a system starting with one of the example systems, where all the components are known to work well together. She can then refine this system according to her needs, requesting alternative makes and models to the ones chosen, and adding and removing components.

Errors: Although CLASSIC and the application minimize the places where a user can make an error, errors can still occur since the application does not ask CLASSIC to precalculate all possible consequences of a given choice. The user could make a choice which would cause a rule to fire, which would then cause a propagation of some inconsistent information. For example, suppose the user wants to build a system, starting with a few components she already owns, including a small TV. She may later add a price range for the whole system, and based on this information, CLASSIC classifies her system as a high-quality system. All the high–quality system rules then fire, including one which requires the system's TV to have at least a 27–inch diagonal. This information gets propagated onto the user's small TV, which causes an error. CLASSIC does not allow the knowledge base to be in an inconsistent state, so it will roll back the knowledge base to the previous consistent state, meanwhile saving copies of all the individuals that led to the error, in their inconsistent states. If the user asks CLASSIC for an explanation of the error, CLASSIC can access the inconsistent state information to generate an explanation.

We feel that description logic technology is particularly well matched to this style of configuration problem for the following reasons: First, the application is fairly logical (not heuristic) so we would either have to implement the logic in a programming language or start with a tool like CLASSIC that incorporates a formal logic. Second, this domain is naturally hierarchical and rule information is appropriate at many different levels of the taxonomy. DLSs support hierarchical rules instead of using a more traditional, flat rule-based approach. This may simplify knowledge engineering and maintenance[4]. CLASSIC rules can be simpler than typical expert system rules because they only need to contain content appropriate to a certain level of concept in the hierarchy, and they do not need to contain any control information. Finally, the application naturally incorporates many different types of inference; a few of which include: inheritance, propagation, bounds constraints, and rules. These can be encoded directly in DLSs instead of needing to be paraphrased into rules. Possibly more importantly, explanations of the reasoning process may be in terms of the naturally occurring inferences.

This home theater system is a simple example of a family of applications where a description logic-based platform is used to implement standard configuration tasks and provide the basis for additional functionalities. The deployed applications built on this design have provided many advantages including decreased order processing intervals (facilitating hypothetical configuration evaluations, which were previously infeasible), reductions in personnel required to maintain product information, accurate and up–to–date pricing for sales quotes, elimination of duplication in databases, and identification of incompatible knowledge.

Acknowledgements

We are indebted to Charles Isbell for his collaboration on the implementation of our demo. We wish to thank the entire CLASSIC group, particularly Peter Patel-Schneider and Ron Brachman, for their insightful comments on this work.

References

[1] 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, pp. 401–456, 1991.

[2] R. J. Brachman, P. G. Selfridge, L. G. Terveen, B. Altman, A. Borgida, F. Halper, T. Kirk, A. Lazar, D. L. McGuinness, and L. A. Resnick. Integrated Support for Data Archaeology. in *International Journal of Intelligent and Cooperative Information Systems*, 2(2), 1993, pp. 159–185.

[3] D. L. McGuinness and A. Borgida. Explaining Subsumption in Description Logics. In *Proc. IJCAI*, Montreal, August 1995.

[4] J. R. Wright, D. L. McGuinness, C. Foster, and G. T. Vesonder. Conceptual Modeling using Knowledge Representation: Configurator Applications. In *Proc. Artificial Intelligence in Distributed Information Networks, IJCAI-95*, Montreal, 1995.

[5] J. R. Wright, E. S. Weixelbaum, K. Brown, G. T. Vesonder, S. R. Palmer, J. I. Berman, and H. H. Moore. A knowledge-based configurator that supports sales, engineering, and manufacturing at AT&T Network Systems. In *Proceedings of the Innovative Applications of Artificial Intelligence Conference*, pp.183–193, 1993.

FLEX-Based Disambiguation in VERBMOBIL

J. Joachim Quantz[1], Guido Dunker[1], Manfred Gehrke[2], Uwe Küssner[1], Birte Schmitz[1]
[1]TU Berlin, {jjq,dunker,uk,birte}@cs.tu-berlin.de
[2] Siemens AG München, Manfred.Gehrke@zfe.siemens.de

1 Introduction

The VERBMOBIL project is funded by the German Federal Ministry of Education, Science, Research and Technology (BMBF) and is concerned with *automatic interpreting of face-to-face dialogues*. In the first phase of the project (1993–1996), in which 31 academic and industrial partners are involved, a demonstrator and a prototype are built. In this paper we describe the different tasks for which the DL system FLEX is used in the VERBMOBIL demonstrator, namely *domain modeling*, *ordering of syntactic parses*, *conceptual disambiguation*, and *determination of dialogue acts*. Before describing these tasks, which are presented in more detail in [5; 15; 16; 24; 25], we briefly sketch the characteristics of the DL system FLEX.

2 The FLEX System

The FLEX system is an extension of the BACK system [7] specifically designed for the use in VERBMOBIL.[1] Its main characteristics are:

1. an expressive term language
2. situated object desriptions
3. weighted defaults
4. flexible inference strategies

A detailed description of the FLEX system is currently in preparation [15], here we contend ourselves with listing the constructs provided by the system and discuss their relevance for the VERBMOBIL application.

c	→	c1 and c2, c1 or c2, not(c) all(r,c), atleast(N,r,c), atmost(N,r,c) r:o, f:c, oneof([o1,...,oi]), rvm_equal(r1,r2)
r	→	r1 and r2, r1 or r2, not(r) domain(c), range(c), r1 comp r2, inv(r)
γ	→	t1 := t2, t1 :< t2, c1 => c2, c1 $\sim N \sim>$ c2, o :: c in s

In the current VERBMOBIL application disjunction is only rarely used for concepts and not at all for roles; negation is only used as primitive negation for both concepts and roles; 'atleast' and 'atmost' are only used for cardinality 1; and role value maps are only used for features.

Term-valued features (f:c) allow a modeling similar to typed feature structures [4]. Their behavior can be characterized by the following subsumption relations:

	→	f:c1 and f:c2 = f:(c1 and c2)
	→	f:cbot = cbot
c1 < c2	→	f:c1 < f:c2

Term-valued features are used to model the conceptual content of NL expressions, as exemplified below in Section 4.2.

The inference rules implemented in FLEX follow the normalize-compare paradigm, i.e. concepts and roles are transformed into normal forms by applying normalization rules. To test subsumption, these normal forms are then structurally compared using subsumption rules. The approach of deriving such inference rules by rewriting proofs in the Sequent Calculus is described in detail in [20; 21; 22]. In principle, each inference rule is implemented twice, once as a normalization rule and once as a subsumptions rule. The flexible inference strategy is then achieved by providing for each inference rule the choice whether it is to be applied during normalization or subsumption cheking, or whether it is to be switched off completely.[2]

The version of the FLEX system used in the VERBMOBIL demonstrator contains those inference rules needed for the application. We are currently integrating the missing inference rules into the system. In general, the architecture chosen for the implementation of the inference rules proved to be rather adequate for such an incremental implementation.

3 Domain Modeling

Domain models are used in Natural Language Processing systems to provide domain-specific background knowledge. This is usually achieved by defining the concepts

[1]FLEX is to a large extent compatible with BACK V5. It does not support revision, however, and provides a different programming interface, in particular with respect to queries.

[2]Choosing a particular inference strategy is a task to be performed by the system developer for a particular application, and *not* by a naive user of the FLEX system.

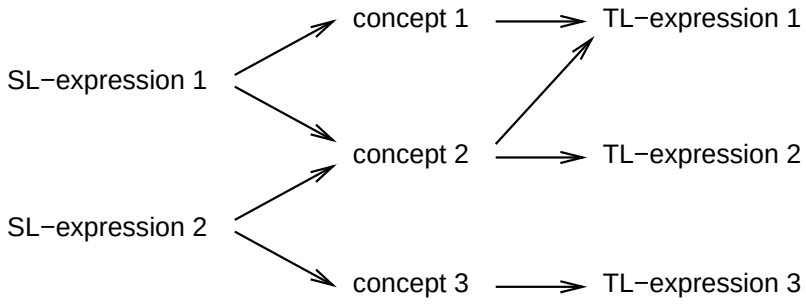


Figure 1: NL expressions and concepts are related by m-to-n mappings.

occurring in a particular application, i.e. by modeling their relevant properties and the hierarchical relations between them. Obviously, the main problem consists in deciding which aspects are relevant for a particular application.

We illustrate this problem by considering the task of assigning concepts to the lexemes of a language. The most fundamental decision concern the *number* of concepts assigned to a lexeme. Consider an example taken from the literature, namely the meaning of the lexeme 'open' discussed in [23, p. 145]:

(3.0) a. Tom opened the door.
b. Sally opened her eyes.
c. The carpenters opened the wall.
d. Sam opened his book on page 37.
e. The surgeon opened the wound.

Searle claims that

> ... the word "open" has the same literal meaning in all five of these occurrences. Anyone who denied this would soon be forced to hold the view that the word "open" is indefinitely or perhaps even infinitely ambiguous since we can continue the example; and indefinite ambiguity seems an absurd result. [23, p. 146]

He gives additional examples in which one might argue that the lexeme 'open' has a different meaning:

(3.0) a. The chairman opened the meeting.
b. The artillery opened fire.
c. Bill opened a restaurant.

These examples indicate that the assignment of concepts to lexemes is influenced both by linguistic and by encyclopedic knowledge.

In the context of Machine Translation, a straightforward criterion for deciding how many conceptual contents to assign to a source-language expression is to provide a separate concept for each possible target-language translation. Consider the examples (1) and (2)—the German translation for 'open' in (1) is 'öffnen', whereas in (2) it is 'eröffnen'. This would be a reason to distinguish the conceptual content of 'open' expressed in (1) and (2). In general, we would thus have m-to-n mappings between NL expressions and concepts, as illustrated in Figure 1.[3]

Note that the structure of the conceptual hierarchy to be modeled thus depends on the particular source and target languages. Based on *contrastive analyses* it can be decided whether a concept has to be included into the domain model or not. Note further, that it is in principle possible to extend such a bilingually motivated hierarchy by adding new source and target languages, i.e. by adding mappings and integrating the additionally required conceptual distinctions [6; 8].

Another requirement to distinguish readings of a lexeme arises from the need to choose the appropriate concept given an NL expression in a particular context (see Section 4.2). Given the task of Machine Translation in general and of conceptual disambiguation in particular, there are thus two main criteria for deciding which conceptual distinctions to include in the domain model:

1. distinctions corresponding to potential translations of an expression;
2. distinctions relevant for selectional restrictions used for disambiguation.

The current version of the domain model in the VERBMOBIL demonstrator contains approximately 300 concepts and 170 roles. In the following we illustrate its use by considering three different examples of FLEX-based disambiguation.

4 Disambiguation

4.1 Ordering Syntactic Parses

In the VERBMOBIL demonstrator the syntactic parser receives as input a word lattice and produces as output syntactic structures of the various paths in the lattice, which are syntactically legal. In general, each of the syntactically legal pathes in the word lattice yields one or more syntactic analyses.[4] In the demonstrator the

[3] We thus assume that the so-called translational mismatches [10] are not the exception but rather the average case.

[4] For the VERBMOBIL corpus the average number of parses per sentence/path is 4.4.

syntactic analysis[5] is augmented by a test on selectional restrictions. Wrt this tests the alternative parses are ordered, and the best analysis is then passed to semantic construction, which builts DRT-like semantic representations for the syntactic parse.

The ordering of alternative parses is performed on the basis of sort information represented in the domain model. Basically, two types of checks are performed:

- A subsumption test between the sort of an argument position and the sort of the possible argument, where the first one has to subsume the second one. The same holds between the sort of a phrase and the sort of an adjunct.
- Checking whether subsumption or a specific relation (a FLEX-role) holds between the sorts of two arguments, in order to take into account some global constraints between arguments and/or adjuncts. This is especially useful for semantically empty verbs such as 'to be' or 'to have'. E.g. a distinctive feature for 'to be' in the definitorial reading is that the sort of the subject has to be subsumed by the sort of predicate noun.

In order to test the selectional restrictions most lexical entries are augmented by a "sort"-feature, whose value is associated with a concept of the domain model[6]. The grammar has to provide for the percolation of the sortal information and to test it at appropriate grammar rules. Though these tests lead to wrong results in cases of type coercion or metonomies they are nevertheless useful to recognize such cases as well as acoustic recognition errors.

Consider the following examples where the first one is a recognizer mismatch, while the second is a case of type coercion (violations of sortal restrictions are emphasized):

(4.0) a. Dann muß ich *drei Uhr* auf den Freitag verschieben.
b. I have to postpone *three o'clock* to friday.

(4.0) a. Dann muß ich *Dreyer* auf den Freitag verschieben.
b. I have to postpone *Dreyer* to friday.
c. (lit.: I have to postpone the meeting with Dreyer to friday.)

If the process of semantic construction should not be burdened with the task to select the semantically sensible readings out of the parsing results, a solution to this dilemma, i.e. discarding meaningless utterances, but still have a chance to interpret cases of type coercion, is to apply the test of the selectional conditions as a soft constraint, which gives each selectional clash a penalty instead of immediatelly rejecting it. Thus the application of selectional restrictions results in an ordering, where the "sortally" best readings are presented first to semantic construction process.[7]

These selectional conditions are thus modeled as *soft constraints* and not as strict filters. Such a modeling of selectional restrictions as *soft constraints* instead of as strict filters allows preferential disambiguation without making sentences containing *type coercion* ungrammatical and it enhances the robustness of the system. Technically, this is realized by a bonus system, where each successfull application of a condition increments the bonus counter. The bonus system is additionally refined by giving preferred constructions an extra bonus. Normalizing the total bonus figure with the number of all applications of selectional conditions of the utterance gives a measure for the selectional quality of an analysis. Thus we get the following "soft" quality criteria:

$$\frac{N(successes)}{N(tests)} = \begin{cases} \geq 1 & \text{literal meaning} \\ < 1 & \text{non-literal meaning or} \\ & \text{acoustic recognition error} \end{cases}$$

Given the Verbmobil test corpus with approximately 200 turns, i.e. about 400 sentences, as evaluation basis, the test of selectional restrictions increases the total parsing time by 5 %. For a part of this corpus (76 utterances) that syntactic reading of each utterance has been determined which is the intended one in the respective dialogue. An application of the parser without testing selectional restrictions on this smaller corpus with written input gives the intended reading as the first reading in 49 (64 %) of all cases, while with selectional restrictions the first reading is the intended one in 71 (95 %) cases. Most of the remaining cases, where the correct reading is not the first one, are due to either PP-attachment (adjunct or modifier) or the lack of prosodic information, where the boundary between two sentences of an utterance is misplaced in the first reading. The evaluation on word lattices gives similar results. But one has to consider, that sortally deviant utterances are not discarded, they just have a worse scoring (< 1). When there is just one result later stages of evaluation have to decide whether to do some repair due to acoustic mismatches or to interprete type coercion.

4.2 Conceptual Disambiguation

One of the central tasks of Machine Translation is to choose the appropriate concept given an NL expression in a particular context. Usually this involves complex background knowledge and the major problem for conceptual disambiguation is

1. to make background knowledge available, i.e. to store it in the computer;

[5] Cf. [2].

[6] Wrt the m-to-n-mapping between lexemes and concepts exemplified in section 3 the sort of a lexeme is often enough a rather general concept, located near the top of the concept hierarchy. In the course of conceptual disambiguation it will be refined to a more specific concept depending on the context of actual usage as e.g. described in [1; 12; 13].

[7] Note that this ordering thus follows the strategy of *preferential disambiguation* as described in [14].

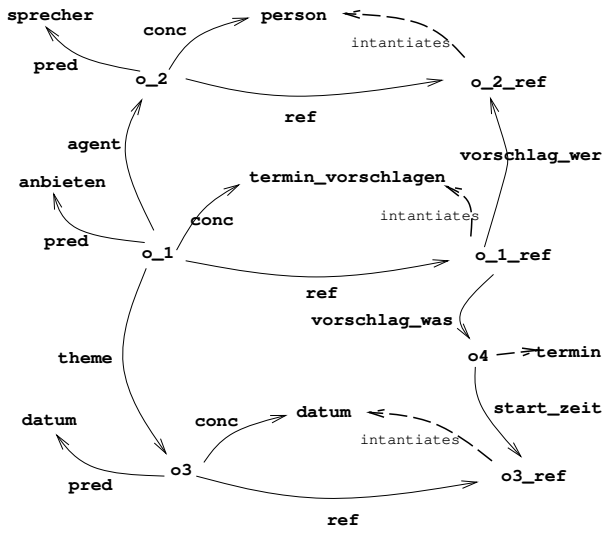


Figure 2: Sample representation of linguistic and conceptual structure.

2. to provide fast access to background knowledge, i.e. to provide inference mechanisms drawing the *relevant* inferences.

In the absence of a satisfying solution to this task which comprises our entire common-sense knowledge one has to devise heuristics based on partial information and incomplete reasoning. The most popular way of doing conceptual disambiguation is by means of *selectional restrictions*. Roughly speaking, conceptual disambiguation is performed in this approach by considering the arguments of an expression, or vice versa the functor taking an expression as argument. Note that this is a rather limited way of taking context into account, but it is exactly this limitation which makes it computationally feasible.

In the demonstrator, conceptual disambiguation is achieved by partly translating the Discourse Representation Structure (DRS) [11] built by semantic construction [3] into FLEX. In the demonstrator we use a 3-level representation: for each utterance we introduce a FLEX object; this object is related via the role 'cond' to objects representing the semantic conditions in the DRS; these conditions are in turn related via the role 'ref' to the discourse referents. The conditions themselves are related by thematic roles, such as 'agent', 'theme', etc.

Figure 2 shows a simplified representation of this 3-level representation for the sentence:

(4.0) a. Ich kann Ihnen 17 Uhr anbieten.
b. (lit.) I can offer you 5 pm.

The mapping of the semantic predicates used in the DRS's into concepts is achieved by rules like:

pred:anbieten and the(theme comp ref,termin_c)
=> conc:termin_vorschlagen_c
pred:vergessen and the(theme comp ref,zeit_c)
=> conc:nicht_passen_c
pred:gut and the(ref,passen_c)
=> conc:gut_passen_c
pred:unguenstig and the(ref,zeit_c)
=> conc:schlechter_termin_c

Note that 'conc' is modeled as a concept-valued feature and that the fillers of the role 'ref' are constrained to be instances of the concept filling the 'conc' feature.

4.3 Determining Dialogue Acts

Given the restricted scenario of the VERBMOBIL demonstrator, namely appointment scheduling, approximately 20 domain-specific dialogue acts (e.g. 'suggest', 'request_comment', 'accept', 'reject') have been defined.[8]

The automatic determination of the dialogue act performed by utterances is based on preference rules encoding conventions pertaining on various levels (see [14; 18] for details on *preferential disambiguation*).

Technically, these preference rules are represented as *weighted defaults* [19]: if there is a piece of information X in the representation of the utterance, then there is a preference of weight w for the utterance to be of type Y. The information represented on the left-hand side of a default concerns different types of knowledge, namely:

- syntactic information (e.g. sentence type, voice of the verb),
- keywords (certain discourse markers like German 'leider' or 'schon'),
- semantic information (the conceptual content of expressions, the conceptual type of referents),
- macro-structural information (e.g., the dialogue act of the previous utterance).

[8] Recently, a new terminology wrt dialogue acts in VERBMOBIL has been agreed upon [9]. Though our implementation used the dialogue acts defined in [24], we use the more recent terms in the following.

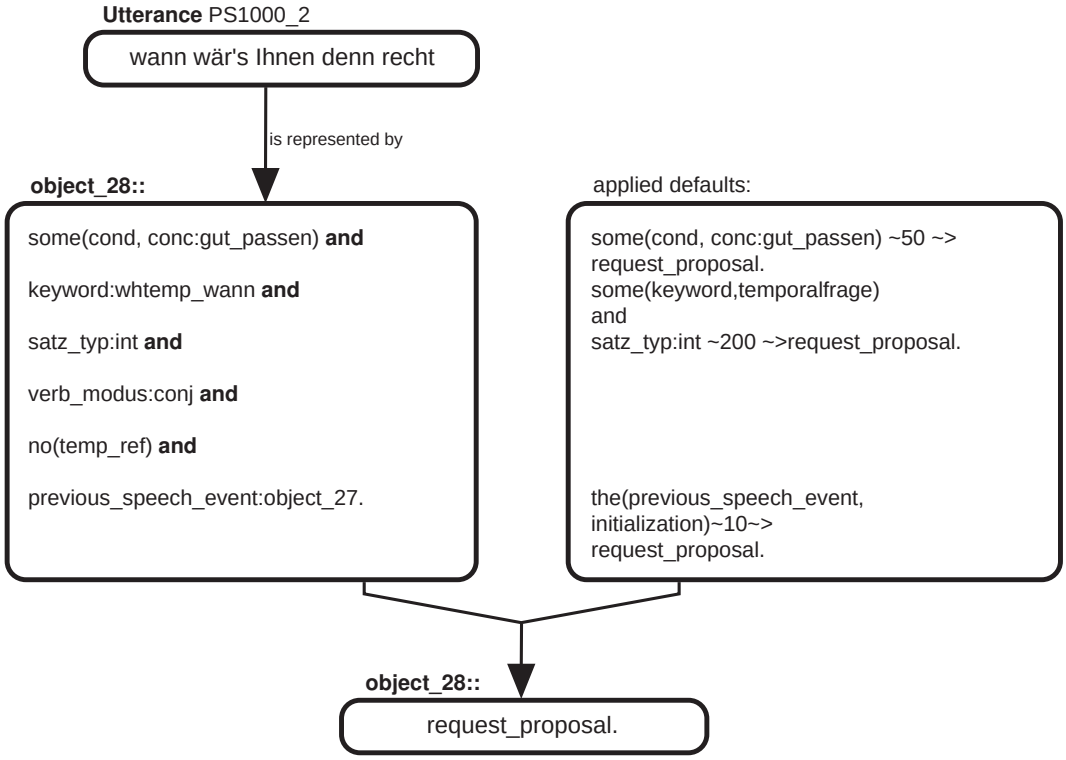


Figure 3: Determining the dialogue act for utterance (6).

dialogue act	*sum*	*recognized*		*in the set*	*recognition failed*
reject	30	29	97 %	-	1
accept	23	16	70 %	1	6
request_comment	7	6	86 %	-	1
request_suggest	17	16	94 %	-	1
give_reason	15	12	80 %	-	3
confirm	19	13	68 %	-	6
init	10	10	100 %	-	-
clarify_query	5	1	20 %	-	4
clarify_answer	10	5	50 %	4	1
comment	5	4	80 %	-	1
suggest	80	73	91 %	7	-
sum	221	185	84 %	12	24

Figure 4: First evaluation of the accuracy of dialogue act assignment.

In this way each utterance is partially represented by a DL concept, i.e. roughly speaking by a list of feature-value pairs. Part of these feature values are provided by the syntactic component, namely those concerning keywords, the sentence type and the voice of the verb, others are semantic information. Figure 3 illustrates how the utterance

(4.0) a. Wann wäre es Ihnen denn recht?
b. (lit.) When would it suit you?

is represented in the FLEX system and recognized as an instance of 'request_proposal' by applying weighted defaults.

The current implementation used in the VERBMOBIL demonstrator contains 83 rules, of which 64 are defaults and 19 strict. 22 of the defaults rely on keyword information alone, 9 are exclusively based on syntactic information, whereas 27 use only semantic information. The remaining defaults draw on a combination of these types of information.

Since input from semantic construction was not yet available we evaluated the system by using 15 annotated

dialogues as input (see Figure 4). On the basis of 221 annotated utterances we obtained an accuracy of 84 %—a rather encouraging result for this first implementation, which is based only on micro-structural information. We hope to achieve even better results when taking into account also macro-structural information.

5 Conclusion

The application of the FLEX system in the VERBMOBIL demonstrator has in general been successful. Given the expressivity of FLEX, most information could be represented in a declarative manner. In addition to these declarative representations, small Prolog programs had to be written, realizing the interface to the other components in VERBMOBIL. These programs contained tells and queries of the FLEX Prolog interface.

Wrt the weighted defaults used for the determination of dialogue acts two results seem particularly important. As had to be expected, choosing the right weights is a non-trivial task. The choice of the weights has impact not only on the resulting dialogue act but also on the time needed to compute it. For the demonstrator we achieved an efficient performance on the basis of a trial-and-error approach. It is obvious, however, that additional tools are needed to support the weighting of defaults.

Finally, the architecture of FLEX made the incremental integration of inference rules straightforward. We added normalization/subsumption rules whenever they were needed in our application. In this respect, the idea of flexible inference strategies underlying FLEX proved to be rather adequate.

Acknowledgements

This work was funded/partially funded by the German Federal Ministry of Education, Science, Research and Technology (BMBF) in the framework of the Verbmobil Project under Grants 01 IV 101 Q 8 and 01 IV 102 A 0. The responsibility for the contents of this study lies with the authors.

References

[1] M. Bierwisch, "Semantische und konzeptuelle Repräsentation lexikalischer Einheiten", *Studia grammatica* **XII**, 61–99

[2] H.U. Block, S. Schachtl,"Trace & Unification Grammar", *COLING-92*, 87 – 93

[3] J. Bos, E. Mastenbroek, S. McGlashan, S. Millies, M. Pinkal, "A Compositional DRS-based Formalism for NLP Applications", *Proceedings of the International Workshop on Computational Semantics*, Tilburg, 1994

[4] B. Carpenter, *The Logic of Typed Feature Structures*, Cambridge: Cambridge University Press, 1992

[5] M. Gehrke, *Sorting Syntactic Analysis by Selectional Restrictions*, Verbmobil Report in preparation, Siemens AG, 1995

[6] R. Henschel, J. Bateman, "The Merged Upper Model: A Linguistic Ontology for German and English", *COLING-94*, 803–809

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

[8] E. Hovy, S. Nirenburg, "Approximating an Interlingua in a Principled Way", in *Proc. of the DARPA Speech and Natural Language Workshop*, Hawthorne, NY, Feb. 1992

[9] S. Jekat, A. Klein, E. Maier, I. Maleck, M. Mast, J.J. Quantz, *Dialogue Acts in VERBMOBIL*, Verbmobil Report 65, Universität Hamburg, 1995

[10] M. Kameyama, R. Ochitani, S. Peters, "Resolving Translation Mismatches With Information Flow", *ACL-91*, 193–200

[11] H. Kamp, U. Reyle, *From Discourse to Logic*, Dordrecht: Kluwer, 1993

[12] J. Pustejovsky, "Current Issues in Computational Lexical Semantics", *EACL-89*, xvii–xxv

[13] J. Pustejovsky, "Type Coercion and Lexical Selection", in J. Pustejovsky (Ed.), *Semantics and the Lexicon*, Dordrecht: Kluwer, 1993, 73–94

[14] J.J. Quantz, *Preferential Disambiguation in Natural Language Processing*, PhD Thesis, Technische Universität Berlin, 1995

[15] J.J. Quantz, G. Dunker, F. Bergmann, I. Kellner, *The FLEX System*, KIT Report in preparation, Technische Universität Berlin, 1995

[16] J.J. Quantz, M. Gehrke, U. Küssner, "Domain Modeling for Machine Translation", *to appear in TMI-95*

[17] J.J. Quantz, M. Gehrke, U. Küssner, B. Schmitz, *The VERBMOBIL Domain Model*, KIT Report 122, Technische Universität Berlin, 1994

[18] J.J. Quantz, B. Schmitz, "Knowledge-Based Disambiguation for Machine Translation", *Minds and Machines* **4**, 39–57, 1994

[19] J.J. Quantz, S. Suska, "Weighted Defaults in Description Logics–Formal Properties and Proof Theory", in B. Nebel, L. Dreschler-Fischer (eds), *KI-94: Advances in Artificial Intelligence*, Berlin: Springer, 1994, 178–189

[20] 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, 1992, 84–105

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

[22] V. Royer, J.J. Quantz, "On Intuitionistic Query Answering in Description Bases", in A. Bundy (Ed.), *CADE-94*, Berlin: Springer, 1994, 326–340

[23] J.R. Searle, *Intentionality*, Cambridge: Cambridge University Press, 1983

[24] B. Schmitz, S. Jekat-Rommel, "Eine zyklische Approximation an Sprechhandlungstypen—zur Annotierung von Äußerungen in Dialogen", *Verbmobil Report 28*, 1994

[25] B. Schmitz, J.J. Quantz, "Dialogue Acts in Automatic Dialogue Interpreting", *to appear in TMI-95*

A Concept Language for an engineering application with part–whole relations

Ulrike Sattler
RWTH – Aachen, uli@cantor.informatik.rwth-aachen.de

Introduction

Terminological Knowledge Representation (TKR) Systems are powerful means to represent unambiguous knowledge – like knowledge in technical domains. We investigate how TKR Systems can be used in process modeling, a field dealing with modeling huge chemical plants. As these plants are very complex, support of top–down modeling is a quite ambitious, but useful task for TKR Systems. An interesting problem to solve in this context is the handling of composite objects: An appropriate TKR System for this application should be able to

- handle different part–whole relations, for example Component–Composite or Stuff–Object (most of which are transitive),
- represent inverses of these part–whole relations,
- model transitivity–like interactions between part–whole relations since some chains of different part–whole relations imply further part–whole relations. These implicit relations permit reasoning about objects having parts, which have parts, which. . . which have certain properties.
- represent special characteristics of composite objects, for example parts belonging exclusively to a single whole.

Hence the application calls for a concept language with powerful role forming operators.

In this paper, results of an investigation of part–whole relations and their relevance for a process–modeling application are given and a concept–language $\mathcal{P}$ with appropriate expressive power is defined. Satisfiability of concept terms in $\mathcal{P}$ is undecidable, hence it is necessary to drop some (but not many) of the demands made for the benefit of decidability. Several ways to handle the high complexity of inference algorithms of $\mathcal{P}$ are discussed.

The application

Process modeling plays an important role in process engineering, for planning as well as for optimization and controlling. To model chemical plants, they are decomposed into components like distillation columns, tubular reactors, valves, mixed phases, signal transformers, etc. These components are again decomposed into components which are again decomposed, etc., until components are obtained whose physico–chemical characteristics can be described via differential, algebraic or integro–algebraic equations. Hence support of modeling with varying granularity and standard building blocks is indispensable. Furthermore, there is a large number of standard building blocks, which have to be specialized or modified for each concrete model. It is very difficult to define these numerous blocks such that the implicit taxonomy is the same as the explicitly stated and intended one, and a system able to infer implicit subsumption relations could help the model builder to verify his/her definitions. A TKR System able to handle part–whole relations could give this support.

Demands made by the application

Classifications of part–whole relations can be found in [Winston *et al.*,1987; Gerstl and Pribbenow,1993] and a fusion of these classifications seems to be adequate for the given application. Integration of part–whole relations into TKR Systems is treated in [Padgham and Lambrix,1994; Artale *et al.*,1994; Franconi *et al.*,1994], but the application asks for an integration with more expressive power since reasoning in this application needs for example transitive part–whole relations and consequences of transitivity–like interactions between these relations.

A widely held opinion is that *the* part–whole relation is not transitive in general. However, the non–transitive counter–examples come from mixing up different types of part–whole relations. For example: This arm is part of Herbert. Herbert is part of this orchestra. To conclude "This arm is part of the orchestra." does not make much sense – at least it sounds odd. In this case, Component–Object and Member–Collection relations have been mixed.

The next table shows all types of part–whole relations relevant for the given technical application, how they interact and, in the diagonal, whether they are transitive or not. Examples are given to illustrate these relations and their domains, exact definitions cannot be given here. Please note that the three relations "Quantity-Mass", "Stuff-Object" and "Ingredient-Object" are nec-

Relation, Example	aCC$b \wedge$ $bXc \Rightarrow$	aMCb $bXc \Rightarrow$	aSEb $bXc \Rightarrow$	aSOb $bXc \Rightarrow$	aIOb $bXc \Rightarrow$	Remarks
Component – **C**omposite `motor-car`	aCCc	aCCc	aCCc	aSOc	$\exists d : a\text{QM}d$ $\wedge\ d\text{SO}c$	Component inherent part-whole boundaries, whole is inhomogeneous
Member – **C**ollection `grain-2 g salt`	Nothing[1]	Nothing	Nothing	aSOc	$\exists d : a\text{QM}d$ $\wedge\ d\text{SO}c$	parts defined through whole, parts not locally fixed
Segment – **E**ntity `front car-car`	Impos.[2]	aMCc	aSEc	aSOc	$\exists d : a\text{QM}d$ $\wedge\ d\text{SO}c$	Arbitrary part-whole boundaries
Quantity – **M**ass `2 kg steel-steel`	Impos.	aMCc	aQMc	aSOc	$\exists d : a\text{QM}d$ $\wedge\ d\text{SO}c$	Whole has no boundaries, no volume; part is bounded
Stuff – **O**bject `steel-bike`	Impos.	Nothing	Impos.	aSOc	Impos.	Part is a unbounded mass
Ingredient – **O**bject `2 kg steel-bike`	Impos.	aSOc	$\exists d : a\text{QM}d$ $\wedge\ d\text{SO}c$	aSOc	$\exists d : a\text{QM}d$ $\wedge\ d\text{SO}c$	Part and whole are spatially inseparable

Figure 1: A survey of different part–whole relations and their transitivity–like interactions

essary to reason about objects made of some ingredients if we can not calculate: If we know that this bike has a frame and that its frame is made of 6 kg aluminium — what is to infer for the whole bike beside the fact that it is made of aluminium?

The five middle columns list pseudo–transitive consequences. Relations are abbreviated by their initials, e.g. aCCb stands for an element a being part of b with respect to the relation Component–Composite. The variable X refers to the relation in the respective row.

A column showing consequences of $a\text{QM}b \wedge bXc$ (QM is short for the relation "Quantity–Mass") for part–whole relation X is missing since there are no such consequences (note that the Quantity–Mass relation is not transitive).

Since a relation X is transitive iff it is the transitive closure of some relation $Y \subseteq X$, we introduce six primitive roles `is_d_component, is_d_member`, etc., to model these direct, minimal relations Y and define six roles representing the transitive closure of the respective direct role (if it is transitive) and the above mentioned transitivity–like connections between them:

(defrole is_component (is_d_component $\sqcup$
(is_member $\circ$ is_d_component))$^+$),
(defrole is_member
(is_d_member $\sqcup$ (is_d_member $\circ$ is_segment)
$\sqcup$ (is_d_member $\circ$ is_quantity))),
(defrole is_segment (is_d_segment)$^+$),
(defrole is_quantity
(is_d_quantity)$\sqcup$ (is_segment $\circ$ is_d_quantity)),
(defrole is_ingredient (is_d_ingredient)),
(defrole is_stuff (is_d_stuff $\sqcup$
(is_d_stuff $\circ$ (is_component $\sqcup$ is_member $\sqcup$
is_segment $\sqcup$ is_ingredient $\sqcup$ is_quantity))
$\sqcup$ (is_member $\circ$ is_ingredient)
$\sqcup$ ((is_quantity)$^{-1}$ $\circ$ is_segment $\circ$ is_ingredient)
$\sqcup$ ((is_quantity)$^{-1}$ $\circ$ is_ingredient
$\circ$ (is_ingredient $\sqcup$ is_component $\sqcup$ is_member
$\sqcup$ is_segment $\sqcup$ is_quantity)))$^+$),

Syntax and semantics defined in the usual manner, for example see [Baader and Hollunder,1991]. If we want to allow only for models where the direct part–whole roles are interpreted as pairwise disjoint relations, we have to define a superconcept `Disj` of all other concepts by:

`(defconcept Disj (`$\forall$ `Y`*`.`
`(`$\forall$ `((is_d_component` $\sqcap$ `is_d_member)` $\sqcup$
`(is_d_component` $\sqcap$ `is_d_segment)` $\sqcup \cdots$`).`$\bot$`)))`

where `Y` has to be replaced by a disjunction of all primitive roles occuring in the TBox and their respective inverses (see for example [Schild,1991] for an explanation of this universal role). By defining each concept as a subconcept of `A`, we prevent parts from being simultaneously related to one whole via two different direct part–whole relations. Hence direct part–whole roles can only be interpreted as pairwise disjoint relations.

Beside these relations, the application asks for means to express special characteristics on part–whole relations, as for example parts belonging exclusively to a whole.

Exclusive parts: A part is an exclusive part of a whole, if it has at most one role–filler with respect to one of the direct is–part–of relations. One way to express this in a terminological system is to use number re-

[1] This means that a, b, c with these relations are possible, but nothing can be concluded.

[2] Where "Impossible" means that there cannot be such b.

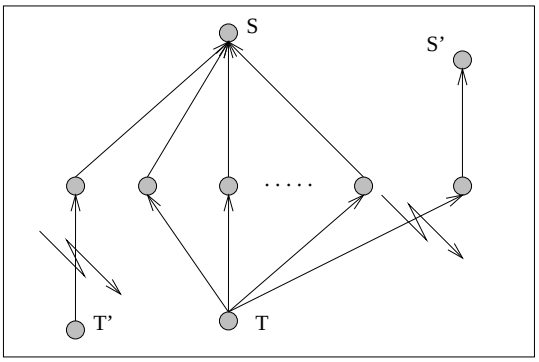
strictions on roles, for example to express that a motor is part of at most one whole:

`(defconcept motor (≤ 1 is_d_component: car) ⊓ ... ).`

Multi–Possessed parts: In our technical applications, we have to model composite devices whose parts use a common part. In the next figure for example, a device `S` contains some reactors that all use the same tank `T`. It is not possible to view the tank as part of `S`, since the reactors have a tank as a necessary direct part. One way to define those multiple possessions is the following.

`(defconcept S ((= 1 ((is_d_component)`$^{-1}$`)`^{2}`: T)⊓ ...))`

Owner–Restricted parts: These objects are parts of a set of wholes, which are characterized as being themselves parts of one single whole.



For example, we do not want any other reactor beside the ones contained in the device `S` to use the same tank `T`. This restriction of wholes to contain parts can be expressed through a restriction on parts to be contained by wholes:

`(defconcept T ((= 1 (is_d_component)`^{2}`:S) ⊓ (∀is_d_component.(∃is_d_component.S))⊓ ...))`

Essential parts: The existence of an essential part is essential for the existence of a whole. This can easily be expressed using Exists Restriction

`(defconcept human (∃ (is_d_component)`$^{-1}$`.brain))`

Dependent parts: For a dependent part the existence of its whole is essential. The definition of dependent parts can easily be done using exists restrictions:

`(defconcept ceiling (∃ is_d_component.room))`

Finally, part–whole relations are acyclic. This demand can be realized by introducing a second superconcept `Acyc` of each concept which is defined as:

`(defconcept Acyc (∀ Y`*`.(∀ (self ⊓ (is_component ⊔ is_segment ⊔ is_d_ingredient ⊔ ...)).⊥)))`

where `Y` is the previously described universal role and `self` $= id(\top)$.

Consequences of these demands

Summarizing, to represent these part–whole relations as well as their transitive–like extensions and characteristics in the suggested way, a concept language $\mathcal{P}$ is defined where role terms are built from role names using the following role–forming operators:

a top role ($\top \times \top$), inverse roles (r^{-1}), role conjunction ($r \sqcap s$), role disjunction ($r \sqcup s$), role composition ($r \circ s$) and transitive closure of roles (r^*).

Concept terms are built from role terms and concept names using the following concept–forming operators:

concept conjunction ($C \sqcap D$), primitive concept negation ($\neg A$, where A is a primitive concept), exists restriction ($\exists\ r.\ C$), value restriction ($\forall\ r.\ C$), primitive single restriction ($\leq\ 1\ r_p\ :\ C$).

Concept terms and role terms are interpreted in the usual manner. Note that $\mathcal{P}$ includes neither concept disjunction nor a top concept and that we can restrict single restrictions to primitive or negated primitive roles r_p because of the following equivalence

$$(\leq\ 1\ r \circ s\ :\ C) \equiv \quad (\leq\ 1\ r\ :\ (\exists\ s.\ C)) \sqcap (\forall\ r.(\leq\ 1\ s\ :\ C)).$$

Investigation of the decidability of the satisfiability of concept terms in $\mathcal{P}$ led us to the question whether concept disjunction in general can be expressed using role disjunction, inverse roles and composition of roles. This would mean that renouncement of concept disjunction with the goal of achieving lower complexity is pointless if we have such strong role–forming operators.

Lemma 1 Concept disjunction can be expressed in $\mathcal{P}$.

More formally: Let C be a concept term of $\mathcal{P}$ including (possibly nested) concept disjunction. Then a concept term $\hat{C}$ of $\mathcal{P}$ can be constructed with the following properties: Each interpretation I of C can be extended to an interpretation $\hat{I}$ of $\hat{C}$ such that $(*)$ holds.

$$\begin{array}{c} \mathrm{dom}(I) = \mathrm{dom}(\hat{I}) \text{ and} \\ \forall x^I \in \mathrm{dom}(I) : x^I \in C^I \text{ iff } x^{\hat{I}} \in C^{\hat{I}}. \end{array} \quad (*)$$

Vice versa, if $\hat{I}$ is an interpretation of $\hat{C}$, and I is the restriction of $\hat{I}$ to role and concept names in C, then $(*)$ is satisfied. The idea in the construction of $\hat{C}$ is to substitute concept terms of the form $D \sqcup E$ by

$$(\exists(d \sqcup e).N) \sqcap (\forall d \circ d^{-1}.D) \sqcap (\forall e \circ e^{-1}.E),$$

where d, e are new primitive roles and N is a new primitive concept.

Corollary 1 Satisfiability of concept terms in $\mathcal{P}$ is undecidable.

Using lemma 1 and observing that

1. a top concept $\top$ can be simulated by (`C` $\sqcup \neg$`C`),
2. global features (functional roles) can be expressed (let `r`$_1$`,...,r`$_n$ be all role names appearing in a concept term `C`, then in all connected models of `C` $\sqcap$ (∀ (`r`$_1$ $\sqcup$ `r`$_1^{-1}$ $\sqcup \ldots \sqcup$ `r`$_n^{-1}$)*`.(≤ 1 r:`$\top$`)`) the role `r` is interpreted as a feature),

3. role value maps on feature chains can be simulated, for example $(\sqsubseteq$ `f g`$)$ as
   $((\leq$ `1 (f` $\sqcup$ `g):` $\top) \sqcup (\forall$ `f.`$\bot))$,

Corollary 1 follows from undecidability of $\mathcal{FSL}$ with role intersection [Schild,1991] or can easily be shown by a reduction of the domino problem.

Note, that in [Baader and Hanschke,1993] it was shown that the extension of $\mathcal{ALC}$ by functional roles, transitive closure of roles and integration of concrete domains leads to undecidable inference problems. This means that a knowledge representation system able to handle part–whole relations can not "calculate". Hence all constants (especially landmark–values) and their comparisons will have to be treated symbolically, and their exact treatment will be left to the user or a numerical system – which will be needed anyway to continue the modeling process.

Possible ways out

One way to handle undecidability (or the high complexity of a reduced language) is to use incomplete algorithms to solve satisfiabilty or subsumption problems. The aim of this knowledge base is not "just" to classify concepts and primitives in process engineering, but to support modeling of chemical plants, which means that the user has to be able to interprete answers given by his/her system. For example, if a user wants to know whether a concrete model of a chemical process contains a phase which is carcinogenic, then an incorrect answer "yes" is not dangerous. By contrast, an incorrect answer "yes" to the question whether all phases contained in this plant are eatable could be dangerous. This means that the semantic of answers given by an incomplete inference algorithm has to be known. Furthermore, as the one asking questions has to ask "good" questions in order to get no dangerous answers, users have to know about this semantic. There are two first ideas for incomplete reasoning with part–whole relations where the meaning of incompleteness is well defined.

A first approach, which can be viewed as a special case of reasoning with incomplete inference algorithms where only "yes" answers to satisfiability questions and "no" answers to subsumption questions can be incorrect, is to disallow role conjunction. This modification of $\mathcal{P}$ yields a concept language with decidable inference problems. As shown in [Schild,1991; De Giacomo and Lenzerini,1994b; De Giacomo and Lenzerini,1994a], subsumption is decidable for this highly expressive sublanguage, even provided with full concept negation. For our application, this means that we have to allow for models where direct part–whole roles are no longer interpreted as disjoint relations and with possibly cyclic part–whole relations, whereas all part–whole relations as well as their transitive–like extensions and their special characteristics can still be expressed.

The next approach is just an idea – and might or might not work: It is to modify the language $\mathcal{P}$ in order to decrease the complexity of its inference algorithms by substituting the transitive closure of a role R – which is the smallest transitive role extending R – by a role R' which is transitive and contains R. Let us call such a role R' the transitive orbit of R. There are hints from modal logic that reasoning with transitive orbits is easier than with transitive closures. The consequences for the application are the following: A TBox $\mathcal{T}$ where R' is the transitive orbit of R has more models than the same TBox $\mathcal{T}'$, with R' beeing the transitive closure of R. Hence theorems of $\mathcal{T}$ are subsets of theorems of $\mathcal{T}'$. This means that "C subsumes D" can be true in $\mathcal{T}'$ while it is not true in $\mathcal{T}$, whereas a concept can be satisfiable in $\mathcal{T}$ and unsatisfiable in $\mathcal{T}'$. Hence, as in the above mentionned case, only "yes" answers to satisfiabilty questions and "no" answers to subsumption questions can be incorrect.

These and other possibilities will be thoroughly investigated.

Acknowledgement

I would like to thank Franz Baader, Diego Calvanese and the anonymous referees for valuable suggestions and comments.

References

[Artale *et al.*, 1994] A. Artale, F. Cesarini, E. Grazzini, F. Pippolini, and G. Soda. Modelling composition in a terminological language environment. In *Workshop Notes of the ECAI Workshop on Parts and Wholes: Conceptual Part-Whole Relations and Formal Mereology*, pages 93–101, Amsterdam, 1994.

[Baader and Hanschke, 1993] Franz Baader and Philipp Hanschke. Extensions of concept languages for a mechanical engineering application. In *Proc. of the 16th German AI-Conference, GWAI-92*, volume 671 of *LNCS*, pages 132–143, Bonn, Deutschland, 1993. Springer-Verlag.

[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*, volume 567 of *LNAI*, pages 67–86. Springer-Verlag, 1991.

[De Giacomo and Lenzerini, 1994a] Giuseppe De Giacomo and Maurizio Lenzerini. Boosting the correspondence between description logics and propositional dynamic logics (extended abstract). In *Proc. of AAAI-94*, 1994.

[De Giacomo and Lenzerini, 1994b] Giuseppe De Giacomo and Maurizio Lenzerini. Concept language with number restrictions and fixpoints, and its relationship with mu-calculus. In *Proc. of ECAI-94*, 1994.

[Franconi *et al.*, 1994] Enrico Franconi, Alessandra Giorgi, and Fabio Pianesi. A mereological characterization of temporal and aspectual phenomena. In Carlos Martin-Vide, editor, *Current Issues in Mathematical Linguistics*, pages 269–278. Elsevier, North-Holland Linguistic Series, 1994.

[Gerstl and Pribbenow, 1993] Peter Gerstl and Simone Pribbenow. Midwinters, end games and bodyparts. In N. Guarino and R. Poli, editors, *International Workshop on Formal Ontology-93*, pages 251–260, 1993.

[Padgham and Lambrix, 1994] Lin Padgham and Patrick Lambrix. A framework for part-of hierarchies in terminological logics. In Jon Doyle, Erik Sandewall, and Pietro Torasso, editors, *Proc. of KR-94*, pages 485–496, 1994.

[Schild, 1991] Klaus Schild. A correspondence theory for terminological logics: Preliminary report. In *Proc. of IJCAI-91*, pages 466–471, Sydney, 1991.

[Winston *et al.*, 1987] M.E. Winston, R. Chaffin, and D. Herrmann. A taxonomy of part whole relations. *Cognitive Science*, 11:417–444, 1987.

Parallelizing Description Logics

Frank W. Bergmann and J. Joachim Quantz
Technische Universität Berlin, Projekt KIT-VM11

Abstract

Description Logics (DL), one of the major paradigms in Knowledge Representation, face efficiency problems due to large-scale applications, expressive dialects, or complete inference algorithms. In this paper we investigate the potential of parallelizing DL algorithms to meet this challenge. Instead of relying on a parallelism inherent in logic programming languages, we propose to exploit the application-specific potentials of DL and to use a more data-oriented parallelization strategy that is also applicable to imperative programming languages. We argue that object-level propagation is the most promising inference component for such a parallelization, as opposed to normalization, comparison, or classification.

We present two alternative PROLOG implementations of parallelized propagation on a loosely coupled MIMD (Multiple Instruction, Multiple Data) system, one based on a *farm* strategy, the other based on *distributed objects*. Whereas the farm strategy yields only poor results, the implementation based on distributed objects achieves a considerable speedup, in particular for large-size applications.

1 Introduction

In the last 15 years Description Logics (DL) have become one of the major paradigms in Knowledge Representation. Combining ideas from Semantic Networks and Frames with the formal rigor of First Order Logic, research in DL has focussed on theoretical foundations [Donini Et Al. 91] as well as on system development [Brachman Et Al. 91] and application in real-world scenarios [Quantz, Schmitz 94].

Whereas in the beginning it was hoped that DL provide representation formalisms which allowed efficient computation, at least three trends in recent years caused efficiency problems for DL systems and applications:

- a trend towards expressive dialects;
- a trend towards complete inference algorithms;
- a trend towards large-scale applications.

With the current state of technology it seems not possible to build a DL system for large-scale applications which offers an expressive dialect with complete inference algorithms. The standard strategy to cope with this dilemma is to restrict either expressivity, or completeness, or application size.

In this paper we investigate an alternative approach, namely a parallelization of Description Logics. Due to physical limitations in performance gains in conventional processor architectures, parallelization has become more and more important in recent years. This comprises parallel structures inside processors as well as outside by scaling several processors to parallel systems.

Several fields of high-performance computing already adopted to this new world of paradigms, such as image processing [Burkhard Et Al 94], finite element simulation [Diekmann Et Al 94], and fluid dynamics [Strietzel 94]. We expect that in future parallelism will become a standard technique in the construction of complex AI applications.

A standard approach to parallelization in the context of logic programming concentrates on the development of parallel languages that exploit the parallelism *inherent* in the underlying logic formalism ([Clark, Gregory 87], [Pontelli, Gupta 94] and many more). In this paper we will follow a rather different approach which analyzes a particular application, namely Description Logics. The parallelization we propose uses *explicit* parallelism based on the notion of processes and messages that is programming language independent.

In the next section we give a brief introduction into Description Logics and the parallelization potential of DL inference algorithms. In Section 3 we describe two different strategies of parallelizing object-level propagation in DL systems. The corresponding implementations are evaluated in detail in Section 4.

2 Description Logics

In this section we give a brief introduction into Description Logics. In doing so we sketch the main inference components of a DL system and point out their complexity and their potential for parallelization.

Basically, the alphabet of a Description Logic contains *concepts* (unary predicates), *Roles* (binary predicates), and *objects* (individual constants). DL dialects vary wrt the term-forming operators they support, e.g. conjunction, disjunction, and negation for concepts and roles; value restrictions and number restrictions for concepts; composition and inversions of roles.

Three types of formulae are usually distinguished in DL systems, namely *term introductions*[1] ($t_n :< t$ or $t_n := t$), *rules* ($c_1 => c_2$), and *object descriptions* ($o :: c$). Given a modeling, i.e. a list of such formulae, DL systems basically answer two types of queries:

$$\begin{array}{ccc} t_1 & ?< & t_2 \\ o & ?: & c \end{array}$$

The first query succeeds iff the term t_1 is subsumed by the term t_2(i.e. t_2 is more general than t_1), the second one iff the object o is an instance of the concept c.

Two main reasoning paradigms are used in DL systems. Originally, inferencing was realized by *normalize-compare* algorithms, which first transform concepts and roles into normalforms and then structurally compare these normalforms. Note that the normalforms of objects are similar to the normalforms of concepts and can thus be generated and compared by the same algorithms.

At the end of the 1980's *tableaux methods*, as known from FOL were applied to DL (e.g. [Donini Et Al. 91]). The resulting subsumption algorithms had the advantage of providing an excellent basis for theoretical investigations. Not only was their correctness and completeness easy to prove, they also allowed a systematic study of the decidability and the tractability of different DL dialects.

The main disadvantage of tableaux-based subsumption algorithms is that they are not constructive but rather employ refutation techniques. Thus in order to prove the subsumption $c_2 \sqsubseteq c_1$ it is proven that the term $c_1 \sqcap \neg c_2$ is inconsistent, i.e. that $o :: c_1 \sqcap \neg c_2$ is not satisfiable. Though this is straightforward for computing subsumption, this approach leads to efficiency problems in the context of retrieval. In order to retrieve the instances of a concept 'c' in a situation 's', we would in principle have to check for each object 'o' whether $\Gamma \cup \{o :: c \text{ in } s\}$ is satisfiable.[2]

In most existing systems, on the other hand, inference rules are more seen as production rules, which are used to pre-compute part of the consequences of the initial information. This corresponds more closely to Natural Deduction or Sequent Calculi, two deduction systems also developed in the context of FOL. A third alternative, combining advantages of the normalize-compare approach and tableaux-based methods has therefore been proposed in [Royer, Quantz 92]. The basic idea is to use *Sequent Calculi* instead of tableaux-based methods for the characterization of the deduction rules. Like tableaux methods, sequent calculi provide a sound logical framework, but whereas tableaux-based methods are refutation based, i.e. suitable for theorem checking, sequent calculi are constructive, i.e. suitable for theorem proving.

Based on the ideas presented in [Royer, Quantz 92; Royer, Quantz 94] the DL system FLEX has been developed at the Technische Universität Berlin. The parallelization described in the following has been performed for the FLEX system. The general techniques are applicable to all normalize-compare systems, however, and the following presentation does not rely on the particular features of FLEX.

In the remainder of this section we briefly sketch three main inference components of DL systems, describe their realization within the normalize-compare paradigm, and evaluate their potential for parallelization.

Subsumption Checking. To test subsumption between two terms, both terms are first *normalized* and then structurally compared. The format of normalization rules depends on the expressiveness of the DL dialect. For the purpose of this paper it is sufficient to consider normalforms as sets of atoms and normalization rules as having the form

$$\alpha_1, \ldots, \alpha_n \quad \rightharpoonup \quad \beta$$

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

In the comparison phase it is then checked whether for each atom in the subsuming normalform we find an atom in the subsumed normalform which is more specific.

Classification. When processing the term introductions, each term name is classified, i.e. compared with all previously introduced names. As a result *subsumption hierarchies* for concepts and roles are obtained, which are directed acyclic graphs. Classification is basically realized by searching direct supers and direct subs in the subsumption hierarchy, i.e. by a number of subsumption checks between the new term and previously introduced terms.

Propagation. The two reasoning components described so far are usually called *terminological reasoning*. We will now turn our attention towards *assertional reasoning*, i.e. reasoning on the object level. The main difference between terminological and assertional reasoning is that the former is inherently local, whereas the latter is inherently global. In principle we can distinguish between a local phase and a nonlocal phase in object-level reasoning.

In the local phase we determine for an object the *most specific concept* it instantiates. This can be done by using the standard normalize and compare predicates and the search for direct supers in the classification component. Thus we normalize the description of an object thereby obtaining a normal form and compare it with

[1] We use 'term' to designate both concepts and roles.

[2] See [Schaerf 94] for tableaux-based algorithms for object-level reasoning and [Kindermann 95] for a discussion of efficiency problems.

the normal forms of the concepts in the hierarchy. In addition to this standard classification we also have to apply DL rules when processing objects. This is achieved by applying all rules whose left-hand sides subsume the object's normal form. After this application the normal form is again normalized and classified until no new rules are applicable [Owsnicki-Klewe 88].

In the nonlocal phase we have to propagate information to other objects. For illustration consider the following propagation rules:

$$
\begin{array}{rcl}
o_1 :: \text{all(r,c)} \;\&\; r{:}o_2 & \rightharpoonup & o_2 :: c \\
o_1 :: r{:}o_2 \;\&\; \text{atmost(1,r)},\; o_2 :: c & \rightharpoonup & o_1 :: \text{all(r,c)} \\
o_1 :: r_1{:}o_2,\; o_2 :: r_2{:}o_3 & \rightharpoonup & o_1 :: r_1 \text{comp } r_2{:}o_3 \\
o_1 :: r{:}o_2 & \rightharpoonup & o_2 :: \text{inv(r)}{:}o_1
\end{array}
$$

We call these rules nonlocal since information at an object o_1 can have impact on an object o_2. Depending on the "connectivity" of the objects, adding a new description at an object can thus cause a reclassification of arbitrarily many other objects.

Parallelization. In principle, all three inference components show some parallel potential. We will argue, however, that parallelizing propagation is the most promising. The reason for this is that the basic operations in normalization, comparison, and classification are rather fine-grain, compared with the message passing overhead of MIMD systems.

Object-level propagation, on the other hand, is an ideal candidate for our parallelization strategy. Each propagation is rather time-consuming and causes additional propagations which can be straightforwardly parallelized since they are both independent and monotonic. In the following section we will present two different strategies for parallelizing propagation.

3 Parallelization Strategies

We begin by noting several relevant properties of object-level propagation. As already indicated above propagation of information from one object to another can cause additional propagation to other objects. This kind of 'ping-pong' interaction terminates only when a 'fixed point' is reached and no new information is produced. Since propagation in Description Logics is monotonic, we can execute propagations in an arbitrary order, always ending up with the same result. We will refer to this property as *confluence*.

FLEX Data Flow. Figure 3 shows the first few stages after the start of a propagation process. In this example every propagation causes three other propagations. This creates a 'chain reaction', thus increasing the number of 'pending propagations' exponentially. This rise will stop as soon as the new information (from the object tell) becomes more and more integrated into the network.

This results in a smaller 'fan out' that leads to a decrease of pending propagations until the fixed point is reached. Figure 3 shows qualitatively the increase and decrease of pending propagations with respect to to propagation steps. Please note that the steps in the middle part take much longer than the steps at the sides, because each processor (only a limited number of processors available) has to compute several propagations.

Given the analysis of the FLEX data flow, we consider two parallel paradigms as potential candidates for an implementation: The *farm* paradigm and the *distributed objects* paradigm. In the remainder of this section we briefly present these two alternatives. Theoretical considerations and numerical calculations towards efficiency and scalability can be found in the detailed analysis in [Bergmann 95].

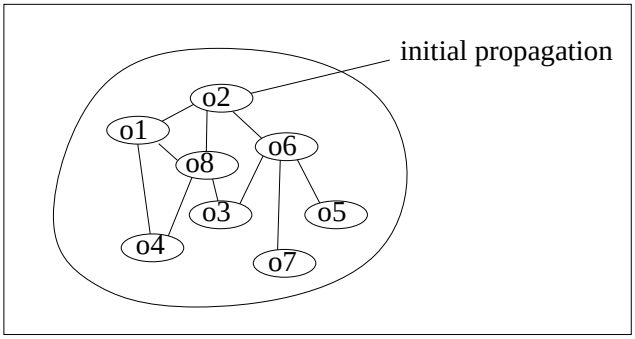


Figure 1: A group of objects interchanging propagations.

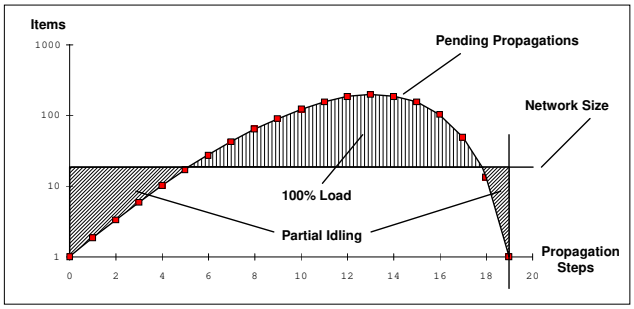


Figure 2: Exponential increase of propagations.

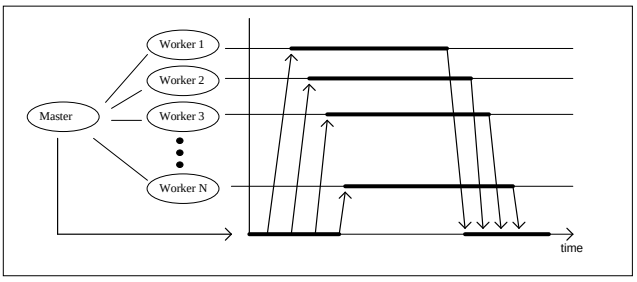


Figure 3: Timing of the farm communication scheme.

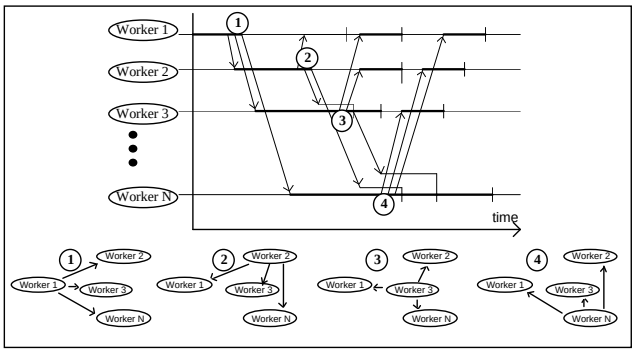


Figure 4: Communication events and workload distribution during the first two Propagation Stages.

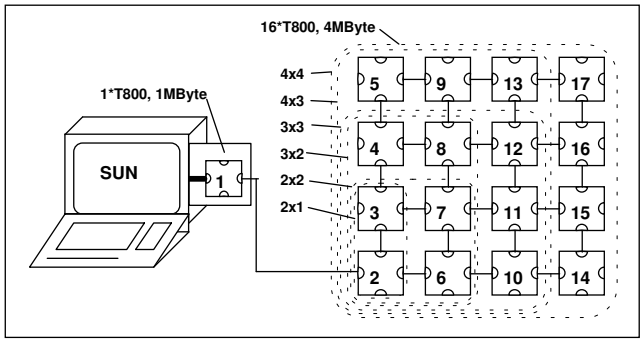
Farm Parallelism. The *farm* communication structure shown in Figure 3 is widely used in industrial applications such as image processing [Burkhard Et Al 94] and finite element simulation [Diekmann Et Al 94]. It is theoretically well known and there exists a variety of strategies to distribute workload evenly across a network.

A farm consists of several parallel processes with a fixed structure: one process is called 'master' and is responsible to distribute tasks to a group of 'worker' processes which perform their tasks in parallel and return control and results back to the master. Farm structures are frequently used to parallelize applications that can be split into subtasks with a priori known duration. Examples are image processing or finite element systems. From a theoretical point of view, there are two potential sources of inefficiency in this architecture:

1. uneven distribution of workload and
2. a communication bottleneck created by the centralized position of the master.

Communicating Objects Parallelism. In the *communicating-objects* paradigm the central management institution (master) of the farm parallelism is replaced by (local) knowledge of all objects about the 'addresses' of their neighbors. Objects communicate directly with each other, in contrast to the centered communication scheme of the farm. This helps to avoid communication bottlenecks in a network. The general difference between a farm and a network of communicating objects is the different perspective of parallelism: Within a farm, tasks are distributed; within the distributed objects scheme, objects are distributed.

This approach appears to be similar to the agent-based paradigm developed by distributed AI research [Dossier 91]. In contrast to this approach, objects within FLEX have to be considered elements of a distribution strategy rather then independently interacting entities. With respect to the definition given in [Bond, Gasser 88] we have to subsume our efforts here under the field of 'distributed problem solving'.



Topology	2x1	2p1	2x2	3x2	3x3	4x3	4x4
Processors	3	4	5	7	10	13	17
Workers	1	2	3	5	8	11	15

Figure 5: Hardware configuration and working nodes.

For an effective balancing of workload, certain assumptions about tasks and the computational environment have to be made. In our case, all processors can be assumed to behave identical and the statistical distribution of the task length is assumed to be narrow. Uneven distributions of workload can finally be treated by special load balancing algorithms (see below).

4 Experimental Results

We chose the 'Parsytec Multicluster II' machine as base for the parallel implementation of FLEX. It consists of 16 processing nodes that each contain an INMOS T800 Transputer with 4 MByte of RAM. Each Transputer consists of a RISC processing kernel, a memory interface and 4 DMA driven serial interfaces, called 'links'. Each link has a transfer rate of approximately 1.2 MByte/s and all 4 links can run independently together with the RISK kernel, hardly affecting processing performance (communication delays could be neglected). This hardware platform is especially suitable to serve as a testbed for parallel implementations due to its simple architecture and the availability of comfortable development environments. However, it does not provide state of the art computational performance and suffers substantially from memory restrictions.

Figure 5 shows the topologies used for the tests in this chapter and the number of available worker nodes. The overhead of 2 processors is due to memory limitations. Processor 1 could not be used because its 1 MByte RAM is not sufficient to hold the FLEX code. Processor 2 is used to hold the 'shell' process that synchronizes the generation of new objects. Normally this process can be located somewhere in the network and would not consume any computing performance, but in this case it had to be separated due to memory restrictions.

The language used to implement Distributed FLEX is a Prolog dialect called Brain Aid Prolog (BAP). It represents a 'standard' Prolog with parallel library extensions, implementing a scheme similar to CSP [Hoare 85]. Paral-

r	:<	rtop	o1	::	r:o3 and r:o2 and r:o8
c1	:<	all(r,c2)	o2	::	r:o4 and r:o7 and r:o2
c2	:<	all(r,c3)	o3	::	r:o7 and r:o2 and r:o1
c3	:<	all(r,c1)	o4	::	r:o1 and r:o8 and r:o6
			o5	::	r:o2 and r:o7 and r:o8
			o6	::	r:o1 and r:o7 and r:o5
			o7	::	r:o3 and r:o8 and r:o4
			o8	::	r:o7 and r:o4 and r:o6
			o1	::	c1

Figure 6: A sample benchmark with 8 Objects, 3 Concepts and Fanout 3.

lelism and synchronization is expressed explicitly using the notion of 'processes' and 'messages'. A process in BAP is a single and independent Prolog instance with a private database. A message is any Prolog term that becomes exchanged between two processes. Messages are send and received using the send_msg(Dest, Msg) and rec_msg(Sender, Msg) predicates. Message sender and destination are identified by their 'process id' (PID). Messages are routed transparently through the network. The order of messages is maintained when several messages are send from the same sender to the same destination. When a message reaches its destination process, it is stored in a special database, called 'mailbox'. Each process owns its private mailbox in which the messages are stored FIFO.

Although the development of parallel FLEX was greatly simplified by the way BAP expresses parallelism, it is possible to apply the same parallel techniques to future FLEX implementation in programming languages such as LISP or C.

The main area of FLEX applications within the KIT research group is Natural Language Processing (NLP) [Quantz, Schmitz 94]. Unfortunately memory limitations kept us from using these applications as benchmarks. Instead we imitate the structure of our NL applications leading to benchmarks with similar behavior but much lower memory requirements.

Base of our considerations is the fact that in the NL applications each propagation creates a certain number of propagations to other objects. This results in an 'avalanche' of propagations, rising exponentially until the systems slowly reaches a fixed point. The average fan out (the number of propagations following an initial propagation) is a measure to describe this avalanche effect and turned out to be a major factor in the system performance.

To evaluate the FLEX performance with benchmarks of different sizes, we created a benchmark generator that is capable of generating randomly connected networks of objects. These benchmarks maintain a structure similar to our application while consuming much less memory resources. [3]

[3][Bergmann 95] analyzes quantitatively the influence of the avalanche *exponent* on the applicability of parallel execution.

	(seq)	1	2	3	5	8
c10_3_2	(58)	78	63	55	56	58
c20_3_2	(177)	253	185	159	160	162

Figure 7: Execution times (seconds) for the farm parallelization.

Figure 7 shows the execution times for the farm benchmarks. The first row contains the benchmark names that are composed by three numbers that indicate the number of objects, concepts and fanout respectively (for example 'c20_5_3' means that the benchmark consists of 20 objects, 5 concepts and a fan out of 3). The following rows give the execution times with respect to the number of processors. The '(seq)' fields gives the reference time of the (original) sequential version of FLEX.

The parallelization of FLEX using the farm paradigm showed very poor results. This can be explained by the rather high costs to distribute the system state out to the workers and to integrate the computation results back into the system state. Both activities have to be done sequentially, thus slowing down the parallel execution.

Although there is some potential for optimizing the FARM implementation, we stopped the development and focused on the distributed-object version of FLEX.

The parallelization of FLEX using the distributed objects paradigm turned out to be a lot more promising. Figure 8 shows the absolute execution times of the considered benchmarks. The names of the benchmarks are composed as in Figure 7.

Note that the execution times in Figure 8 are measured with an accuracy of ± 2 seconds. The sequential execution times (entries in the '1' row) for several benchmarks are not available due to the memory limitations. This means that it is not possible to calculate the relative speedup in such a line (future tests on Transputer machines with more memory will fill these gaps). This is the reason why we omited the speedup figures in all but the first 4 benchmarks.

The table shows high speedups (efficiencies $> 80\%$) for all benchmarks, if the number of objects exceeds the number of processors by a certain factor (between 5 and 10). This result can be interpreted by the perspective of Section 3, where we saw that network efficiently is dependent on the number of pending propagations in the network. If this number is too low, few processing nodes are busy, resulting in a bad network efficiency.

Within [Bergmann 95] the quantitative analysis shows that the propagation-processor ratio is more relevant to system performance than the overhead caused by message passing. [4] It also indicates how these problems can be overcome, allowing for even larger networks.

A major problem for all distributed systems is the balance of load inside the network. Within distributed

[4]This is valid for Transputer systems with 2..256 processors, 2D matrix topology and shortest path routing algorithm

	1		2		3		5		8		11		15
c10_3_2	(1.0)	59	(1.9)	30	(1.8)	32	(3.3)	18	(3.3)	18	(2.8)	21	(3.3)	18
c10_3_3	(1.0)	43	(1.5)	28	(1.5)	28	(2.4)	18	(2.7)	16	(2.5)	17	(2.9)	15
c10_3_4	(1.0)	327	(2.0)	159	(2.3)	141	(3.1)	105	(3.8)	87	(4.4)	74	(4.4)	73
c20_3_2	(1.0)	179	(1.8)	97	(3.0)	59	(3.1)	58	(4.0)	45	(4.8)	37	(4.5)	40
c20_3_3				145		129		58		56		66		51
c20_3_4				240		173		164		70		74		68
c20_5_3				527		355		173		155		141		160
c20_5_4				344		453		411		185		126		189
c40_3_2				314		176		137		105		111		72
c40_3_3						258		231		141		111		87
c40_3_4						569		467		264		319		230
c80_3_2						1032		665		225		200		181
c80_3_3										443		336		266
c80_3_4										947		662		583

Figure 8: Benchmark Execution Times.

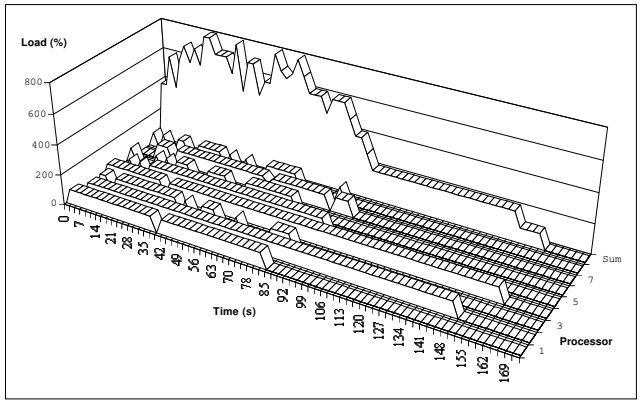


Figure 9: Runtime behavior of distributed FLEX within a 3x3 Network

FLEX each object represents a potential load. Unfortunately the presence of objects is only a statistical measure for load, while the actual distribution depends on runtime conditions. The illustration in Figure 9 depicts execution of a benchmark with an uneven load distribution. The Transputers 2 and 4 slow down the overall performance. It is easy to see that the network is quite busy during the first half of the execution time (ca. 75% efficiency). At the second half, all object servers have terminated, except two (ca. 25% efficiency). This leads to a reduction of the overall efficiency to ca. 50% and explains the variation of the results in Figure 8

The necessary optimization of the uneven distribution of processor load over the network can be achieved by temporarily 'relocating' objects to other processors. Such a mechanism would be capable of reducing overhead time created by loads remaining on a few processors. We are currently implementing this optimization.

5 Conclusion

The results of the parallel implementation of FLEX are in general very satisfying. We achieved high speedups with benchmarks that are structurally similar to the real-world applications in natural language processing ($> 80\%$ for benchmarks that fit the size of the network). The efficiency of execution rises mainly with the *propagation/processor* ratio and thus with the application size. This is an important result because especially large applications are to be considered candidates for a parallel implementation. Theoretical considerations [Bergmann 95] show that there are only few technical limits to the scalability of the distributed objects implementation.

We have to state that the Transputer system under consideration here is not applicable to real world problems due to its poor overall performance and its memory restrictions. Ideal candidates for such implementations are parallel computers with large (local) memory resources and high communication bandwidth. Alternatively, shared-memory multiprocessor workstations fulfill all requirements for an efficient parallelization.

We assume that the communication structure of FLEX is similar to many other applications in Artificial Intelligence. In particular, applications involving complex, forward-chaining inferencing are potential candidates for a parallelization based on the distributed-objects approach presented in this paper.

References

[BAP 93] F.W. Bergmann, M. Ostermann, G. von Walter, "Brain Aid Prolog Language Reference" Brain Aid Systems, 1993

[Bergmann 95] F.W. Bergmann, *Parallelizing FLEX*, KIT Report in preparation, TU Berlin

[Bond, Gasser 88] A. Bond, L. Gasser, "Readings in Distributed Artificial Intelligence", Morgan Kaufmann, Los Angeles, CA, 1988

[Brachman Et Al. 91] R. Brachman, D.L. McGuiness, P.F. Patel-Schneider, L. Alperin Resnick, A. Borgida, "Living with CLASSIC: When and How to Use a KLONE-like Language", in J. Sowa (Ed.), *Principles*

of Semantic Networks: Explorations in the Representation of Knowledge, San Mateo: Morgan Kaufmann, 1991, 401–456

[Burkhard Et Al 94] H. Burkhard, A. Bienick, R. Klaus, M. Nlle, G. Schreiber, H. Schulz-Mirbach, "The Parallel Image Processing Sytem PIPS" in R. Flieger, R. Grebe (eds), *Parallelrechner Grundlagen und Anwendung* IOS Press, Amsterdam, Netherlands, 1994, 288–293

[Clark, Gregory 87] K. Clark, S, Gregory, "PARLOG: Parallel Programming in Logic" in E. Shapiro (ed), *Concurrent Prolog* The MIT Press, Cambridge, Massachusetts, 1987, 84 – 139

[Diekmann Et Al 94] R. Diekmann, D. Meyer, B. Monien, "Parallele Partitionierung unstrukturierter Finite Elemente Netze auf Transputernetzwerken" in R. Flieger, R. Grebe (eds), *Parallelrechner Grundlagen und Anwendung* IOS Press, Amsterdam, Netherlands, 1994, 317–326

[Donini Et Al. 91] F.M. Donini, M. Lenzerini, D. Nardi, W. Nutt, "The Complexity of Concept Languages", KR'91, 151–162

[Dossier 91] A.C. Dossier, "Intelligence Artificielle Distribuee", *Bulletin de l'AFIA*, **6**, 1991

[Hoare 85] C. A. R. Hoare, "Communicating Sequential Processes" Prentice Hall, Englewood Cliffs, N.J., USA, 1985

[Kindermann 95] C. Kindermann, *Verwaltung assertorischer Inferenzen in terminologischen Wissensbanksystemen*, PhD Thesis (submitted), TU Berlin, 1995

[Owsnicki-Klewe 88] B. Owsnicki-Klewe, "Configuration as a Consistency Maintenance Task", in W. Hoeppner (Ed.), *Proceedings of GWAI'88*, Berlin: Springer, 1988, 77–87

[Pontelli, Gupta 94] E. Pontelli, G. Gupta, "Design and Implementation of Parallel Logic Programming Systems", *Proceedings of ILPS'94 Post Converence Workshop* 1994

[Quantz, Schmitz 94] J.J. Quantz, B. Schmitz, "Knowledge-Based Disambiguation for Machine Translation", *Minds and Machines* **4**, 39–57, 1994

[Royer, Quantz 92] 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, 1992, 84–105

[Royer, Quantz 94] V. Royer, J.J. Quantz, "On Intuitionistic Query Answering in Description Bases", in A. Bundy (Ed.), *CADE-94*, Berlin: Springer, 1994, 326–340

[Schaerf 94] A. Schaerf, *Query Answering in Concept-Based Knowledge Representation Systems: Algorithms, Complexity, and Semantic Issues*, Dissertation Thesis, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", 1994

[Strietzel 94] "Large Eddy Simulation turbulenter - Strömungen auf MIMD-Systemen" in R. Flieger, R. Grebe (eds), *Parallelrechner Grundlagen und Anwendung*, IOS Press, Amsterdam, Netherlands, 1994, 357 - 366

Implementing and Testing Expressive Description Logics: a Preliminary Report

Paolo Bresciani, Enrico Franconi and Sergio Tessaris
Knowledge Representation and Reasoning group
IRST, I-38050 Povo TN, Italy
{bresciani|franconi|tessaris}@irst.itc.it

Abstract

The aim of the CRACK project is the research and the development of a knowledge representation system based on description logics. The CRACK language differentiates itself from other knowledge representation systems for its high expressivity and its provably sound and complete reasoning procedures. With respect to other systems available in the research community CRACK is more expressive, it is expandable to new constructs, it treats the conceptual and individual levels in a homogeneous way, it is modular, it is comparably fast. However, CRACK algorithms do not work in polynomial space, i.e. in the (arguably rare in practice) worst cases they may require exponential memory. The performance of the system has been tested against several different classes of random knowledge bases, characterized by an order parameter generating phase transitions in the satisfiability probability space.

1 Introduction

In this paper we present the knowledge representation system CRACK, an implementation of an expressive description logic. In the full paper [Bresciani *et al.*,1995] more details on CRACK could be found. The greatest motivation for the CRACK project is the possibility of having a modular system, where the expressivity of the language could be extended, without re-implementing the control part of the system. Thus, experimentation with many operators and, if a very expressive language is implemented, with different sub-languages can be done.

Algorithms in CRACK are based on the tableaux calculus technique, according to the many recent theoretical works in description logics. However, differently from other implementations, rules of tableaux are directly implemented, allowing for a high grade of modularity by just adding/removing rules. As a further consequence of this choice, it turns out that the conceptual and the individual levels are treated in a uniform manner. The architecture of the system has been organized in such a way that different possible extensions of the language – involving many different condition testing for the rules, different basic data management, or different clash checking – can be easily added on top of the core system.

The final part of the paper is dedicated to the topic of testing such a system, by considering the problem of generating random *average* knowledge bases. Stimulated by the studies in the literature of the hardness of the satisfiability problems for some classes of random propositional formulae, we have tried to explore the probability space of satisfiability in the case of description logics. Such tests are just at the beginning, but we believe that it is a good direction to explore.

2 The $\mathcal{ALCRIFO}$ Description Logic

In the following, we will consider only $\mathcal{ALCRIFO}$, a significative fragment of the language currently implemented in CRACK, because it embeds the main peculiarities of the system. We will not go into details of the language, since it is just a collection of standard constructors in description logics. $\mathcal{ALCRIFO}$ extends the propositionally complete concept language $\mathcal{ALC}$ [Schmidt-Schauß and Smolka,1991] with role conjunction ($\mathcal{R}$)[Hollunder *et al.*,1990], inverse roles ($\mathcal{I}$) [Donini *et al.*,1991], feature selection, agreement and disagreement ($\mathcal{F}$) [Hollunder and Nutt,1990], enumerated types ($\mathcal{O}$) [Schaerf,1994]. Section 4 describes the full CRACK language and some of its possible extensions. Figure 1 defines syntax and semantics of $\mathcal{ALCRIFO}$.

Knowledge base satisfiability is the basic reasoning task, since all the relevant reasoning tasks – like concept satisfiability, concept subsumption, instance checking, retrieval – are reducible to it. Checking for KB satisfiability is deciding whether there is at least one model for a knowledge base Σ, made out of TBox axioms and ABox statements.

The tableaux-based calculus to decide the satisfiability of $\mathcal{ALCRIFO}$ knowledge bases operates on constraints. A constraint is an expression of the type $x : C$, xRy, xpy and $x\,neq y$, where x, y are objects, i.e. either individuals or variables belonging to a predefined set of variable symbols. Starting from a system S associated to a knowledge base – i.e. obtained by directly translating into constraints every ABox statement – the *propagation*

$C, D \rightarrow$	$A \mid$	
	$\top \mid$	$\Delta^{\mathcal{I}}$
	$\bot \mid$	$\emptyset$
	$\neg C \mid$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
	$C \sqcap D \mid$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
	$C \sqcup D \mid$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
	$\forall R.C \mid$	$\{i \in \Delta^{\mathcal{I}} \mid \forall j.\, R^{\mathcal{I}}(i,j) \rightarrow C^{\mathcal{I}}(j)\}$
	$\exists R.C \mid$	$\{i \in \Delta^{\mathcal{I}} \mid \exists j.\, R^{\mathcal{I}}(i,j) \wedge C^{\mathcal{I}}(j)\}$
	$p\!\uparrow \mid$	$\Delta^{\mathcal{I}} \setminus \mathrm{dom}\, p^{\mathcal{I}}$
	$p : C \mid$	$\{i \in \mathrm{dom}\, p^{\mathcal{I}} \mid C^{\mathcal{I}}(p^{\mathcal{I}}(i))\}$
	$p \doteq q \mid$	$\{i \in \mathrm{dom}\, p^{\mathcal{I}} \cap \mathrm{dom}\, q^{\mathcal{I}} \mid p^{\mathcal{I}}(i) = q^{\mathcal{I}}(i)\}$
	$p \not\doteq q \mid$	$\{i \in \mathrm{dom}\, p^{\mathcal{I}} \cap \mathrm{dom}\, q^{\mathcal{I}} \mid p^{\mathcal{I}}(i)\, neq\, q^{\mathcal{I}}(i)\}$
	$\{\, a_1 \ldots a_n \,\}$	$\{a_1^{\mathcal{I}} \ldots a_n^{\mathcal{I}}\}$
$R, Q \rightarrow$	$P \mid$	
	$R^{-1} \mid$	$\{(i,j) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid R^{\mathcal{I}}(j,i)\}$
	$R \sqcap Q$	$R^{\mathcal{I}} \cap Q^{\mathcal{I}}$
$p, q \rightarrow$	$f \mid$	
	$p \circ q$	$p^{\mathcal{I}} \circ q^{\mathcal{I}}$

Figure 1: Syntax and semantics for the $\mathcal{ALCRIFO}$ description logic.

rules are applied, until a contradiction (a *clash*) is generated or a model of Σ is explicitly obtained: the propagation rules preserve satisfiability. A clash is a system having one of the forms: $\{x : \bot\}$; $\{x : A,\ x : \neg A\}$; $\{x\, neq x\}$; $\{x : f\!\uparrow,\ xfy\}$; $\{xfa,\ xfb\}$ if a and b are individuals; $\{a : \{\, a_1 \ldots a_n \,\}\}$ with $a\, neq a_i$ for all i.

Figure 2 lists a complete set of propagation rules for $\mathcal{ALCRIFO}$; they are a simple variant of the standard rules found in the literature. A role $R = R_1 \sqcap \ldots \sqcap R_k \sqcap (R_{k+1}{}^{-1} \sqcap \ldots \sqcap R_n{}^{-1})$ is written as the pair $\langle R^+, R^- \rangle$ where $R^+ = \{R_1, \ldots, R_k\}$ and $R^- = \{R_{k+1}, \ldots, R_n\}$. We say that xRy *holds in* a constraint system S if xQy is in S and Q specializes R – i.e. $R^+ \subseteq Q^+$ and $R^- \subseteq Q^-$. For the calculus, we consider only concepts in a sort of normal form [Bresciani *et al.*,1995]. An arbitrary $\mathcal{ALCRIFO}$ concept can be transformed in polynomial time into an equivalent concept in the normal form.

Propagation rules are either deterministic – they yield a uniquely determined constraint system – or nondeterministic (e.g. $\rightarrow_{\sqcup}$) – yielding several possible constraint systems. A constraint system is said to be *complete* if no propagation rule is applicable to it. Because of the presence of nondeterministic rules, several complete systems can be derived from the starting one. A constraint system is satisfiable if and only if there exist a clash free complete constraint system which can be derived from it by applying the propagation rules.

$S \rightarrow_{\sqcap}$ $\{x : C_1,\ x : C_2\} \cup S$
if $x : C_1 \sqcap C_2$ in S,
and both $x : C_1$ and $x : C_2$ not in S

$S \rightarrow_{\sqcup}$ $\{x : D\} \cup S$
if $x : C_1 \sqcup C_2$ in S,
and neither $x : C_1$ nor $x : C_2$ in S,
and $D = C_1$ or $D = C_2$

$S \rightarrow_{\exists}$ $\{xRy,\ y : C\} \cup S$
if $x : \exists R.C$ in S,
and there is no z s.t. both xRz and $z : C$ in S,
and y is a new variable

$S \rightarrow_{\forall}$ $\{y : C\} \cup S$
if $x : \forall R.C$ in S, and xRy holds in S,
and $y : C$ not in S

$S \rightarrow_{-1}$ $\{yPx\} \cup S$
if $xP^{-1}y$ in S,
and yPx not in S

$S \rightarrow_{+1}$ $\{yP^{-1}x\} \cup S$
if xPy in S,
and $yP^{-1}x$ not in S

$S \rightarrow_{:}$ $\{xpy,\ y : C\} \cup S$
if $x : (p : C)$ in S,
and there is no z s.t. both xpz and $z : C$ in S,
and y is a new variable

$S \rightarrow_{\doteq}$ $\{xpy,\ xqy\} \cup S$
if $x : p \doteq q$ in S,
and there is no z s.t. both xpz and xqz in S,
and y is a new variable

$S \rightarrow_{\not\doteq}$ $\{xpy,\ xqz,\ yneqz\} \cup S$
if $x : p \not\doteq q$ in S,
and there are no y', z' s.t. xpy', xqz' and $y'neqz'$ in S,
and y, z are new variables

$S \rightarrow_{\circ}$ $\{xpz,\ zqy\} \cup S$
if $x(p \circ q)y$ in S,
and there is no w s.t. both xpw and wqy in S,
and z is a new variable

$S \rightarrow_{funct}$ $[y/z]S$
if xfy and xfz in S,
and y is not an object,
and y is not equal to z

$S \rightarrow_{\{\}}$ $[x/a_i]S$
if $x : \{\, a_1 \ldots a_n \,\}$ in S,
and x is not an object

$S \rightarrow_{\neg\{\}}$ $\{xneqa_1 \ldots xneqa_n\} \cup S$
if $x : \neg\{\, a_1 \ldots a_n \,\}$ in S

Figure 2: Propagation rules for the $\mathcal{ALCRIFO}$ description logic.

3 Implementing Tableaux Calculus

In this section we will show a general architecture for implementing reasoning systems based on tableaux calculus; in our case, it will be a procedure for deciding satisfiability of $\mathcal{ALCRIFO}$ knowledge bases.

The main idea is to implement directly the rules of tableaux calculus, as opposed to the currently implemented tableaux-based description logics relying on a *functional algorithm*, like e.g. KRIS [Baader and Hollunder,1991]. A functional algorithm exploits the property of independency between *traces* of a satisfiability proof. It assumes that a completed system can be partitioned into traces, where the computation can be performed independently – i.e. a clash can be gen-

erated only by constraints belonging to the same trace. The functional algorithm for $\mathcal{ALCR}$ can be sketched as follows [Hollunder *et al.*,1990]:

$sat(S)$ = **if** S includes a clash
then false
elseif $C \sqcap D \in S$ **and** $C \notin S$ **or** $D \notin S$
then $sat(S \cup \{C, D\})$
elseif $C \sqcup D \in S$ **and** $C \notin S$ **and** $D \notin S$
then $sat(S \cup \{C\})$ **or** $sat(S \cup \{D\})$
else forall $\exists R.C \in S$
$sat(\{C\} \cup \{D \mid \forall Q.\, D \in S \wedge R \sqsubseteq Q\})$

Unfortunately, it seems that expressive languages do not have this nice property. For example, inverse roles and one-of introduce interactions between traces. The unsatisfiability of the concept:

$\exists \texttt{CHILD}.(\texttt{Man} \sqcap \{\texttt{peter}\}) \sqcap \exists \texttt{CHILD}.(\neg \texttt{Man} \sqcap \{\texttt{peter}\})$

can not be detected with a trace-based algorithm. The two traces generated by the two existential quantifications on `CHILD` are independently satisfiable, but are globally unsatisfiable, since both existential variables should be co-referenced to the individual `peter`.

Thus, our general satisfiability algorithm does not assume any independency between traces, since it has to be used for very expressive description logics. This allows for deductions like that in the above example. However, there is a price to pay for this choice: sub-languages where the complexity of satisfiability is PSPACE-complete are not handled properly with such a non-trace algorithm. In fact, the plain non-deterministic application of the propagation rules may generate an exponential number of variables. For example, the completed system generated by the concept (of dimension n) [Donini *et al.*,1992a]:

$\exists R.C_1 \sqcap \exists R.C_2 \sqcap$
$\forall R.\ (\exists R.C_1 \sqcap \exists R.C_2 \sqcap \forall R.\ (\ldots \boxed{\cdots_n}\ )_2\ )_1$

has an exponential number of variables with respect to n, as shown below:

$x : \exists R.C_1 \sqcap \exists R.C_2 \sqcap \forall R.(\exists R.C_1 \sqcap \exists R.C_2 \sqcap \forall R.(\ldots))$
$xRx_1,\ x_1 : C_1$
$x_1 : \exists R.C_1 \sqcap \exists R.C_2 \sqcap \forall R.(\ldots)$
$\boxed{\begin{array}{l}\cdots\\ \cdots\end{array}}$
$xRx_2,\ x_2 : C_2$
$x_2 : \exists R.C_1 \sqcap \exists R.C_2 \sqcap \forall R.(\ldots)$
$\boxed{\begin{array}{l}\cdots\\ \cdots\end{array}}$

We believe that such worst cases are unlikely to happen in real cases, just as exponential in time worst cases do not happen in real and average knowledge bases; for a preliminary study on the meaning of *average knowledge base* refer to the section 5 of the paper.

Another peculiarity of our approach is that the computation is driven by the constraints, and not by the non-deterministic choice of applicable rules. At the beginning of a satisfiability proof, constraints are inserted in a queue and processed in a sequential manner. A constraint triggers the application of only one rule, and thereafter it could be either inserted in the database of permanent constraints – if the constraint itself could be necessary in the future for checking the application of some other rule triggered by some other constraint or for detecting a clash – or otherwise discarded. The application of a rule may generate other new constraints, which are pushed in the queue and further elaborated. Figure 3 presents the CRACK architecture.

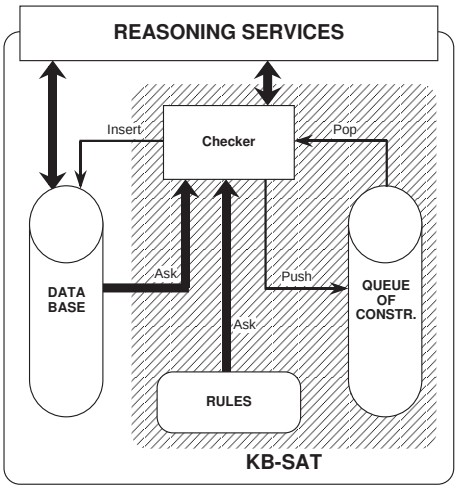


Figure 3: The CRACK architecture.

This architecture may deal with very expressive languages, and it can be extended to handle new propagation rules without having to reconsider the control mechanism of the system. Moreover, the treatment of new types of clashes or new types of operations on constraints can be easily added to the system by changing only the affected modules. In the following, the formal operational semantics describing the behavior of the system for handling $\mathcal{ALCRIFO}$ will be introduced, the data structures will be analyzed, and an abstract overview to the satisfiability algorithm itself will be given.

3.1 Operational Semantics

The operational semantics is defined through a formal system that allows to prove assertions of the form:

$$\langle S, DB \rangle \vdash v$$

where S is a sequence of constraints, DB is a database and v is either a database or the symbol **FAIL**. An assertion of this kind states that an "evaluation" of constraints in S over the database DB produces the result v. If v is **FAIL** the constraint set with respect to the database is unsatisfiable, otherwise the result is a new database DB' from which a model may be derivable.

A database is a pair $\langle \Sigma, \rho \rangle$, where Σ is a set of constraints and ρ is a rename function mapping objects into objects. Due to the unique name assumption, the rename function restricted to individuals must be the iden-

tity. The rename function may change as new constraints are added to the constraint system by the propagation rules; for this reason, we introduce the notation $\rho[z,r]$ – where ρ is a rename function – denoting a new rename function ρ' equal to ρ except that $\rho'(r) = z$. We also impose some restrictions on the structure of role and feature constraints in the database. For every pair of related objects there must be at most one role constraint – conjoining all the asserted role constraints – and if a role binds a pair of objects its inverse should bind the inverted pair of objects. A similar restriction is applied also for feature constraints (and their inverses). According to this, adding role or feature constraints involves some extra operations on the set; for sake of simplicity, will will ignore this in the following.

There are five types of constraints: instance, relation, attribute, inequality and equality constraints ($x : C$, $x\,R\,y$, $x\,f\,y$, $x neq y$, $x = y$). Only *persistent* constraints are stored in a database. They are those which may be used more than once during a satisfiability proof. For example, a universal constraint is processed any time a new filler for the role (i.e. a new role constraint) is added to the constraint system; on the contrary, an existential constraint is processed only once. It can be easily verified that persistent constraints are only of the kind: $x : A$, $x : \neg A$, $x : \forall R.\, C$, $x(R^+ \sqcap R^-)y$, xfy, $xf^{-1}y$, $x neq y$, $x : f\!\uparrow$, where R^+, R^- are conjunctions of roles and inverse roles respectively.

The formal system of operational semantics is given in form of pre-conditions which should be verified in order to prove the truth value of every assertion. Operationally, the control of the "executor" is fixed by the sequence of pre-conditions to check. Let's see, for example, the conditional assertion about the **and** concept:

$$\frac{\langle \pi(\rho x : D)(\pi(\rho x : C)S), \langle \Sigma, \rho\rangle\rangle \vdash v}{\langle <(x : C \sqcap D)\,.\,S>, \langle \Sigma, \rho\rangle\rangle \vdash v}$$

This formula states that the result of the evaluation on a sequence, whose head is the conjunction of two concepts, over a database ($\langle \Sigma, \rho\rangle$) is the same as the evaluation on a sequence obtained by adding two new constraints ($\rho x : C$ and $\rho x : D$) to the rest of the original sequence (S) over the same database. Please note the *strategy* two-place function π: it maps a constraint and a sequence of constraints into a new sequence of constraints. Intuitively, the strategy function inserts a new constraint into a sequence determining a "good" place for it. The strategy function does not modify a sequence when trying to add an already present constraint.

In the following the set of the conditional assertions for $\mathcal{ALC}$ is listed – the complete set for $\mathcal{ALCRIFO}$ can be found in [Bresciani *et al.*,1995]. The domains of the operational semantics are: objects names ($\mathcal{O}$), including individuals names ($\mathcal{J}$) and variables names ($\mathcal{V}$); Concept Names ($\mathcal{C}_{\text{nam}}$); Role Names ($\mathcal{R}_{\text{nam}}$); Feature Names ($\mathcal{F}_{\text{nam}}$); Concept Expressions ($\mathcal{C}$); Role Expressions ($\mathcal{R}$); Feature Expressions ($\mathcal{F}$). The terminology function $\mathcal{T}$ maps a concept name into its definition; if a concept name does not have a definition, the terminology function returns the name itself. The normal form function Nf maps a concept expression into its normal form.

Termination on Success

- $\dfrac{}{\langle \Box, \langle \Sigma, \rho\rangle\rangle \vdash \langle \Sigma, \rho\rangle}$

Bottom Concept

- $\dfrac{}{\langle <(x : \bot)\,.\,S>, \langle \Sigma, \rho\rangle\rangle \vdash \textbf{FAIL}}$

Atomic concept

- $\dfrac{\neg A(\rho x) \in \Sigma}{\langle <(x : A)\,.\,S>, \langle \Sigma, \rho\rangle\rangle \vdash \textbf{FAIL}}$

- $\dfrac{\begin{array}{l}\neg A(\rho x) \notin \Sigma \\ \mathcal{T}(A) = A \\ \Sigma' = \Sigma \cup \{A(\rho x)\} \\ \langle S, \langle \Sigma', \rho\rangle\rangle \vdash v\end{array}}{\langle <(x : A)\,.\,S>, \langle \Sigma, \rho\rangle\rangle \vdash v}$

- $\dfrac{\begin{array}{l}\neg A(\rho x) \notin \Sigma \\ \mathrm{Nf}(\mathcal{T}(A)) = C \qquad C neq A \\ \Sigma' = \Sigma \cup \{A(\rho x)\} \\ \langle \pi(\rho x : C)S, \langle \Sigma', \rho\rangle\rangle \vdash v\end{array}}{\langle <(x : A)\,.\,S>, \langle \Sigma, \rho\rangle\rangle \vdash v}$

Negated Atomic Concept

- $\dfrac{A(\rho x) \in \Sigma}{\langle <(x : \neg A)\,.\,S>, \langle \Sigma, \rho\rangle\rangle \vdash \textbf{FAIL}}$

- $\dfrac{\begin{array}{l}A(\rho x) \notin \Sigma \\ \mathcal{T}(A) = A \\ \Sigma' = \Sigma \cup \{\neg A(\rho x)\} \\ \langle S, \langle \Sigma', \rho\rangle\rangle \vdash v\end{array}}{\langle <(x : \neg A)\,.\,S>, \langle \Sigma, \rho\rangle\rangle \vdash v}$

- $\dfrac{\begin{array}{l}A(\rho x) \notin \Sigma \\ \mathrm{Nf}(\neg\mathcal{T}(A)) = C \qquad C neq \neg A \\ \Sigma' = \Sigma \cup \{\neg A(\rho x)\} \\ \langle \pi(\rho x : C)S, \langle \Sigma', \rho\rangle\rangle \vdash v\end{array}}{\langle <(x : \neg A)\,.\,S>, \langle \Sigma, \rho\rangle\rangle \vdash v}$

And concept

- $\dfrac{\langle \pi(\rho x : D)(\pi(\rho x : C)S), \langle \Sigma, \rho\rangle\rangle \vdash v}{\langle <(x : C \sqcap D)\,.\,S>, \langle \Sigma, \rho\rangle\rangle \vdash v}$

Or concept

- $\dfrac{\begin{array}{l}\langle \pi(\rho x : C)S, \langle \Sigma, \rho\rangle\rangle \vdash v \\ v neq \textbf{FAIL}\end{array}}{\langle <(x : C \sqcup D)\,.\,S>, \langle \Sigma, \rho\rangle\rangle \vdash v}$

- $\dfrac{\begin{array}{l}\langle \pi(\rho x : C)S, \langle \Sigma, \rho\rangle\rangle \vdash \textbf{FAIL} \\ \langle \pi(\rho x : D)S, \langle \Sigma, \rho\rangle\rangle \vdash v\end{array}}{\langle <(x : C \sqcup D)\,.\,S>, \langle \Sigma, \rho\rangle\rangle \vdash v}$

Some concept

- $\dfrac{\begin{array}{l}y \text{ new variable} \\ \rho' = \rho[y,y] \\ \langle \pi(y : C)(\pi(\rho x\, R\, y)S), \langle \Sigma, \rho'\rangle\rangle \vdash v\end{array}}{\langle <(x : \exists R.\, C)\,.\,S>, \langle \Sigma, \rho\rangle\rangle \vdash v}$

Universal concept

- $\dfrac{\begin{array}{l}S' = \mathbf{min}_{\mu} \supseteq S \text{ s.t.} \\ \quad \forall Q(\rho x, y) \in \Sigma.\, Q \sqsubseteq R \Rightarrow \mu = \pi(y : C)\mu \\ \langle S', \langle \Sigma, \rho\rangle\rangle \vdash v\end{array}}{\langle <(x : \forall R.\, C)\,.\,S>, \langle \Sigma, \rho\rangle\rangle \vdash v}$

Role

- $\dfrac{\begin{array}{l}S' = \mathbf{min}_{\mu} \supseteq S \text{ s.t.} \\ \quad (\forall(\forall Q.\, C)(\rho x) \in \Sigma.\, R \sqsubseteq Q \Rightarrow \mu = \pi(\rho y : C)\mu) \wedge \\ \quad (\forall(\forall Q.\, C)(\rho y) \in \Sigma.\, R^{-1} \sqsubseteq Q \Rightarrow \mu = \pi(\rho x : C)\mu) \\ \langle S', \langle \Sigma, \rho\rangle\rangle \vdash v\end{array}}{\langle <(x\, R\, y)\,.\,S>, \langle \Sigma, \rho\rangle\rangle \vdash v}$

Slot name	*Content*	*Indexed by*
name	A	*none*
preceding-entry	$\longmapsto$ entry	*none*
is-variable	{T|F}	*none*
Constraints:		
isa	$\langle A_1, \ldots\rangle$	concept name
isnt	$\langle A_1, \ldots\rangle$	concept name
universals	$\langle [R_1 \,.\, C_1], \ldots\rangle$	*none*
roles	$\langle [x_1 \,.$ direct $\longmapsto$ role-struct / inverse $\longmapsto$ role-struct$], \ldots\rangle$	related object
features	$\langle [x_1 \,. \longmapsto$ feature-struct$], \ldots\rangle$	related object
inv-features	$\langle [x_1 \,. \longmapsto$ feature-struct$], \ldots\rangle$	related object
not-equal	$\langle x_1, \ldots\rangle$	object name
undefinedness	$\langle f_1, \ldots\rangle$	feature name

Figure 4: The data structure for an entry.

3.2 The Data Structures

A special data structure is used to implement the database, in order to allow for efficient computation of the applicability conditions of the rules, for a fast detection of clashes, and for an effective insertion of new constraints in the database itself. Moreover, the data structure is capable to cope with the non-determinism of the database updating, and with the problem of variable renaming.

The data structure has been organized in a way that new rules can be added without reimplementing the database structure. Thus, the expressivity of the language can be extended by just adding new rules, being not necessary to change the whole system architecture or the data structures.

The *Constraint System* is the structure for the database, holding descriptions of persistent constraints. A constraint system is represented by a stack of *frames*. Each frame is a backtrack point for a non-deterministic rule: when a non-deterministic rule is applied, a new empty frame is pushed on the top of the stack. The top frame is called *active* frame. The active frame is the location where the information is stored at a certain point of the non-deterministic computation. Backtracking pops the top frame from the constraint system, such that the preceding frame becomes active.

A *Frame* is the data structure for a deterministic part of a database. It is a pair consisting of a table of *entries* indexed by objects (either variables or individuals) and a mapping from variables to objects – implementing a rename function. Note that an index may appear, obviously, at most once within a frame; however, the same object may be an index for different entries in different frames. The mapping binds variables to objects and allows to collapse two or more objects without an explicit change in the frame structure. The rename function is defined only through the mapping in the active frame.

An *Entry* is the data structure for persistent constraints indexed by objects – see figure 4 for details. It maintains information about constraints which must be stored during a satisfiability proof. Associated to each object, there are the persistent constraints containing the object itself. A role (or feature) structure contains the information about the relation between two objects – see figure 5 for details.

In the constraint system there is always only one active role-structure between the same pair of objects: the last one added. When a new relation is established between two objects already related in a non-active frame, the old relation is copied from the newest of the non-active frames and updated in the current active frame. On the contrary, information on attributes is distributed over the frame chain. The difference between an attribute and a role is that the latter needs an efficient handling of the *holds* test predicate.

Every instance of the above described structures is local to a single frame; thus, entries of different frames do not share any structure. This is necessary for the backtracking mechanism, since it is necessary to discard the last frame without any change in the previous ones.

3.3 The Satisfiability Abstract Algorithm

During the computation, a structure called *constrack* maintains constraints before their use. The satisfiability checker iterates over such constraints, applies the propagation rules and inserts persistent constraints in the constraint system. The constrack structure is indexed by object names and it is simpler than the constraint system structure, since it does not allow for complex in-

+	$\langle R_1, \ldots\rangle$	$\leftarrow$ direct relations (R^+)
−	$\langle R_1, \ldots\rangle$	$\leftarrow$ inverse relations (R^-)

$\langle f_1, \ldots\rangle$ $\leftarrow$ conjunction of features

Figure 5: The role and feature data structures.

dexing between objects and concepts.

The function **SAT** takes a set of constraints and a constraint system structure and returns the completed constraint system, if the set of constraints are satisfiable, nil otherwise.

```
function sat(ConstraintsSet,database)
  constrack := set2constrack(ConstraintsSet)
  new-dbase := database
  foreach object in constrack
    new-dbase :=
      eval-object(object,constrack,new-dbase)
    if new-dbase = NIL then EXIT_WITH_FAILURE
  endloop
  EXIT_WITH_SUCCESS
endfunction
```

The function **EVAL-OBJECT** evaluates all the constraints related to a given object. It modifies the constrack, by inserting new constraints, and the database. If a clash is found, the it returns NIL, otherwise it returns the modified database.

```
function eval-object(obj,constrack,dbase)
  new-dbase := dbase
  loop until there are no constraints for obj
    new-dbase :=
      (OR eval-and-constraints
            (obj,constrack,new-dbase)
          ...
          eval-role-constraints
            (obj,constrack,new-dbase))
    if new-dbase = NIL return(NIL)
  endloop
  return(new-dbase)
endfunction
```

There is a **EVAL-...-CONSTRAINTS** function for each type of constraint; functions for $\mathcal{ALC}$ are listed below – the complete listing for $\mathcal{ALCRIFO}$ can be found in [Bresciani *et al.*,1995]. Those functions apply the corresponding rule to the constrack and the database; they return NIL if a clash is found; otherwise they return the modified database. They change the constrack, removing used constraints and possibly adding new constraints.

```
function
  eval-and-constraints(object,constrack,dbase)
  foreach concept of and-constraint for object
    if concept is simple
      then if (concept = bottom) or
                (:instance object (:not concept))
               in database
              then return(NIL)
              else add (:instance object concept)
                     to dbase
      else add (:instance object concept)
              to constrack
  endloop
  return(dbase)
endfunction


function
  eval-or-constraints(object,constrack,dbase)
  foreach concept of or-constraint for object
    new-db := sat(copy(constrack) +
                    (:instance object concept),
                  add-frame(dbase))
    unless new-db=NIL return(new-db)
  endloop
  return(NIL)
endfunction


function
  eval-some-constraints(object,constrack,dbase)
  foreach (:some role concept)
          of some-constraint for object
    new-object := create-variable
    add (:related object new-object role) and
        (:instance new-object concept)
      to constrack
  endloop
  return(dbase)
endfunction


function
  eval-role-constraints(object,constrack,dbase)
  foreach (:related object object1 role)
          of role-constraint for object
    if (:instance object (:all role1 concept))
         in dbase and
       role specializes role1
      then add (:instance object1 concept)
             to constrack
    if (:instance object1 (:all role1 concept))
         in dbase and
       (:inverse role) specializes role1
      then add (:instance object concept)
             to constrack
    add (:related object object1 role) to dbase
  endloop
  return(dbase)
endfunction
```

4 Extensions to the language

The CRACK system currently implements complete reasoning for a larger language than $\mathcal{ALCRIFO}$. Roles and features have been syntactically unified, so that the existential and universal quantifiers can be uniformly used with them. Composition, range and domain have been added for roles. Existential agreement between role paths, singleton variable and fill operators have been added for concepts. The CRACK language can express, for example, the *schema-views* language for querying object-oriented databases – with the exception of cyclic definitions in the schema [Buchheit *et al.*,1994].

We plan to add the possibility of expressing general inclusion statements – as in [Buchheit *et al.*,1993]. Of course, in this case only the subset $\mathcal{ALCR}$ of the language should be used in order to guarantee the termination of the satisfiability procedure. A more ambitious plan is the implementation of $\mathcal{CIF}$ [De Giacomo and Lenzerini,1994] – a description logic in correspondence with an extension of *converse-PDL*, i.e. propositional dynamic logic with the converse operator. De Giacomo

and Lenzerini have proved the decidability of satisfiability in $\mathcal{CIF}$, which includes $\mathcal{ALC}$, functional restrictions, and disjunction, composition, transitive closure, inverse, and identity for roles. In $\mathcal{CIF}$, full terminological axioms can be expressed. Number restrictions, which are present in KRIS and in other systems, can be naively implemented within CRACK, using the same basic modules (like the one for variable renaming) and data structures; however, the efficiency of such a straightforward implementation should be tested. Perhaps, more sophisticated data structures are needed to handle efficiently number restrictions. Two other possible extensions we have taken into consideration are concrete domains [Baader and Hanschke,1991] and plural entities [Franconi,1993].

CRACK supports also an efficient way of management of ABox constraints, i.e. constraints where only individuals are involved. From an abstract point of view, the architecture described so far already supports ABox constraints, since they are uniformly processed as a special type of constraint. Since no variables but only individuals are involved, ABox constraints may be considered static – i.e. they may be stored permanently, being the same for every computation. This avoids redundant applications of the same rules. Thus, a *permanent frame* containing pre-completed ABox constraints is maintained in the constraint system, as the bottom frame. A pre-completion is the processing of the deterministic part of a constraint, which is necessarily equal in every completion of the initial constraint system. The non-deterministic ABox constraints are stored in the entries of the permanent frame, such that they could be processed in later stages. Moreover, dependencies between individuals is maintained, in order to consider only connected parts of the ABox in the processing. Of course, revision of ABox constraints is disallowed without canceling the permanent frame (or at least the connected part of it) and recomputing the whole pre-completion. Preliminary, but not sufficiently complete, tests show that ABox reasoning is quite efficient. We are still working on optimizing this task. It should be noted, however, that the presence of enumerated types – i.e., `one-of` – yields to implicit assertions which can not be easily treated as ABox constraints in the general architecture.

A powerful query language for retrieval has been implemented in CRACK. Apart from the possibility of querying an ABox with the usual open world semantics, it is possible to *efficiently* query an ABox under a closed world assumption. Closed queries make sense because give to the knowledge base a *database* interpretation, which is the preferred one in most real applications. The answer to such a query is computed as if the knowledge base had a closed world semantics; it is equivalent to a query on the knowledge base having the usual open world semantics, where each atomic role and concept is prefixed by the epistemic operator [Donini *et al.*,1992b]. Individuals are indexed with respect to the taxonomic structure of concept terms, in order to optimize the retrieval task. Two different indices sets are pre-computed, using the one for open queries and the other for closed queries. The query is classified and the obtained indexed individuals are checked; thus, the number of instance checking problems to solve is considerably reduced.

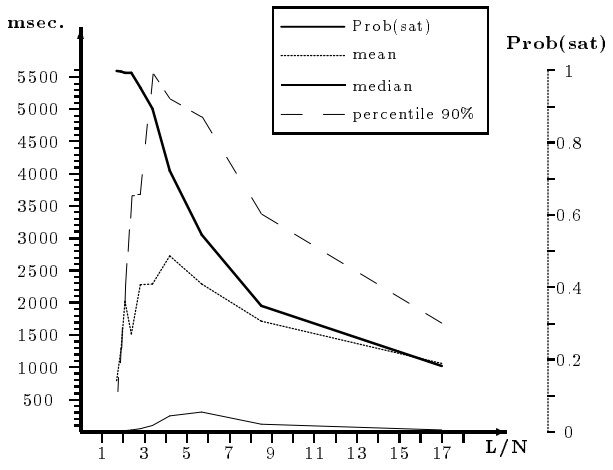


Figure 6: Random propositional 3-SAT transition. N is the number of propositional variables (i.e. primitive concepts); L is the number of `and` branches.

5 Testing Description Logics

In this section we present some preliminary results on testing the hardness of satisfiability in description logics. The first problem in such an effort is to find a reasonable way of generating random knowledge bases to be used in the test. It turns out that it is not even an easily defined task. In fact, it is not clear how a *random* distribution in the multi-dimensional space of well formed formulae of description logics (knowledge bases) could be expressed. Which parameters are the relevant ones? Which structures are representative of the average element of the space? Does such a distribution include both easy and hard problems?

It is possible, like it has been done in [Baader *et al.*,1994], to generate random knowledge bases[1] according to a predefined structure, which should resemble the one of real knowledge bases used in different applications based on description logics. A more general idea would be to generate knowledge bases according to the grammar of the language itself, where each transition has the same probability. However, an extensive testing proved that the generated knowledge bases do not include the real hard cases, and, moreover, they are almost always satisfiable cases (more than 95%, for $\mathcal{ALC}$ knowledge bases).

For these reasons, we looked for particular structures of knowledge bases that could be easily characterized in terms of an order parameter with respect to satisfiability, and showing some neat difficult case in correspon-

[1] In this section, knowledge bases mean just TBoxes.

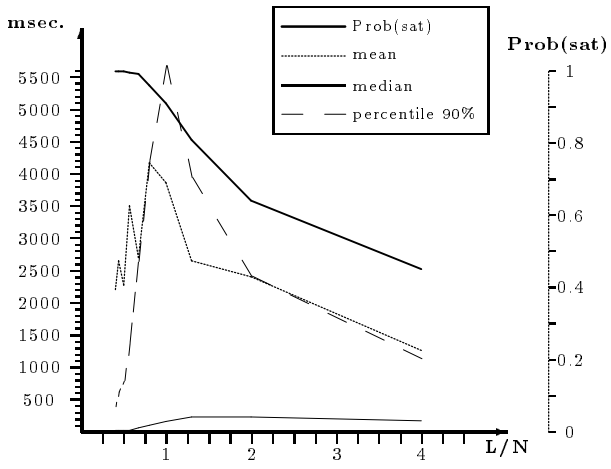


Figure 7: Random modal 2-SAT transition (degree 1). N is the number of propositional variables (i.e. primitive concepts); L is the number of `and` branches.

dence to a phase transition in the satisfiability probability space. [Cheeseman *et al.*,1991] conjecture that all NP-complete problems have at least one order parameter and the hard to solve problems are around a critical value (phase transition) of this order parameter; the phase transition separates the over-constrained from the under-constrained regions of the problem space. There are many papers – e.g. refer to [Gent and Walsh,1994] – giving for propositional logic quite extensive case studies and some theoretical results. Figure 6 points out the random propositional 3-SAT transition obtained with the propositional sub-language of CRACK. Of course, this is not a surprising result, since theoretical studies foresee the easy-hard-easy pattern for satisfiability, within a sat/unsat transition in 3-SAT problems, at around the critical empirical value of 4.2 L/N. The sat/unsat transition appears smooth since we did not consider formulae with more that 20 conjuncts for our limited computing resources[2]. Extensive tests show that average cases in *propositional* 3-SAT are 3 orders of magnitude harder than average cases in *full* $\mathcal{ALC}$ with random grammar: this gives strong evidence that the random grammar based generator does not capture any of the hard cases, where the testing is more meaningful.

It is interesting to notice that in the (*easy*) $\mathcal{ALC}$ cases CRACK is 3 times faster than KRIS, whereas in the *hard* propositional cases (with 4.2 L/N) the ratio is inverted. This is due to the way CRACK handles in general nondeterministic rules (like the disjunction rule: $\rightarrow_{\sqcup}$): CRACK naively generates a fresh new frame at each nondeterministic point of the computation. We are re-implementing the basic resource manager in order to reuse unused frames instead of generating new ones.

Then, we tried to extend the study to the $\mathcal{ALC}$ case – which is in correspondence to the propositional modal logic $\boldsymbol{K}_{(m)}$ [Schild,1991; Halpern and Moses,1985]. It turned out that it is not easy to find a parameter leading to a phase transition; moreover, no results are presented in the literature for satisfiability in PSPACE-complete languages – like $\mathcal{ALC}$. Nonetheless, we have found some interesting cases[3]. For example, figure 7 indicates that even in the modal case a phase transition can be found; formulae are in conjunctive normal form, with L conjuncts and 2 disjuncts, where each literal could be either an existential or a universal expression, whose restriction is a L-2-CNF formula with ground literals. Again, also in this experiment the transition appears smooth. The same transition has been found using both CRACK and KRIS. For the same reasons as above, CRACK is slower than KRIS in the hard cases. More experimental and theoretical studies are needed in order to generalize this preliminary result.

[2]Satisfiability checkers tailored for propositional calculus are much more efficient than our tableaux-based architecture.

Acknowledgments

Many people contributed to the effort of developing the ideas behind CRACK. We would like to thank Francesco M. Donini, Andrea Schaerf, Werner Nutt, Bernhard Hollunder, Roberto Sebastiani for the invaluable discussions we had with them. We thank also the anonymous referees who carefully read previous versions of the paper. All the errors of the paper are, of course, our own.

References

[Baader and Hanschke, 1991] Franz Baader and Philipp Hanschke. A scheme for integrating concrete domains into concept languages. In *Proc. of the 12 th IJCAI*, pages 452–457, Sidney, Australia, 1991.

[Baader and Hollunder, 1991] F. Baader and B. Hollunder. A terminological knowledge representation system with complete inference algorithm. In *Proc. of the Workshop on Processing Declarative Knowledge*, pages 67–86, Kaiserslautern, Germany, 1991.

[Baader *et al.*, 1994] F. Baader, E. Franconi, B. Hollunder, B. Nebel, and H. J. Profitlich. An empirical analysis of optimization techniques for terminological representation systems. *Applied Intelligence*, 4(2):109–132, April 1994. Kluwer Academic Publishers. Special Issue on Knowledge Base Management. Edited by John Mylopoulos.

[Bresciani *et al.*, 1995] Paolo Bresciani, Enrico Franconi, and Sergio Tessaris. Implementing and testing expressive description logics: a preliminary report. Long Version. Submitted, February 1995.

[Buchheit *et al.*, 1993] Martin Buchheit, Francesco M. Donini, and Andrea Schaerf. Decidable reasoning in terminological knowledge representation systems. In *Proc. of the 13 th IJCAI*, pages 704–709, Chambery, France, August 1993.

[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.

[3]Thanks to Roberto Sebastiani.

[Cheeseman *et al.*, 1991] Peter Cheeseman, Bob Kanefsky, and William M. Taylor. Where the really hard problems are. In *Proc. of the 12th IJCAI*, pages 331–337, 1991.

[De Giacomo and Lenzerini, 1994] Giuseppe De Giacomo and Maurizio Lenzerini. Boosting the correspondence between description logics and propositional dynamic logics. In *Proc. of AAAI-94*, 1994.

[Donini *et al.*, 1991] F. M. Donini, M. Lenzerini, D. Nardi, and W. Nutt. Tractable concept languages. In *Proc. of the 12 th IJCAI*, pages 458–465, Sidney, Australia, August 1991.

[Donini *et al.*, 1992a] F. M. Donini, B. Hollunder, M. Lenzerini, A. Marchetti Spaccamela, D. Nardi, and W. Nutt. The complexity of existential quantification in concept languages. *Artificial Intelligence*, 53:309–327, 1992.

[Donini *et al.*, 1992b] F. M. Donini, M. Lenzerini, D. Nardi, A. Schaerf, and W. Nutt. Adding epistemic operators to concept languages. In *Proc. of the 3 rd International Conference on Principles of Knowledge Representation and Reasoning (KR-92)*, pages 342–353, Cambridge, MA, 1992.

[Franconi, 1993] Enrico Franconi. A treatment of plurals and plural quantifications based on a theory of collections. *Minds and Machines*, 3(4):453–474, November 1993. Kluwer Academic Publishers. Special Issue on Knowledge Representation for Natural Language Processing.

[Gent and Walsh, 1994] Ian P. Gent and Toby Walsh. The SAT phase transition. In *Proc. of the 11th ECAI*, pages 105–109, 1994.

[Halpern and Moses, 1985] J. Y. Halpern and Y. Moses. A guide to the modal logics of knowledge and belief: Preliminary draft. In *Proc. of the 9 th IJCAI*, pages 480–490, Los Angeles, CA, 1985.

[Hollunder and Nutt, 1990] B. Hollunder and W. Nutt. Subsumption algorithms for concept languages. Research Report RR-90-04, DFKI, April 1990.

[Hollunder *et al.*, 1990] B. Hollunder, W. Nutt, and M. Schmidt-Schauß. Subsumption algorithms for concept description languages. In *Proc. of the 9 th ECAI*, pages 348–353, Stockholm, Sweden, 1990.

[Schaerf, 1994] Andrea Schaerf. Reasoning with individuals in concept languages. *Data and Knowledge Engineering*, 13(2):141–176, 1994.

[Schild, 1991] Klaus D. Schild. A correspondence theory for terminological logics: preliminary report. In *Proc. of the 12th IJCAI*, pages 466–471, October 1991.

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

From Frames to Concepts: Building a Concept Language within a Frame-based System

T. Kessel, O. Stern, F. Rousselot
ERIC (Equipe de Recherche en Ingenierie de Connaissances), ENSAIS
24, boulevard de la Victoire, 67084 Strasbourg, France
Tel.: (+33) 88 14 47 37
⟨kessel, stern, rousse⟩@steinway.u-strasbg.fr

1 Introduction

Elicitating and acquiring knowledge from texts written in natural language, in order to build a domain ontology and establish (semi-automatically) in the following phase a knowledge-based system, is the overall objective of our research project. This task induces some specific requirements for an underlying knowledge representation and reasoning (KR) system, for instance automatic classification of domain terms, handling of partial individual descriptions and assuring a maximum of coherence in the knowledge base.

We have chosen the paradigm of KL-ONE languages or Description Logics (DL) in combination with a frame language as a base for the implementation of the KR module , because it seems to fulfill most of our requirements. Therefore the developed C3L (**C**onstraint-based **C**oncept **C**lassification **L**anguage) system is implemented in a frame-based language, integrating so most of the powerful features of object-oriented representation [?; ?]. But it still provides the usual expressive power of Description Logics, its inferences and declarative semantics. We would like to bridge the gap between frame languages and description logics by showing common foundations as well as substantial differences, as it might be already indicated in this paper's title.

C3L is a research prototype, implemented in Common Lisp on Macintosh Computers, which will be officially released in summer '95. A C++ version is planned to be incorporated in more conventional software environments.

2 Principles of the C3L-TBox

In the following paragraphs a brief description outlines the main characteristics of C3L:

- The system is based on the "structural" subsumption method as described in [?], because it seems to be more appropriate for object-oriented or frame languages. The algorithm does not compare syntactical constructs, but it follows a rather role-centered approach, which checks the corresponding expressions (stored as frames) role by role.
- An optimization of the subsumption computation could be obtained by means of diverse caching strategies and complex concept structures. Furthermore there exist implementations of three different classification methods (brute force, simple and enhanced traversal), based on the article of [?], allowing us to compare the consequences of each classification depending on a special knowledge base.
- C3L provides a set of usual role operators: **ALL**, **SOME**, **ONE-OF**, **FILLS** and **CARD**. The last one is a combination of the well-known *atleast* and *atmost* operators. A role hierarchy was recently introduced, as well as the specification of numerical values by means of intervals.
- An enhancement of the expressive power is planned towards a "local" complement and disjunction. We follow the rather pragmatic argumentation of [?], giving the user a maximum of expressiveness. The "local" complement is a kind of weakened negation that is exclusively restricted to role operators and therefore cannot be applied to logical connectives (AND, OR) or complete concept expressions. The motivation for such a limited complement is that it can be easily implemented in our system and the (role operators) arguments need not be inverted.
  This enhancement will be implemented as follows: each role operator is attached to a complementary one which stores the arguments. The "local" complement is interpreted as the set of all elements of the domain universe without the explicit arguments given in the role operator. For instance let r be a role and c a concept name, then (**COMP** (**ONE − OF** r *individual* − *list*)) is transformed into the internal representation ($COMP\_ONE - OF$ r *individual* − *list*). All role fillers have to be distinct from the elements of the individual-list. Another example : (**COMP** (**ALL** r c)) is transformed into the internal representation ($COMP\_ALL$ r c). It is interpreted that no role filler is of the concept c (instead of the usual interpretation that there exists at least one role filler which is not c).
  A restricted "local" disjunction allows to give sev-

eral concepts as operands in the **ALL** role operator. For instance (**ALL** r ($c1\,c2\,c3$)) is interpreted as whether all role fillers can be instances of the concepts $c1$, $c2$ or $c3$. In our opinion such an enhancement is sufficient.

A rule-based system is going to be linked to the system's kernel, allowing the modeling and execution of dynamic knowledge. In a short term, it will be restricted to object instances, and in the following phases be attached to a more powerful query language based on the ABox. It might be soon completed by a planning language denoting actions and plans, by an object-oriented database system and a graphical user interface.

3 ERICA — An ABox for C3L

For the representation of individuals and some reasoning about them we have built a subsystem which is known as *ABox* in the literature [?]. Assuming an *open world semantics*, it is possible to assign partial descriptions to individuals. Information about them may be completed, and implicit information which is not initially given in the description may be deduced. The system is domain-independent and can therefore be used in many applications.

The following paragraphs provide a short overview of ERICA:

- *Assert-time deduction* is used to detect inconsistencies as soon as possible. The TBox-contents are additionally used as an integrity constraint for the ABox. This feature is supported by the use of the TBox inference algorithms for subsumption and classification without any change for the ABox inferences. With the integration of large parts of the TBox-syntax into the individual definition part of the ABox-syntax, we obtained an even closer coupling of these subsystems [?]. Concepts, Roles and Individuals can be declared in approximately the same fashion with the same operators.
- A further key objective in the construction of the ABox could be achieved by the definition of a semantics which is common to both, the ABox and the TBox. Combined with the syntax this results in exactly the same expressiveness of the two subsystems. Considering the complementary fact that mutual effects of the subsystems (introduction of new concepts, changes of cardinalities, etc.) are also computed in an appropriate fashion, it can be concluded that the assertional and the terminological component are well balanced [?].
- To compute instantiation relationships between individuals and TBox-concepts (*recognition*) we use a method called abstraction on demand, as described in [?]. This means that only abstractions of terms needed in the current recognition step are really calculated. The necessary classification of concept descriptions is only temporary to avoid the introduction of new concepts into the TBox-taxonomy. In combination with the forward and backward propagation of individual-information we can assure (relative) completeness of the recognition-process [?]. Almost all possible inferences can be drawn at the current state of implementation. In our tests we were able to deduce all consequences on other individuals intended by the individual descriptions.
- To provide a faster access to the once calculated instantiation relationships we use a *semantic indexing* technique. By means of this feature we might achieve a considerable speed-up for the common *retrieval* facilities of the ABox [?].

3.1 Implementation of ERICA

The ABox has been designed in an entirely object-oriented fashion. The implementations of our ABox and TBox were completely separated from each other. Communication between them is only performed by means of the general user interfaces of the subsystems. This design makes it even possible to use several ABoxes with only one underlying TBox for an application [?].

Inside the ABox there exist different kinds of information representation for usage with the user interface, the knowledge base, and for internal use inside the inference algorithms. With such a division into a kernel (internal data representation and inference algorithms) and some interfaces (transformation of the representations and interaction of the subsystems) it might be very easy to attach further components.

4 Topics of interest

Our research interests concern less the theoretical foundations of Description Logics, but applications of these principles in the knowledge acquisition domain or in the field of CAD and configuration problems. Furthermore we obtained a lot of interesting results by choosing a frame language as the base for the implementation. Besides, our position (in the middle of Description Logics and frame-based Languages) fits rather well into a discussion in the wrench scientific community about the limits and common characteristics of object-centered representation and object-oriented programming languages.

The proposition of enhancing C3L by a "local" complement and disjunction might be worth discussing with other researchers, having already implemented similar expressive power or estimated its computational costs. Furthermore we are curious to talk about possible rule-based extensions of DL systems.

Last but not least, we would appreciate to exchange our experiences with other researchers, how to speed up, optimize or validate a DL system. We would be interested in obtaining large or special knowledge bases, already tested or used by other research groups, to validate our system and study its performance [?].

References

[BHNP92] F. Baader, B. Hollunder, B. Nebel, H.-J. Profitlich, et al. An empirical analysis of optimization techniques for terminological representation systems. In *Principles of Knowledge Representation and Reasoning*, San Mateo, CA, 1992.

[DP91] J. Doyle and R. S. Patil. Two theses of knowledge representation: language restrictions, taxonomic classification, and the utility of representation services. In *Artificial Intelligence*, Volume 48, pages 261-297, 1991.

[HKNP94] J. Heinsohn, D. Kudenko, B. Nebel and H.-J. Profitlich. An empirical analysis of terminological representation systems. In *Artificial Intelligence*, Volume 68, pages 367-397, 1994.

[KIND94] C. Kindermann. Personal communication, 1994.

[KMRBK] T. Kessel, H. de Medeiros, F. Rousselot, J. Bürkle and B. Keith. Some useful enhancements of KL-ONE languages to become modeling languages. In M. Willems, D. Fensel, F. v. Harmelen, editors, *Workshop on Modeling Languages for Knowledge-Based Systems*, 1995.

[MACG88] R. MacGregor. A deductive pattern matcher. In *Proceedings of the National Conference on Artificial Intelligence*, AAAI-88, pages 403-408, 1988.

[NEB90] B. Nebel. *Reasoning and Revision in Hybrid Representation Systems*, volume 422 of *Lecture Notes in Artificial Intelligence*. Springer, 1990.

[NL87] B. Nebel and K. Luck. Issues of integration and balancing in hybrid knowledge representation systems. In K. Morik, editor, *GWAI-87*, pages 114-123. Springer, 1987.

[RK94] F. Rousselot, T. Kessel. Implementing subsumption in a declarative frame language to build a knowledge base. In R. Cunis, D. Champeux and H. Kaindl, editors, *ECAI'94*, Workshop on Integrating Object-orientation and Knowledge Representation, 1994.

[ST95.1] O. Stern. *Development of a Recognition Algorithm for C3L*. Internal working paper of ERIC. ENSAIS Strasbourg, 1995.

[ST95.2] O. Stern. *Development of an Assertional Component (ABox) for C3L*. Internal working paper of ERIC, forthcoming. ENSAIS Strasbourg, 1995.

Selecting description logics for real applications

Piet-Hein Speel

Knowledge-Based Systems Group
University of Twente
Enschede, the Netherlands
speel@cs.utwente.nl

Abstract

In order to use knowledge representation systems (KRSs) in realistic, knowledge-intensive applications, a wide gap between theory and practice needs to be bridged. We have introduced a bridge: the KRS selection procedure. The purpose of this procedure is to help users find answers to the general question "*Which KRSs are adequate for realistic, knowledge-intensive applications?*"

This bridging principle has been illustrated in a particular case study. In this study we have focused on the Plinius project which aims at developing a system for semi-automatic extraction of domain knowledge from short texts in a sub-field of material science. From a set of candidate KRSs, namely description logics, we have selected adequate ones for this application, based on the expressive power and (runtime and memory usage) performance properties.

We found that the DLs BACK, BACK++, C-CLASSIC, CLASSIC, KRIS, and LOOM have sufficient expressive power to *approximately* represent the Plinius domain knowledge. From the empirical performance study we did for four DLs, we conclude that all candidate DLs showed insufficient performance. However, except for BACK, the remaining DLs BACK++, CLASSIC, and LOOM showed promising performance.

1 Introduction

When we started the Plinius project, four years ago, one of the purposes was to investigate the reuse of existing systems. In particular, we intended to reuse KRSs and NLP systems.[1] It turned out to be very difficult to make a legitimate choice for an adequate KRS. One of the main reasons is the rather wide gap between theoretical knowledge representation and reasoning issues, on the one hand, and practical requirements of realistic, knowledge-intensive applications, on the other. This gap has arisen in the KRS research field, where most attention has been paid to the development of KRSs. Therefore, research has been mainly focused on the expressiveness of KRS formalisms in relation to their reasoning mechanisms. Relatively little attention has been paid to the use of KRSs in realistic, knowledge-intensive applications, although consensus exists of the necessity of such enterprise (Brachman [1990]; Padgham [1994]).

In this context, we have identified the need for the selection of adequate KRSs for realistic, knowledge-intensive applications. In our attempt to bridge this gap by means of systematic selection, we have introduced the *KRS selection procedure*. Using a list of KRS properties, users should be able to match the requirements of the applications against the characteristics of the candidate KRSs in order to select adequate KRSs. In general, the KRS selection procedure consists of the following steps:

1. Determine the set of candidate KRSs;
2. Determine the relevant KRS properties with respect to the application;
3. Determine the selection criteria, the required values of the properties, imposed by the application;
4. Determine the values of the relevant properties of the candidate KRSs;
5. Determine the subset of *adequate* KRSs by assessing the property values of the candidate KRSs with respect to the selection criteria.

We have illustrated this KRS selection procedure in a detailed case study. In this study, we have focused on the Plinius KBS as a realistic, knowledge-intensive application (Mars et al. [1994]; Van der Vet et al. [1994]). In addition, from a large, varied set of candidate KRSs, we have isolated DLs in general, and six concrete DLs in particular. Finally, we have selected two particular KRS properties, namely the expressive power and the (runtime and memory usage) performance. This case study has been described in detail in Speel [1995].

Before summarizing the conclusions of the Plinius case study in Section 3, we discuss the DL properties expressive power and performance in some detail in Section 2.

[1] Discussion of the reuse of NLP systems is beyond the scope of this thesis.

We close with some general conclusions in Section 4.

2 DL properties

2.1 Expressive power of DLs

Using a slight extension of Baader's formal definition of expressive power (Baader [1990]; Baader [1992]), DL constructions representing specified knowledge of an application domain can be formally verified. However, in many cases the expressive power of DL formalisms is, strictly speaking, insufficient. Nevertheless, sets of DL axioms which "approximately" represent the specified knowledge might be useful. We have considered two possibilities, namely (i) sets of DL axioms which represent more general knowledge which include the required specified knowledge, and (ii) sets of DL axioms representing more restrictive knowledge than specified. Here, we discuss the first possibility in more detail.

Often, sets of DL axioms which represent more general expressions can be created. These axioms represent the specified knowledge, but due to the generality of the constructions, these axioms can be treated in an unintended way (not corresponding to the knowledge specifications). However, if a particular, restricted way of usage would be enforced, exactly the specified knowledge is represented. The restrictions to this particular way of usage are called *representational restrictions*. Due to this restricted way of usage, the set of DL axioms in fact is satisfied by a subset of the usual set of models.

Note that the usage of generalized representations is dangerous, since inferences which seem obvious to users may not be drawn.

For example, consider the knowledge specification expressed by disjunction $t = t_1 \cup t_2$ where t, t_1 and t_2 stand for object classes. We are interested to know whether a DL formalism without concept disjunction ($\mathsf{C}_1 \sqcup \mathsf{C}_2$) is able represent this knowledge specification. The axioms $\mathsf{T}_1 \sqsubseteq \mathsf{T}$ and $\mathsf{T}_2 \sqsubseteq \mathsf{T}$ represent this disjunction if treated in an appropriate way (Brachman et al. [1991]). In general, it might be the case that an instance of T is an instance of neither T_1 nor T_2. If the representational restriction enforces that this case must never occur (which results in the elimination of all models in which $\mathsf{T}^{\mathcal{I}} \supset \mathsf{T}_1^{\mathcal{I}} \cup \mathsf{T}_2^{\mathcal{I}}$), then this set of axioms can be used as a representation of the disjunction. However, note that the system will never deduce that an instance of T is also an instance of T_1 if it cannot be an instance of T_2.

2.2 Performance of DLs

Performance can be measured both analytically and empirically. In the analytical approach, which is widely adopted in the DL field, one usually determines worst-case performance. The standard of worst-case time efficiency is motivated by the desire to develop general-purpose DLs which are of use in all applications, and thus, fast enough even for the most critical ones. However, the core subsumption determination process in DLs with formalisms with reasonable expressive power is intractable and sometimes even undecidable. This is an undesirable result since one can only state that the performance of DLs for applications will never exceed the measures of the worst-case analytical studies (if optimal algorithms are used), while applications probably have more restrictive criteria. Thus, although worst-case performance analysis is useful for DL development (especially the (un)decidability analysis), it seems to be not really useful for DL adequacy assessment with respect to realistic, knowledge-intensive applications. This statement is also motivated by the fact that worst cases often have a theoretical nature which seems not or seldomly to occur in practice.

Peter F. Patel-Schneider[2] has proposed the following compromise. Instead of worst or average cases, *normal cases* can be considered. These normal cases are created by the formulation of particular assumptions which exclude *abnormal* cases. Within these assumptions, then, worst-case analysis can be performed. If adequate assumptions are formulated, the actual performance for realistic, knowledge-intensive applications is closely related to the normal-case performance, and will not exceed the normal-case performance. In CLASSIC, for example, normal-case analysis has been used, based on the assumptions that expanding concepts does not significantly increase the size of a KB, and that the size of role hierarchies are small.

Instead of analytical performance studies, performance can also be studied empirically. Since DLs are software programs, the performance can be measured by running the DLs through a particular compiler on a particular machine with a particular (worst- or average-case) input, and then measure the actual processing time and space taken. Due to the influence of external factors it is a problem to determine the performance of the DL in isolation. However, when applications of DLs are considered, the external influences are also important. For example, if a machine is used with restricted memory capacity, then DLs with efficient memory usage should be preferred. However, in a DL competition, the external influences should be either eliminated or equated as far as possible.

Empirical studies have been worked out in the *DFKI performance experiment* (Heinsohn et al. [1992]; Heinsohn et al. [1994]). In addition to performance measures of some hard cases, the time to load and classify six real terminological KBs (100–400 concepts) in six DLs was measured in this experiment. Moreover, the time to load and classify DFKI synthetic KBs with up to 5000 concepts was measured. The main result is that both size and structure of KBs can have significant impact on the performance.

3 DLs with adequate expressiveness and performance for the Plinius KBS

In this section, we answer the question "*Which DLs have sufficient expressive power and show adequate perfor-*

[2] This information has been obtained via personal communications with Peter F. Patel-Schneider.

mance for the Plinius application? ". In particular, we have focused on the candidate DLs BACK, BACK++, C-CLASSIC, CLASSIC, KRIS, and LOOM.

Expressive power
Based on an extension of Baader's formal definition of expressive power, in which also approximate representations can be formally analyzed, the sufficiency of the expressive power of the candidate DLs has been determined (Speel [1995]). This leads to the following conclusion:

> Under various restrictions, all six DLs have sufficient expressive power to represent the Plinius domain knowledge. In our opinion, the restrictions attached to LOOM are less drastic than those of BACK, BACK++, CLASSIC and KRIS which in turn are less drastic than those of C-CLASSIC. Based on this judgement, for the Plinius application, we prefer the expressive power of LOOM to that of BACK, BACK++, CLASSIC and KRIS, which we prefer to the expressive power of C-CLASSIC.

Note that the DL expressiveness property has influence on other DL properties, such as inference properties. In this work, we have defined the DL expressiveness property in terms of standard, model-theoretic semantics. Given these standard semantics, the inference engine of BACK, BACK++ and LOOM are known to be incomplete. In contrast, KRIS supports a complete inference engine. C-CLASSIC and CLASSIC have an incomplete inference engine with respect to standard model-theoretic semantics, but a complete inference engine with respect to a variant semantics (Borgida & Patel-Schneider [1994]). This means that DLs with sufficient expressive power for a particular application, might have insufficient inference properties.

Performance
In addition, we have focused on the performance property with regard to the candidate DLs BACK, BACK++, CLASSIC, and LOOM in the Plinius scalability experiment (Speel et al. [1995]; Speel [1995]). In order to make comparisons between these candidate DLs possible, we have considered representations expressed in a common subset of DL constructions. Both runtime and memory usage have been measured for storage of various Plinius synthetic terminological and assertional KBs. In addition, runtime has been measured for retrieval of Plinius knowledge. In particular, the performance of DLs for large amounts of knowledge have been considered, with Plinius terminological KBs containing more than 12,500 concepts and roles, and Plinius assertional KBs containing more than 110,000 assertions.

For the storage and retrieval of Plinius domain knowledge on a Sun SPARCsystem 10, we draw the following conclusion with respect to the performance property.

> The combined runtime and memory usage performance of the candidate DLs BACK, BACK++, CLASSIC, and LOOM with respect to the Plinius selection criteria is insufficient.

However, except for BACK, all the candidate DLs showed performance which was rather close to the Plinius selection criteria. A test performed on a more powerful computer will result in better performance measures that might be sufficient with respect to the Plinius selection criteria. In addition, we expect that particular improvements of DL implementations will increase the performance considerably. For example, the performance of BACK++ might increase if manipulations of concrete domain individuals are implemented more efficiently. Moreover, the performance of CLASSIC might increase due to memory usage improvements.

Combination
We conclude that based on the combination of expressiveness and performance properties, BACK, BACK++, CLASSIC and LOOM have insufficient measures to satisfy the Plinius selection criteria. However, the measures of BACK++, CLASSIC and LOOM are promising. BACK cannot be used for the Plinius application due to insufficient performance. We cannot give a final conclusion for C-CLASSIC and KRIS since we have no performance measures.

4 General conclusions

Based on the previous section, we conclude that it is possible to systematically select a set of adequate DLs for the Plinius application with respect to the expressiveness and performance properties. In the remainder of this section we abstract from the Plinius application in order to draw some general conclusions with respect to DLs.

Since most attention in the DL research field has been paid to efficient inferences, we have focused on the expressiveness property in order to arrive at a complementary result. We conclude that knowledge representation in DLs is a complex task. First, knowledge of realistic, knowledge-intensive applications needs to be shoehorned into representations of DLs due to their restrictive expressive power. This has turned out to be a far from trivial task. We have introduced *approximate* representations in order not to simply fail. Second, various philosophical assumptions are used in DLs which require a particular use of DLs. For example, the open-world assumption has consequences for knowledge representation in terminological KBs in order to avoid unwanted situations in assertional KBs. Third, concise and transparent representations are preferred. Last but not least, the resulting DL representations should support effective and efficient reasoning.

Based on the performance property, we conclude that research on efficient reasoning is moving towards average cases. On the one hand, worst cases have been replaced

by normal cases in computational complexity analyses (and applied in CLASSIC). On the other hand, empirical performance studies, namely the DFKI performance experiment and the Plinius scalability experiment, focus on average cases. Besides, from a practical point of view, concrete DLs become more efficient. Due to improved inference algorithms and efficient code, large KBs can be dealt with by most DLs. A few years ago in the DFKI performance experiment, DLs could deal with KBs containing at most 5,000 concepts. At this moment, most current DLs are able to deal with terminological KBs containing over 10,000 concepts and roles, and with assertional KBs containing tens of thousands assertions.

Note that *all* properties relevant to a particular application need to be studied in order to systematically select a set of adequate DLs. In addition to the expressiveness and performance properties, many other properties need to be discussed, such as "philosophical background", "inference engine capacity" and "reliability".

In order to determine the sufficiency of the expressive power of the candidate DLs, we have gained practical experience in representing knowledge in DLs. For the task of knowledge representation for realistic, knowledge-intensive applications, we successfully used an approach similar to that of KRS selection. First, we focused on the application requirements which led to knowledge specifications. These knowledge specifications can be expressed in *knowledge-specification languages*, such as Ontolingua/KIF (Gruber [1992]; Genesereth & Fikes [1992]) and Z (Spivey [1992]). Second, various DL constructions were produced which correspond to these specifications. Third, the knowledge specifications were matched against the DL constructions in order to select the appropriate representations. For a more detailed discussion of this approach, we refer to Speel [1995].

5 Acknowledgments

I would like to thank anyone who helped me creating my Ph.D. thesis. In particular, I would like to thank Nicolaas Mars, Paul van der Vet and Franz Baader for their contribution. In addition, I would like to thank the developers of the systems BACK, BACK++, CLASSIC, C-CLASSIC, KRIS and LOOM for providing me their systems and for their support.

6 References

Franz Baader [1990], "A formal definition for the expressive power of knowledge representation languages," in *Proceedings Ninth European Conference on Artificial Intelligence, Stockholm, 6–10 August 1990*, Luigia Carlucci Aiello, ed., Pitman Publishing, London, 53–58.

Franz Baader [1992], "A formal definition for the expressive power of terminological knowledge representation languages," to appear in Journal of Logic and Computation as a revised version of DFKI Research Report RR-90-05, titled "A formal definition for the expressive power of knowledge representation languages".

Alexander Borgida & Peter F. Patel-Schneider [1994], "A semantics and complete algorithm for subsumption in the CLASSIC description logic," *Journal of Artificial Intelligence Research* 1, 277–308.

Ronald J. Brachman [1990], "The future of knowledge representation: extended abstract," in *Proceedings Eight National Conference on Artificial Intelligence, Boston, MA, 29 July–3 August 1990*, AAAI Press/MIT Press, Menlo Park, CA, 1082–1092.

Ronald J. Brachman, Deborah L. McGuinness, Peter F. Patel-Schneider, Lori Alperin Resnick & Alexander Borgida [1991], "Living with CLASSIC: when and how to use a KL-ONE-like language," in *Principles of semantic networks: explorations in the representation of knowledge*, John F. Sowa, ed., The Morgan Kaufmann series in representation and reasoning, Morgan Kaufmann Publishers, Inc., San Mateo, CA, 401–456.

Michael R. Genesereth & Richard E. Fikes [1992], "Knowledge Interchange Format, Version 3.0, Reference Manual," Computer Science Department, Stanford University, Report Logic-92-1, Stanford, CA.

Thomas R. Gruber [1992], "Ontolingua: a mechanism to support portable ontologies," Knowledge Systems Laboratory, Stanford University, Technical Report KSL 91–66, Palo Alto, CA.

Jochen Heinsohn, Daniel Kudenko, Bernhard Nebel & Hans-Jürgen Profitlich [1992], "An empirical analysis of terminological representation systems," in *Proceedings Tenth National Conference on Artificial Intelligence, July 12–16, 1992*, AAAI Press/MIT Press, Menlo Park, CA, 767–773, also as DFKI Research Report RR-92-16.

Jochen Heinsohn, Daniel Kudenko, Bernhard Nebel & Hans-Jürgen Profitlich [1994], "An empirical analysis of terminological representation systems," *Artificial Intelligence* 68, 367–397.

Nicolaas J.I. Mars, Hidde de Jong, Piet-Hein Speel, Wilco G. ter Stal & Paul E. van der Vet [1994], "Semi-automatic knowledge acquisition in Plinius: an engineering approach," in *Proceedings of the 8th BANFF Knowledge Acquisition for Knowledge-Based Systems Workshop, Alberta, Canada, January 30–February 4, 1994*, Brian R. Gaines & Mark Musen, eds., 4-1–4-15.

Lin Padgham [1994], "Systems vs. theory vs. ...: KR&R research methodologies," in *Proceedings of the Fourth International Conference on Principles of Knowledge Representation and Reasoning, KR'94, May 24–27, 1994, Bonn, Germany*, Jon Doyle, Erik Sandewall & Pietro Torasso, eds., Morgan Kaufmann Publishers, Inc., San

Francisco, CA, 649.

Piet-Hein Speel [1995], "Selecting knowledge representation systems," Ph.D. thesis, University of Twente, Enschede, the Netherlands.

Piet-Hein Speel, Frank van Raalte, Paul E. van der Vet & Nicolaas J.I. Mars [1995], "Scalability of the Performance of Knowledge Representation Systems," in *Towards Very Large Knowledge Bases: Knowledge Building & Knowledge Sharing 1995*, Nicolaas J.I. Mars, eds., IOS Press, Amsterdam, the Netherlands, 173–183.

J.M. Spivey [1992], *The Z Notation: a reference manual; Second edition*, Prentice-Hall International Series in Computer Science; edited by C.A.R. Hoare, Prentice Hall International, UK.

Paul E. van der Vet, Hidde de Jong, Nicolaas J.I. Mars, Piet-Hein Speel & Wilco G. ter Stal [1994], "Plinius intermediate report," University of Twente, Memorandum UT-KBS-94-10, Memoranda Informatica 94-35, Enschede, the Netherlands.