1

Reasoning in Description Logics

F. M. Donini, M. Lenzerini, D. Nardi, A. Schaerf

ABSTRACT.

Description Logics stem from Semantic Networks and Frames. They deal with the representation of structured concepts, and reasoning with them. The structure of a concept is described using a language, called concept language, comprising boolean operators (conjunction, disjunction, negation) and various forms of quantification over the attributes (or, roles) of the given concept. A noticeable difference with other structuring formalisms is that also roles can be given some internal structure, e.g. a role can be expressed as a conjunction of two other roles.

We survey techniques for reasoning, and computational complexity of reasoning problems in Description Logics, referring to four different settings: 1) reasoning with plain concept expressions; 2) reasoning with instances of concepts; 3) reasoning with axioms expressing properties of concepts; 4) reasoning with both instances of concepts and axioms.

In the end of the paper, we mention important aspects of Description Logics not dealt within this survey.

1 Introduction

The idea of developing knowledge representation systems based on a structured representation of knowledge was first pursued with *Semantic Networks* and *Frames.*

Semantic Networks (Quillian 1967, Lehmann 1992) represent knowledge under the form of a labeled directed graph, where nodes denote concepts, and arcs represent the various relations between concepts. As pointed out by Woods (1975) and Brachman (1979), despite their name, early Semantic Networks suffered from the drawback that they did not have clear semantics. Difficulties arise from the fact that in Semantic Networks arcs can represent different kinds of relations between nodes. In particular, Woods

A Great Collection
U. Gnowho and U. Gnowho-Else, eds.

identified two main types of arcs: Those representing *intensional* knowledge and those representing *extensional* knowledge.

A *Frame* (Minsky 1981) usually represents a concept (or a class) and is defined by an identifier, and a number of elements called slots. Each slot corresponds to an attribute that members of the class can have. It contains information about the corresponding attribute, such as default values, restrictions on possible fillers, attached procedures or methods for computing values when needed, and procedures for propagating side effects when the slot is filled. The values of the attributes are either elements of a concrete domain (e.g. integers, strings) or identifiers of other frames. A frame can also represent a single individual, in this case it is related with the attribute *instance-of* to the frame representing the class of which the individual is an instance. Early frame-based systems suffered from the same problem highlighted above for Semantic Networks. In fact, the semantics of frames was not formally defined, and in particular the distinction between the different types of knowledge embedded in a frame system was not clear.

One attempt to providing a formal ground to Semantic Networks and Frames led to the development of the system KL-ONE (Brachman and Schmolze 1985). Subsequently, the research has gone further in this direction by providing systems with an explicit model-theoretic semantics (see for example the system OMEGA, Attardi and Simi 1981). More recently, KR systems based on Description Logics (or *terminological systems*), have been proposed as successors of KL-ONE, with the Tarskian semantics for first order logic. The family of these systems includes KRYPTON (Brachman et al. 1985), NIKL (Kaczmarek et al. 1986), BACK (Quantz and Kindermann 1990), LOOM (MacGregor and Bates 1987), CLASSIC (Borgida et al. 1989), KRIS (Baader and Hollunder 1991), and others (see Rich 1991, Woods and Schmolze 1992).

Description Logics systems are the subject of this survey, which focuses in particular on the foundations of the techniques needed for reasoning in these systems, and the computational properties of the reasoning mechanisms. It is important to note that such foundations have been proved useful not only in Artificial Intelligence, but also in areas such as Databases and Programming Systems, where structured formalisms for the representation of knowledge have been investigated.

Description Logics systems have been used for several application areas, including configuration, natural language processing, and software engineering (see e.g. Wright et al. 1993).

1.1 Representing Knowledge with Description Logics

The core of a KR system based on Description Logics (*DL-system* from now on) is its *concept language*, which can be viewed as a set of constructs for denoting classes and relationships among classes. In concept languages, concepts are used to represent classes as sets of individuals, and roles are binary relations used to specify their properties or attributes.

Concepts are denoted by expressions formed by means of special constructors. For example, the concept expression $\texttt{Parent}\sqcap\texttt{Male}\sqcap\forall\texttt{CHILD.Male}$ denotes the class of fathers (male parents) all of whose children are male. The symbol $\sqcap$ denotes concept conjunction and is interpreted as set intersection. The expression $\forall\texttt{CHILD.Male}$ denotes the set of individuals whose children are all male, thus specifying a property which relates to other individuals through the role CHILD. Expressions of the form $\forall R.C$ are called universal role quantifications. Instead, $\exists\texttt{CHILD.Male}$ is an example of existential role quantification, denoting the set of individuals with a male child. Existential role quantification is sometimes unqualified and written $\exists\texttt{CHILD}$, in which case it denotes the set of individuals with a child. The basic language $\mathcal{FL}^-$, includes concept conjunction, universal role quantification and unqualified existential role quantification. More powerful languages are then defined by adding other constructors to this basic language.

A general characteristic of DL-systems is that the knowledge base is made up of two different components. Informally speaking, the first is a general schema concerning the classes of individuals to be represented, their general properties and mutual relationships, and it is usually referred to as TBox, while the second is a (partial) instantiation of this schema, containing assertions relating either individuals to classes, or individuals to each other, and is usually called ABox. This setting is inherited from KL-ONE (Brachman and Schmolze 1985), and is also shared by several proposals of Database models (see Abrial 1974, Beck et al. 1989, Mylopoulos et al. 1980).

Retrieving information in DL-systems is a deductive process involving both the schema (TBox) and its instantiation (ABox). In fact, the TBox is not just a set of constraints on possible ABoxes, but contains intensional information about classes, and this information is taken into account when answering queries to the knowledge base.

1.2 Goal and Organization of the Paper

One of the main challenges in the study of Description Logics is to identify the fragment of formal logic that both captures the features needed for representation purposes and still allows for the design of effective and efficient reasoning methods. To this aim, several concept languages were considered and their expressiveness and computational complexity were studied, following the work of Brachman and Levesque (1984) who formally studied such properties for a simple language called $\mathcal{FL}^-$ (Frame Language).

Brachman and Levesque (see also Levesque and Brachman 1987) argue that there is a trade-off between the expressivity of a representation language and the difficulty of reasoning on the representations built using that language. In other words, the more expressive the language, the harder the reasoning. They also provide a first example of this trade-off by analyzing the language $\mathcal{FL}^-$ and showing that for such a language the Subsumption problem can be solved in polynomial time, while adding the constructor called *role restriction* to the language makes Subsumption a coNP-hard problem. Their work started a large body of research whose aim is to analyze the complexity of reasoning in different concept languages, relying on standard notions from complexity theory and in particular worst-case complexity analysis.

The goal of this paper is to present the progress made both in the design of reasoning techniques, and in the computational characterization of the reasoning problems arising in DL-systems. For this purpose we consider a rather general framework which takes into account the actual reasoning services that need to be available in DL-systems. As Brachman and Levesque (1984), we start from the computational problems arising when reasoning on concept expressions, but then look at the mechanisms for defining the knowledge base, namely the ABox (which requires reasoning on individuals) and the TBox (which requires to deal with axioms for defining properties of concepts).

One of the assumptions underlying this research is to use worst-case analysis as a measure of the complexity of reasoning tasks. Although such an assumption has sometimes been criticized (see for example Doyle and Patil 1991), we consider the outcomes of this research rather significant. In particular, we outline three main contributions to the area of concept languages and, more generally, to knowledge representation.

First of all, the study of the computational behavior of concept languages has led to a clear understanding of the properties of the language constructors and their interaction. This is not only valuable from a theoretical viewpoint, but gives insight to the designer of deduction procedures, with clear indications of which language constructors are difficult to deal with and general methods to cope with the computational problems. For example, the system CLASSIC (Borgida et al. 1989) was designed taking into account the indications of the complexity analysis in the choice of the constructors included in the language.

Secondly, the complexity results have been obtained by exploiting a general technique for Satisfiability checking in Description Logics. The technique relies on tableaux calculus for first-order logic and is a simplification of the one used by Schmidt-Schauß and Smolka (1991). It has proved extremely useful for studying both the correctness and the complexity of the algorithms. More specifically, it provides an algorithmic framework that is parametric with respect to language constructors. In general, the algorithms for Concept Satisfiability and Subsumption obtained directly from the technique are not practical. However, for some languages they are the only known complete deduction procedures and can lead to actual implementations by the adoption of clever control strategies (see for example Baader et al. 1992).

Finally, the analysis of pathological cases has lead to the discovery of forms of incompleteness of the algorithms developed for implemented systems, and can be used in the definition of suitable test sets for verifying implementations. For example, the comparison of implemented systems described by Baader et al. (1992) benefited from the results of the complexity analysis.

The survey is organized as follows:

- Section 2 provides a formal introduction to concept languages, their use in the construction of the ABox and the TBox, and the associated reasoning services.
- Section 3 deals with reasoning on concept expressions. We start by recalling the notion of structural Subsumption and show that there are inferences that it does not capture. We then present the tableaux-based calculus, and the complexity results for Satisfiability and Subsumption of concept expressions that were obtained by exploiting such a calculus.
- Section 4 deals with reasoning on the ABox. We consider the problem of checking if a set of membership assertions logically implies that a given individual is an instance of a given concept. Further, we introduce an epistemic operator that provides query facilities that go beyond the first-order setting of concept languages. We show that epistemic queries may have both a natural interpretation and good computational properties.
- Section 5 deals with reasoning on the TBox. We discuss different forms of concept definitions available in the TBox, their semantics and the complexity of reasoning with them.
- Section 6 briefly revises the problems related to reasoning in the general setting where both the TBox and the ABox are present.
- Conclusions are drawn in Section 7, where several research issues related to DL-systems and not addressed in this paper, are discussed.

2 Basic notions

In this section we present the basic notions regarding concept languages, knowledge bases built up using concept languages, and reasoning services that must be provided for inferring information from such knowledge bases.

Constructor Name	Syntax	Semantics
concept name	A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
top	$\top$	$\Delta^{\mathcal{I}}$
bottom	$\bot$	$\emptyset$
conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
disjunction ($\mathcal{U}$)	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
negation ($\mathcal{C}$)	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
universal quantification	$\forall R.C$	$\{d_1 \mid \forall d_2 : (d_1, d_2) \in R^{\mathcal{I}} \rightarrow d_2 \in C^{\mathcal{I}}\}$
existential quantification ($\mathcal{E}$)	$\exists R.C$	$\{d_1 \mid \exists d_2 : (d_1, d_2) \in R^{\mathcal{I}} \wedge d_2 \in C^{\mathcal{I}}\}$
number restrictions ($\mathcal{N}$)	$(\geq n\, R)$	$\{d_1 \mid \sharp\{d_2 \mid (d_1, d_2) \in R^{\mathcal{I}}\} \geq n\}$
	$(\leq n\, R)$	$\{d_1 \mid \sharp\{d_2 \mid (d_1, d_2) \in R^{\mathcal{I}}\} \leq n\}$
collection of individuals ($\mathcal{O}$)	$\{a_1, \ldots, a_n\}$	$\{a_1^{\mathcal{I}}, \ldots, a_n^{\mathcal{I}}\}$

TABLE 1 Syntax and semantics of concept-forming constructors.

Constructor Name	Syntax	Semantics
role name	P	$P^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
role conjunction ($\mathcal{R}$)	$Q \sqcap R$	$Q^{\mathcal{I}} \cap R^{\mathcal{I}}$

TABLE 2 Syntax and semantics of the role-forming constructors.

2.1 Syntax and Semantics of Concept Languages

A concept language is composed by the symbols taken from the set of *Concept Names*, *Role Names*, and *Individual Names* (or *Individuals*).

Besides concept, role, and individual names, the alphabet of concept languages includes a number of constructors that permit the formation of *concept expressions* and *role expressions* . The set of constructors for concepts expressions and role expressions considered in this work are listed in Tables 1 and 2, respectively. The description of other constructors can be found in (Baader et al. 1991, Woods and Schmolze 1992, Patel-Schneider and Swartout 1993, Schaerf 1994a).

Concept names are denoted with the letters A, B, role names with P, and individuals names with a, b, possibly with subscripts. Concept expressions and role expressions (or simply *concepts* and *roles*) are denoted respectively with the letters C, D and Q, R.

The various languages differ from one another based on the set of constructors they allow. A concept language $\mathcal{L}$ is uniquely identified by its set of constructors. A concept (resp. role) obtained using the constructors of $\mathcal{L}$ is called an $\mathcal{L}$-*concept* (resp. $\mathcal{L}$-*role*).

For naming concept languages, we adopt the terminology introduced by Schmidt-Schauß and Smolka (1991) and Donini et al. (1991a); following such terminology, an identifying letter is associated to a subset of the constructors, as shown in Tables 1 and 2.

Moreover, after Brachman and Levesque 1984, we call $\mathcal{FL}^-$ the language including universal quantification, conjunction and *unqualified* existential quantification[1] ($\exists R.\top$); and $\mathcal{AL}$ (as named by Schmidt-Schauß and Smolka 1991) the language obtained from $\mathcal{FL}^-$ adding $\top$, $\bot$ and negation of concept names, i.e. negation only of the form $\neg A$.

Following the convention, the superlanguages of $\mathcal{AL}$ will be identified with a string of the form

$$\mathcal{AL}[\mathcal{E}][\mathcal{U}][\mathcal{C}][\mathcal{R}][\mathcal{N}][\mathcal{O}]$$

indicating which constructors are allowed in the language besides those of $\mathcal{AL}$. In a similar way, languages that are superlanguages of $\mathcal{FL}^-$ (but not of $\mathcal{AL}$) will be identified by a string of the form $\mathcal{FL}[\mathcal{E}][\mathcal{U}][\mathcal{R}][\mathcal{N}][\mathcal{O}]^-$.

All the languages we consider in this paper are superlanguages of $\mathcal{FL}^-$ (and most of them are superlanguages of $\mathcal{AL}$).

An *interpretation* $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ consists of a nonempty set $\Delta^\mathcal{I}$ (the *domain*) and a function $\cdot^\mathcal{I}$ (the *interpretation function*) that maps every concept to a subset of $\Delta^\mathcal{I}$, every role to a subset of $\Delta^\mathcal{I} \times \Delta^\mathcal{I}$, and every individual to an element of $\Delta^\mathcal{I}$. Complex concepts and roles are interpreted according to the semantics given in Tables 1 and 2, respectively. In every interpretation different individuals are assumed to denote different elements, i.e. for every pair of individuals a, b, and for every interpretation $\mathcal{I}$, if $a \neq b$ then $a^\mathcal{I} \neq b^\mathcal{I}$. This is called the *Unique Name Assumption* and is usually assumed in database applications.

An interpretation $\mathcal{I}$ is a *model* for a concept C if $C^\mathcal{I}$ is nonempty. A concept is *satisfiable* if it has a model and *unsatisfiable* otherwise. We say that C is *subsumed* by D if $C^\mathcal{I} \subseteq D^\mathcal{I}$ for every interpretation $\mathcal{I}$, and C is *equivalent* to D, written $C \equiv D$, if $C^\mathcal{I} = D^\mathcal{I}$ for every interpretation $\mathcal{I}$.

It is important to observe that, according to the given semantics, the above constructors are not all independent of each other. More precisely, given a language $\mathcal{L}_1$ and a constructor Φ, we say that $\mathcal{L}_1$ *simulates* Φ if for every concept expressible in the language obtained by adding Φ to $\mathcal{L}_1$ there exists an equivalent one expressible in $\mathcal{L}_1$.

For example, in sufficiently powerful languages any concept containing general negation can be rewritten into an equivalent one containing only negation of concept names. In fact, the $\neg$ sign can be "pushed" inside until we obtain only negation of the form $\neg A$. For this reason, the constructor $\mathcal{C}$ can be simulated by the language $\mathcal{ALEU}$. In a similar way, we can also show that $\mathcal{ALC}$ simulates $\mathcal{E}$ and $\mathcal{U}$.

From this point on, we assume (for the sake of simplicity) that in a language $\mathcal{L}$, all the constructors simulated by $\mathcal{L}$ are available in $\mathcal{L}$. This

[1]The constructor $\exists R.\top$, also written $\exists R$, denotes the set of objects d_1 such that there exists an object d_2 related to d_1 by means of the role R. The existential quantification is unqualified in the sense that no condition is stated to d_2 other than its existence.

implies that different strings may identify the same language. For example, $\mathcal{ALEU}$ is the same language as $\mathcal{ALC}$ (and $\mathcal{ALEUC}$).

2.2 DL-systems

A knowledge base built using Description Logics is formed by two components: The *intensional* one, called TBox, and the *extensional* one, called ABox.

2.2.1 Intensional Knowledge

We first focus our attention on the intensional component of a knowledge base, i.e., the TBox. Different proposals for a DL-system provide different mechanisms for expressing the TBox. We list here the types of TBox-statements considered in the literature, different kinds of TBoxes differ from each other by the type of TBox-statements they allow.

Given a language $\mathcal{L}$, a TBox-statement in $\mathcal{L}$ has one of the forms:

(1)	$A \dot{\leq} C$	Primitive Concept Specification,
(2)	$A \doteq C$	Concept Definition,
(3)	$C \dot{\leq} D$	Concept Inclusion,
(4)	$C \doteq D$	Concept Equation

where A is a concept name and C, D are $\mathcal{L}$-concepts.

We call *primitive*-TBox a TBox composed only by statements of type 1. Primitive-TBoxes offer nice computational properties (see e.g. Buchheit et al. 1994b, Buchheit et al. 1994a). For this reason, a mechanism of this kind is employed in other fields such as Object-Oriented Databases and Semantic Data Models (see Calvanese et al. 1994).

We call *simple*-TBox a set of statements of type 1 and 2. Simple-TBoxes are the most commonly considered mechanism, and it is the one employed by almost all the implemented systems (e.g. CLASSIC, BACK).

In a simple-TBox, a concept name appearing on the left-hand side of a concept definition is called a *defined concept*, a concept name appearing on the left-hand side of a primitive concept specification is called a *primitive concept*, and a concept name not appearing on the left-hand side of any TBox-statement is called an *atomic concept*.

An more general kind of TBox, called *free*-TBox, is obtained admitting general concepts in the left-hand side of a TBox-statement. Free TBox-statements are of the form 3 and 4 (and of type 1 and 2 as particular cases). When using free TBoxes, the distinction between defined, primitive and atomic concepts does not make sense anymore.

Different semantics have been proposed for the various TBox mechanisms, depending also on the fact whether cyclic statements are allowed or not within the TBox. We consider below the so-called *descriptive semantics*, which seems to be the most appropriate one, although it is not satisfactory in all cases (Baader 1990, Nebel 1991, De Giacomo and Lenzerini 1994b, Schild 1994). We refer to Section 5 for a discussion on cyclic statements and different semantics for the TBox.

An interpretation $\mathcal{I}$ *satisfies* the statement $A \stackrel{.}{\leq} C$ if $A^{\mathcal{I}} \subseteq C^{\mathcal{I}}$, the statement $A \doteq C$ if $A^{\mathcal{I}} = C^{\mathcal{I}}$, the statement $C \stackrel{.}{\leq} D$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$, and the statement $C \doteq D$ if $C^{\mathcal{I}} = D^{\mathcal{I}}$. An interpretation $\mathcal{I}$ is a *model* for a TBox $\mathcal{T}$ if $\mathcal{I}$ satisfies all the statements in $\mathcal{T}$.

2.2.2 Extensional Knowledge

The construction of the extensional component of a knowledge base, the ABox, is realized by permitting concepts and roles to be used in assertions on individuals. Given a concept language $\mathcal{L}$, an ABox-statement in $\mathcal{L}$ has one of the forms:

$C(a)$	Concept Membership Assertion
$R(a,b)$	Role Membership Assertion

where C is an $\mathcal{L}$-concept, R is an $\mathcal{L}$-role, and a, b are individuals.

The semantics of the above assertions is as follows: If $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ is an interpretation, $C(a)$ is satisfied by $\mathcal{I}$ if $a^{\mathcal{I}} \in C^{\mathcal{I}}$, and $R(a,b)$ is satisfied by $\mathcal{I}$ if $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$.

A set $\mathcal{A}$ of assertions is called an ABox. An interpretation $\mathcal{I}$ is said to be a *model* of $\mathcal{A}$ if every assertion of $\mathcal{A}$ is satisfied by $\mathcal{I}$. $\mathcal{A}$ is said to be *satisfiable* if it admits a model.

2.2.3 Knowledge Bases

Given a concept language $\mathcal{L}$, a *knowledge base* Σ in $\mathcal{L}$ is a pair $\Sigma = \langle \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{T}$ is a TBox in $\mathcal{L}$ and $\mathcal{A}$ is an ABox in $\mathcal{L}$. An interpretation $\mathcal{I}$ is a *model* for Σ if it is both a model for $\mathcal{T}$ and a model for $\mathcal{A}$. A knowledge base Σ logically implies α, where α is either a TBox- or an ABox-statement, written $\Sigma \models \alpha$, if α is true in every model of Σ.

2.3 Reasoning Services

There are several reasoning services to be provided by a DL-system. As we already said in Section 1, we concentrate on the following basic ones, which we can now formally define.

Definition 2.1 Given a knowledge base $\Sigma = \langle \mathcal{T}, \mathcal{A} \rangle$, two concepts C and D, and an individual a, we call:

- *Concept Satisfiability*, written as $\Sigma \not\models C \equiv \bot$, the problem of checking whether C is satisfiable w.r.t. Σ, i.e. whether there exists a model $\mathcal{I}$ of Σ such that $C^{\mathcal{I}} \neq \emptyset$;
- *Subsumption*, written as $\Sigma \models C \sqsubseteq D$, the problem of checking whether C is subsumed by D w.r.t. Σ, i.e. whether $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ in every model $\mathcal{I}$ of Σ;

- *Consistency*, written as $\Sigma \not\models$, the problem of checking whether Σ is satisfiable, i.e. whether it has a model;
- *Instance Checking*, written as $\Sigma \models C(a)$, the problem of checking whether the assertion $C(a)$ is satisfied in every model of Σ;

The importance of Concept Satisfiability and Subsumption has been stressed by several authors (see for example Nebel 1990a). In particular, Subsumption is central to the more complex service of *Classification*: Given a concept C and a TBox $\mathcal{T}$, for all concepts D of $\mathcal{T}$ determine whether D subsumes C, or D is subsumed by C (intuitively, this amounts to finding the "right place" for C in the taxonomy implicitly present in $\mathcal{T}$). Consistency is used for verifying whether the information contained in a knowledge base is coherent. Finally, Instance Checking is used to check whether the knowledge base entails that an individual is an instance of a concept and it can be considered the central reasoning task for retrieving information on individuals in the knowledge-base (Donini et al. 1994b). As we already mentioned, Instance Checking is a also basic tool for more complex reasoning problems. For example, *Retrieval* (or *Query Answering*): can be formulated in the following way: "given a knowledge base Σ and a concept D, find the set $\{a \mid \Sigma \models D(a)\}$", and it can be performed simply by iterating Instance Checking for all the individuals in Σ.

Notice that for languages with the constructor $\mathcal{C}$, for expressing the negation of concepts, we have that $\Sigma \models C \sqsubseteq D$ if and only if $\Sigma \models C \sqcap \neg D \equiv \bot$ and $\Sigma \models C(a)$ if and only if $\Sigma \cup \{\neg C(a)\}\rangle \models$. Therefore, Subsumption can be reduced to Concept Satisfiability and Instance Checking to Consistency.

3 Reasoning with Concept Expressions

The goal of this section is to overview the reasoning techniques and the complexity results for reasoning with concept expressions. In particular, we consider Subsumption and Concept Satisfiability in the case in which the knowledge base is empty. In order to simplify the notation, in this case, we write those problems as $C \not\equiv \bot$ and $C \sqsubseteq D$ instead of $\langle\emptyset,\emptyset\rangle \not\models C \equiv \bot$ and $\langle\emptyset,\emptyset\rangle \models C \sqsubseteq D$.

We first describe the approach based on structural comparison, and subsequently we discuss the approach based on tableaux. Finally we discuss the complexity results for various languages and we highlight the corresponding sources of complexity.

We concentrate here on the so-called $\mathcal{AL}$-languages, i.e. languages in the family $\mathcal{AL}[\mathcal{U}][\mathcal{E}][\mathcal{C}][\mathcal{N}][\mathcal{R}]$. Complexity results for Satisfiability and Subsumption including other constructors has been found, among other, by Donini et al. (1991b), Patel-Schneider (1989), Schild (1988), Schmidt-Schauß (1989).

3.1 Structural Comparison: Syntax-Based Reasoning

The first studies of the computational properties of terminological languages are due to Brachman and Levesque (1984), Patel-Schneider et al. (1984), and Nebel (1988, 1990b). The algorithms they give for computing Subsumption are based on structural comparisons between concept expressions.

At the heart of structural comparison is the idea that if the two concept expressions to be compared are made of subexpressions, one can compare separately one subexpression of a concept with all those of the others. We sketch the structural algorithm of Brachman and Levesque (1984, 1987) for Subsumption checking in the language $\mathcal{FL}^-$.

The algorithm works in two phases: first, concepts are rewritten in a normal form, then their structures are compared. More specifically,

1. All nested conjunctions are flattened, i.e. $A \sqcap (B \sqcap C) \rightarrow A \sqcap B \sqcap C$. All conjunctions of universal quantifications are factorized, i.e. $\forall P.C \sqcap \forall P.D \rightarrow \forall P.(C \sqcap D)$. The rewritten concepts are logically equivalent to the previous ones, hence Subsumption is preserved by this transformation.
2. Let $C = C_1 \sqcap \cdots \sqcap C_m$ and $D = D_1 \sqcap \cdots \sqcap D_n$. Then D subsumes C if and only if for all D_i:
   a. if D_i is either an atomic concept, or is a concept of the form $\exists P$, then there exists a C_j such that $D_i = C_j$;
   b. if D_i is a concept of the form $\forall P.D'$, then there exists a C_j of the form $\forall P.C'$ (same atomic role P) such that D' subsumes C'.

By induction on the nesting of $\forall$-quantifiers, one can prove that the complexity of the above algorithm is $O(|C| \times |D|)$ (Levesque and Brachman 1987). If subexpressions of each concept are ordered (e.g. lexicographically), then it can be shown that the complexity is only linear, so the dominant factor becomes the complexity of ordering concepts (Salomone 1992). The same algorithm can be extended up to the language $\mathcal{ALN}$, with the same complexity.

Similar normalize-and-compare algorithms are actually used in the systems CLASSIC (Borgida and Patel-Schneider 1994), BACK (Quantz and Kindermann 1990), and LOOM (MacGregor 1994). However, all these algorithms are incomplete, if a standard semantics of concepts is assumed. This is due to interactions between constructors, which are not taken into account by structural comparisons. An obvious example is the concept $A \sqcup \neg A$, which is equivalent to $\top$ and therefore subsumes every concept. However, there are more subtle cases of interaction. We highlight this through two examples.

Example 3.1 Consider the language $\mathcal{FL}$, obtained from $\mathcal{FL}^-$ by adding *role restriction*[2] as the role-forming constructor.

One can verify that the following Subsumption holds:

$$\forall(P\,|\,\exists Q).A \sqcap \forall(P\,|\,\forall Q.B).A \sqsubseteq \forall P.A$$

even if the concept $\forall P.A$ subsumes neither of the two conjuncts $\forall(P\,|\,\exists Q).A$ and $\forall(P\,|\,\forall Q.B).A$, if they are considered separately. This is because their conjunction is equivalent to $\forall(P\,|\,(\exists Q \sqcup \forall Q.B)).A$. But the concept $\exists Q \sqcup \forall Q.B$ is equivalent to $\top$ hence the left-hand-side concept is in fact equivalent to $\forall P.A$. Hence structural comparison would fail to recognize this subsumption relation.

Subsumption in $\mathcal{FL}$ is proved coNP-hard by Brachman and Levesque (1984), and the result has been recently sharpened by Donini et al. (1995), where Subsumption in $\mathcal{FL}$ is proved PSPACE-complete.

Example 3.2 Consider the language of the BACK system (called $\mathcal{ALNR}$ following our convention), obtained from $\mathcal{FL}^-$ by adding role intersection, number restrictions and negation of atomic concepts:

One can verify that the following Subsumption holds:

$$\begin{array}{l}(\geq 2\,(\mathtt{CHILD} \sqcap \mathtt{SON})) \sqcap (\geq 2\,(\mathtt{CHILD} \sqcap \mathtt{DAUGHTER}))\sqcap \\ (\forall \mathtt{SON}.\mathtt{Male}) \sqcap (\forall \mathtt{DAUGHTER}.\neg\mathtt{Male})\end{array} \quad \sqsubseteq \quad (\geq 4\,\mathtt{CHILD})$$

although the concept $(\geq 4\,\mathtt{CHILD})$ subsumes neither of the four left-hand-side conjuncts, if they are considered separately. Therefore, also i this case structural comparison would not reveal the subsumption relation. The intuitive reason is that the two value restrictions $\forall\mathtt{SON}.\mathtt{Male}$ and $\forall\mathtt{DAUGHTER}.\neg\mathtt{Male}$ on the subroles $\mathtt{CHILD} \sqcap \mathtt{SON}$ and $\mathtt{CHILD} \sqcap \mathtt{DAUGHTER}$ of the role $\mathtt{CHILD}$ force a case analysis.

Subsumption between BACK-concepts is proved coNP-hard by Nebel (1988), and the result has been sharpened by Donini et al. (1991a), where Subsumption is proved PSPACE-complete.

These examples show that structural Subsumption is incomplete w.r.t. the first-order semantics, if particular combinations of constructors are allowed in the language. Of course, structural algorithms can be enhanced by taking into account also particular combinations of constructors (e.g., see the algorithms proposed by Patel-Schneider 1984, Nebel 1990b). However, the difficult part for structural algorithms is to prove completeness (i.e. prove that enough comparisons have been considered to capture all possible interactions).

We now describe a different approach for developing Subsumption algorithms: We start from the well-known tableaux calculus for first-order

[2] Given a role R and a concept C, the role restriction of R to C is usually denoted as $R\,|\,C$ and its semantics is $(R\,|\,C)^{\mathcal{I}} = \{(d_1, d_2) \mid (d_1, d_2) \in R^{\mathcal{I}} \wedge d_2 \in C^{\mathcal{I}}\}$.

logic, and refine it operations to get a reasoning technique with optimal worst-case behavior.

3.2 Constraint Systems: Semantics-Based Reasoning

The reasoning technique we present is based on a notational variant of the first-order tableaux calculus. The bulk of the work is the adaptation of the calculus to complexity analyses. In fact, to obtain optimal algorithms the calculus is not blindly applied—as it is in a general-purpose first-order theorem prover. Instead, the structures of tableaux obtained when reasoning within a given concept language are carefully analyzed, and redundant verifications in the tableaux are eliminated in order to give a strict upper bound on the complexity of the method.

We give an outline how one can devise a reasoning technique based on the tableaux calculus (for a general introduction to tableaux in first-order logic see, e.g., Bell and Machover 1977, Fitting 1990). The key idea is: To check whether a given formula F is a logical consequence of a given theory T, try to build with suitable propagation rules the most generic model of T where F is false. If you succeed, the answer is NO (since F is not a logical consequence of T, hence the method is complete); if you fail, the answer is YES (since looking for the most generic model ensures that in no possible model of T is F false, hence F is indeed a logical consequence of T, hence the method is sound). The propagation rules come straightforwardly from the semantics of constructors. We give an example of how the semantics of a given constructor can be used to devise a propagation rule.

Consider the constructor $\exists R.C$. If in a given interpretation $\mathcal{I}$, whose domain contains the element a, we have that $a \in (\exists R.C)^{\mathcal{I}}$, then from the semantics we know that there must be an element b (not necessarily distinct from a) such that $(a, b) \in R^{\mathcal{I}}$, and $b \in C^{\mathcal{I}}$. Since this must be true for any interpretation, we can abstract from interpretations and their elements, and say that if in a generic interpretation we have a generic element x that is in the interpretation of the concept $\exists R.C$ (denote this by $x\colon \exists R.C$) then there must be a generic element y such that x and y are in relation through R (denote it xRy) and y belongs to the interpretation of C (denoted as $y\colon C$).

Suppose now we want to construct a generic interpretation S such that the set corresponding to the concept $\exists R.C$ contains at least one element. We can state this initial requirement as the constraint $x : \exists R.C$. Following the semantics, we know that S must have a generic element y such that the constraints xRy and $y : C$ must hold, hence we can add these new constraints to S, knowing that if S will ever satisfy them then it will also satisfy the first constraint. These considerations lead to the following propagation rule:

$S \ \rightarrow_{\exists} \ \{xRy, \ y\colon C\} \cup S$

if $x : \exists R.C$ is in S, there is no z such that both xRz and $z\colon C$ are in S, and y is a new variable

Following Schmidt-Schauß and Smolka (1991), we call the generic interpretation being built a *constraint system*, as it is a set of constraints that incrementally specify the properties that the generic interpretation being built must satisfy.

There is an interesting property of first-order interpretations of concept languages. Since concepts and roles are interpreted as unary and binary predicates, an interpretation can be viewed as a labeled directed graph. Each node is a variable (a generic element of the interpretation), and each arc is a relationship among variables that must hold in the interpretation. Labels on arcs are primitive roles, and labels on nodes are sets of primitive concepts and negations of primitive concepts. This property is not sufficient to allow any result on graph problems to be transferred to concept languages: a graph problem has always a given graph as input, whereas here we must find a graph—if any—that satisfies the concept. Nevertheless, the property is useful to intuitively explain some structural properties of constraint systems within a given concept language, as exemplified in subsection 3.3.1.

3.3 Case Study: A Calculus for $\mathcal{ALCNR}$

As an example of how to devise a tableaux-based deduction method for reasoning in concept expressions, we provide a calculus for the expressive language $\mathcal{ALCNR}$.

We introduce an alphabet of variable symbols $\mathcal{V}$, whose elements are denoted by the letters x, y, z. A *constraint* is a syntactic entity of one of the forms:

$x\colon C$	Concept constraint
xPy	Role constraint
$x \neq y$	Inequalityconstraint

where C is a concept and P is a role name. Concepts are assumed to be *simple*, i.e., the only complements they contain are of the form $\neg A$, where A is a concept name. Simple concepts are the analogous of first-order formulae in negation normal form. Arbitrary $\mathcal{ALCNR}$-concepts can be rewritten into equivalent simple concepts in linear time (Donini et al. 1991a). A constraint system is a finite nonempty set of constraints.

We extend the interpretation of concepts to constraint systems as follows. Given an interpretation $\mathcal{I}$, we define an $\mathcal{I}$*-assignment* α as a function that maps every variable of $\mathcal{V}$ to an element of $\Delta^{\mathcal{I}}$.

A pair $(\mathcal{I}, \alpha)$ *satisfies* the concept constraint $x : C$ if $\alpha(x) \in C^{\mathcal{I}}$, the role constraint xPy if $(\alpha(x), \alpha(y)) \in P^{\mathcal{I}}$, the inequality constraint $x \neq y$ if $\alpha(x) \neq \alpha(y)$. A constraint system S is *satisfiable* if there is a pair $(\mathcal{I}, \alpha)$ that satisfies every constraint in S.

Obviously, a concept C is satisfiable if and only if the constraint system $\{x : C\}$ is satisfiable. In order to check a constraint system S for Satisfiability, our technique adds constraints to S until either an evident contradiction is generated or an interpretation satisfying it can be obtained from the resulting system. Constraints are added on the basis of a suitable set of so-called *propagation rules*.

Before providing the rules, we need some additional definitions, which are evident considering variables in a constraint system as nodes in a graph, and constraints xPy as labeled arcs in this graph. Let S be a constraint system and $R = P_1 \sqcap \ldots \sqcap P_k$ $(k \geq 1)$ be a role. We say that y is an *R-successor of x in S* if $xP_1y, \ldots, xP_ky$ are in S. We say that y is a *direct successor of x in S* if for some role R, y is an R-successor of x. We call direct predecessor the inverse relation of direct successor. If S is clear from the context we simply say that y is an R-successor or a direct successor of y (or direct predecessor). Moreover, we call *successor* the transitive closure of the relation "direct successor", and we call *predecessor* its inverse.

We denote by $S[y/z]$ the constraint system obtained from S by replacing each occurrence of the variable y by the variable z.

We say that x and y are *separated in S* if the constraint $x \neq y$ is in S.

The *propagation rules* of the Calculus for $\mathcal{ALCNR}$ are:

1. $S \rightarrow_{\sqcap} \{x : C_1,\ x : C_2\} \cup S$

   if 1. $x : C_1 \sqcap C_2$ is in S,
      2. $x : C_1$ and $x : C_2$ are not both in S
2. $S \rightarrow_{\sqcup} \{x : D\} \cup S$

   if 1. $x : C_1 \sqcup C_2$ is in S,
      2. neither $x : C_1$ nor $x : C_2$ is in S,
      3. $D = C_1$ or $D = C_2$
3. $S \rightarrow_{\forall} \{y : C\} \cup S$

   if 1. $x : \forall R.C$ is in S,
      2. t is an R-successor of x,
      3. $y : C$ is not in S

4. $S \rightarrow_{\exists} \{xP_1y, \ldots, xP_ky,\ y\colon C\} \cup S$

   if 1. $x\colon \exists R.C$ is in S,
   2. $R = P_1 \sqcap \ldots \sqcap P_k$,
   3. y is a new variable,
   4. there is no z such that z is an R-successor of x in S and $z\colon C$ is in S

5. $S \rightarrow_{\geq} \{xP_1y_i, \ldots, xP_ky_i \mid i \in 1..n\} \cup \{y_i \neq y_j \mid i,j \in 1..n, i \neq j\} \cup S$

   if 1. $x\colon (\geq n\, R)$ 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 x in S

6. $S \rightarrow_{\leq} S[y/z]$

   if 1. $x\colon (\leq n\, R)$ is in S,
   2. x has more than n R-successors in S,
   3. y, z are two R-successors of x which are not separated

We call the rules $\rightarrow_{\sqcup}$ and $\rightarrow_{\leq}$ *nondeterministic* rules, since they can be applied in different ways to the same constraint system. All the other rules are called *deterministic* rules. Moreover, we call the rules $\rightarrow_{\exists}$ and $\rightarrow_{\geq}$ *generating* rules, since they introduce new variables in the constraint system. All other rules are called *nongenerating* ones.

One can verify that rules are always applied to a system S because of the presence in S of a given constraint $x : C$ (condition 1). When no confusion arises, we say that a rule is *applied to* the constraint $x\colon C$ or to the variable x (instead of saying that it is applied to the constraint system S).

While building a constraint system, we can look for evident contradictions to see if the constraint system is not satisfiable. We call these contradictions clashes. A *clash* is a constraint system having one of the following forms:

- $\{x\colon \bot\}$
- $\{x\colon A,\ x\colon \neg A\}$, where A is a concept name.
- $\{x\colon (\leq n\, R)\} \cup \{xP_1y_i, \ldots, xP_ky_i \mid i \in 1..n+1\}$
  $\cup \{y_i \neq y_j \mid i,j \in 1..n+1, i \neq j\}$,

  where $R = P_1 \sqcap \ldots \sqcap P_k$.

A clash is evidently an unsatisfiable constraint system, hence any constraint system containing a clash is obviously unsatisfiable. The last case of clash deals with the situation in which a variable has an at-most restriction and a set of R-successors that cannot be identified because they are pairwise separated.

The purpose of the calculus is to generate a constraint system, and look for the presence of clashes inside. The three important properties that must be proved for such a calculus are soundness, completeness and termination. We sketch each one in turn.

Soundness. Intuitively, we prove that the calculus does not add unnecessary contradictions. That is, deterministic rules always preserve the Satisfiability of a constraint system, and nondeterministic rules have always a choice of application that preserves Satisfiability. The proof (see Donini et al. 1995) resembles soundness proofs for tableaux methods (e.g., Fitting 1990, Lemma 6.3.2). The only non-standard constructors are number restrictions. Without number restrictions, the generating rules could operate by introducing new constants, all different from one another. However, this would not preserve Satisfiability, e.g. in the following example:

$$x\colon \exists P.C \sqcap \exists P.D \sqcap (\leq 1\, P)$$

after having applied two times the $\rightarrow_\sqcap$-rule, and two times the $\rightarrow_\exists$-rule, the resulting constraint system contains the constraints $\{xPy, xPz, x\colon(\leq 1\, P)\}$ (we omit the others), and is unsatisfiable if y, z are different constants. Using variables and $\mathcal{I}$-assignments instead, the resulting constraint system is still satisfied by an $\mathcal{I}$-assignment which maps y and z into the same element.

Termination. A constraint system is *complete* if no propagation rule applies to it. A complete system derived from a system S is also called a *completion* of S. Completions are reached when there is no infinite chain of applications of rules. Intuitively, this can be proved by using the following argument: all rules but $\rightarrow_\forall$ and $\rightarrow_\leq$ are never applied twice on the same constraint; these two rules in turn are never applied to a variable x more times than the number of the direct successors of x, which is bounded by the length of a concept; finally, each rule application to a constraint $y\colon C$ adds constraints $z\colon D$ such that D is a strict subexpression of C.

Completeness. If S is a completion of $\{x\colon C\}$ and S contains no clash, then it is always possible to construct an interpretation for C on the basis of S, such that $C^\mathcal{I}$ is nonempty. The proof is a straightforward induction on the length of the concepts involved in each constraint.

We now analyze the complexity of such a calculus, and how we can refine it to get an optimal upper bound.

3.3.1 Traces and Completions

The notion of constraint system was introduced by Schmidt-Schauß and Smolka (1991), who analyze the language $\mathcal{ALC}$. The key observation made by Schmidt-Schauß and Smolka was that in $\mathcal{ALC}$ a constraint system can be partitioned into *traces*. Recalling that constraint systems can be regarded as graphs, traces correspond to paths in the graph that starts from the variable x of the initial constraint system $\{x\colon C\}$.

The basic property of traces in $\mathcal{ALC}$ is that a constraint system contains a contradiction if and only if it contains a trace that contains a contradiction. We will call this property *independency of traces*. Traces are independent if the graph associated with a constraint system is a tree (with x as root). Whenever in a language $\mathcal{L}$ traces are independent, constraint systems can be checked for Satisfiability just by looking at their traces, one at a time. If each trace has polynomial size with respect to the concept C to be checked, we have a PSPACE algorithm for Satisfiability in $\mathcal{L}$.

The constraint systems built by the above propagation rules are not exactly trees, since role intersection forces the introduction of many arcs between two variable nodes (see the generating rules $\rightarrow_{\exists}$, $\rightarrow_{\geq}$). However, if we consider the successor relation between variables it indeed forms a tree, because no variable is a direct successor of more than one variable. Moreover, the depth of such a tree is bounded by the number of nested quantifiers in the concept (counting also the "at-least" restrictions as existential quantifiers). Hence, generating depth-first such a tree-shaped constraint system needs just polynomial space, since the depth is bounded by the length of the concept. Such a depth-first construction can be achieved by using the following strategy for the application of rules:

1. apply a generating rule only if no nongenerating rule is applicable;
2. when a generating rule is applicable to more than one concept constraint, choose the concept constraint with the most recently generated variable.

In order to actually use polynomial space, one should also discard constraints $y : D$, yPz, $y \not\doteq z$ whenever there is no rule applicable to any successor and any predecessor of y. Since this would also modify the conditions of application of rules (namely, 1.2, 2.2, 3.3, 4.4, 5.4), we do not delve into the details of such a modification.

Note that till now we devised a calculus working in polynomial space which is *nondeterministic*; a deterministic version would require to explore in turn all possible applications of the nondeterministic rules, looking for choices that do not lead to a clash. However, also such an exploration can be done depth-first, using backtracking. The number of backtracking points that need to be memorized is no more than the number of constraints of the form $y : D_1 \sqcup D_2$, and $y : (\leq n\, R)$, and this number can be proved to be polynomially bounded by the length of the initial concept. Hence, Satisfiability of $\mathcal{ALCNR}$-concepts can be decided in polynomial space.

Noting that Subsumption in $\mathcal{ALCNR}$ can be rephrased as Satisfiability, we can conclude the main theorem of this subsection.

Theorem 3.3 *Satisfiability and Subsumption in $\mathcal{ALCNR}$ can be decided with polynomial space.*

As to the time needed by this calculus, both the nondeterministic version and the deterministic one work in exponential time; to prove that such a calculus is really optimal, we need a reduction from a PSPACE-complete problem. We consider this in the next subsection.

3.4 Sources of Complexity

Note that the deterministic version of the calculus for $\mathcal{ALCNR}$ can be seen as exploring an AND-OR tree, where an AND-branching corresponds to the (independent) check of all successors of a variable, while an OR-branching corresponds to the choices of application of a nondeterministic rule.

Realizing that, one can see that the exponential-time behavior of the calculus is due to two independent origins: The AND-branching, responsible for the exponential size of a single constraint system, and the OR-branching, responsible for the exponential number of different constraint systems. We call these two different combinatorial explosions *sources of complexity.*

The OR-branching is due to the presence of disjunctive constructors. The obvious disjunctive constructor is $\sqcup$, hence $\mathcal{ALU}$ is a good sublanguage to study this source of complexity. We have seen also that disjunction can be introduced by combining role restrictions and universal quantification in Example 3.1, and by combining number restrictions and role intersection in Example 3.2. This source of complexity is the same that makes propositional satisfiability NP-hard: in fact, Satisfiability in $\mathcal{ALU}$ can be trivially proved NP-hard by rewriting propositional letters as atomic concepts, $\wedge$ as $\sqcap$, and $\vee$ as $\sqcup$. Many proofs of coNP-hardness of Subsumption were found exploiting this source of complexity (Levesque and Brachman 1987, Nebel 1988), by reducing an NP-hard problem to non-Subsumption.

The AND-branching is more subtle. Its exponentiality is due to the interplay of qualified existential and universal quantifiers, hence $\mathcal{ALE}$ is now a minimal sublanguage of $\mathcal{ALCNR}$ with these features. One can see the effects of this source of complexity by expanding the constraint system $\{x : C\}$, when C is the following concept (taken from Schmidt-Schauß and Smolka 1991):

$$\begin{array}{l}
\exists P._1 C_{11} \sqcap \\
\exists P_1 . C_{12} \sqcap \\
\forall P_1 .(\exists P_2 . C_{21} \sqcap \\
\qquad \exists P_2 . C_{22} \sqcap \\
\qquad \forall P_2 .(\ldots(\exists P_n . C_{n1} \sqcap \\
\qquad\qquad\quad \exists P_n . C_{n2} \sqcap \\
\qquad\qquad\quad \forall P_n . D)\ldots))
\end{array}$$

A clash may be found independently in each trace, i.e. in each branch of the tree of variables given by the successor relation. This source of complexity causes an exponential number of possible *refutations* to be searched through (each refutation being a clash in a trace). This source of complexity is not evident in propositional calculus, but a similar problem appears in predicate calculus, where the interplay of existential and universal quantifiers may lead to infinite models. A proof of coNP-hardness of Satisfiability (so, NP-hardness of Subsumption) has been found using this source of complexity described by Donini et al. (1992a).

It is interesting to observe that, analogously to Examples 3.1,3.2, also this source of complexity can appear without the explicit presence of qualified existential and universal quantification. In fact, the pattern $\exists P.C_1 \sqcap \exists P.C_2 \sqcap \forall P.D$ can also be simulated by role intersection as

$$\exists(P \sqcap Q_1) \sqcap \forall Q_1.C_1 \sqcap \exists(P \sqcap Q_2) \sqcap \forall Q_2.C_2 \sqcap \forall P.D$$

This simulation was used to prove coNP-hardness of Satisfiability of $\mathcal{ALR}$ by Donini et al. (1991a).

In a language containing both sources of complexity, one might expect to code any problem involving the exploration of polynomial-depth AND-OR graphs. The computational analog of such graphs is the class APTIME (problems solved in polynomial time by an alternating Turing machine) which is equivalent to PSPACE (e.g., see Johnson 1990, pg.98). A well-known PSPACE-complete problem is Quantified Boolean Formulae (QBF): Given a sentence of the form

$$(Q_1X_1)(Q_2X_2)\cdots(Q_nX_n)[F(X_1,\ldots,X_n)]$$

where each Q_i is a quantifier (either $\forall$ or $\exists$) and $F(X_1,\ldots,X_n)$ is a boolean formula with boolean variables $X_1,\ldots,X_n$, decide whether the sentence is true. The problem remains PSPACE-complete if F is in 3CNF, i.e., conjunctive normal form with at most three literals per clause. We call prefix the string of quantifiers, and matrix of the QBF the formula F.

This problem can be encoded in an AND-OR graph, using AND-nodes to encode $\forall$-quantifiers, and OR-nodes for $\exists$-quantifiers. We use this analogy to illustrate the reduction, which was first published by Schmidt-Schauß and Smolka (1991) in a slightly different form.

Without loss of generality, we assume that the matrix of the QBF is in 3CNF, and that in each clause do not appear both a literal and its complement. We translate the QBF $(Q_1X_1)\cdots(Q_nX_n)[G_1\wedge\cdots\wedge G_m]$ into an $\mathcal{ALC}$-concept

$$C = D \sqcap C_1^1 \sqcap \ldots \sqcap C_1^n.$$

in which all concept are formed using the atomic concept A and the atomic role P.

The concept D encodes the prefix, and is of the form $D_1 \sqcap \forall P.(D_2 \sqcap \forall P.(\ldots(D_{n-1} \sqcap \forall P.D_n)\ldots)$ where for $i \in 1..n$, each D_i corresponds to a quantifier of the QBF in the following way:

$$D_i := \begin{cases} (\exists P.A) \sqcap (\exists P.\neg A) & \text{if } Q_i = \forall \\ \exists P.\top & \text{if } Q_i = \exists \end{cases}$$

The concept C_1^i is obtained from the clause G_i using the concept A when a boolean variable occurs positively in G_i, $\neg A$ when it occurs negatively, and using l nestings of universal role quantification to encode the variable X_l. In detail, let k be the maximum index of all boolean variables appearing in G_i. Then, for $l \in 1..(k-1)$, one defines

$$C_l^i := \begin{cases} \forall R.(A \sqcup C_{l+1}^i) & \text{if } X_l \text{ appears positively in } G_i \\ \forall R.(\neg A \sqcup C_{l+1}^i) & \text{if } X_l \text{ appears negatively in } G_i \\ \forall R.C_{l+1}^i & \text{if } X_l \text{ does not appear in } G_i \end{cases}$$

and the last concept of the sequence is defined as

$$C_k^i := \begin{cases} \forall R.A & \text{if } X_k \text{ appears positively in } G_i \\ \forall R.\neg A & \text{if } \neg X_k \text{ appears negatively in } G_i. \end{cases}$$

It can be shown that each trace in a constraint system corresponds to a truth assignment to the boolean variables, and that all traces of a constraint system form a set of truth assignments consistent with the prefix. Therefore, we can conclude that Satisfiability in $\mathcal{ALC}$ is PSPACE-hard. Combining this result with the polynomial-space calculus given for $\mathcal{ALCNR}$, one obtains that Satisfiability (and Subsumption) in $\mathcal{ALCNR}$ are PSPACE-complete, and that the exponential-time behavior of the calculus cannot be improved unless PSPACE=PTIME.

Using the simulation of qualified existential quantification via subroles, one can use the same reduction to prove that Satisfiability and Subsumption in $\mathcal{ALUR}$ are PSPACE-hard (and thus PSPACE-complete). With a similar reduction, Donini et al. (1991a) proved that also Satisfiability in $\mathcal{ALNR}$ is PSPACE-hard. Observe that all these languages contain both sources of complexity.

This intuition carries over also when one or both sources of complexity are absent. In this way, the complexity analysis of reasoning in the $\mathcal{AL}$-languages was completed by Donini et al. (1991a). The only language not completely classified is $\mathcal{ALEN}$, where both sources of complexity are present (it can be proved both NP- and coNP-hard), but their combination in a PSPACE-hardness proof has not been found yet.

The results are summarized in Table 3. The complexity classes mentioned refer to Subsumption. For the complexity of Satisfiability, NP and coNP should be interchanged.

4 Reasoning with the ABox

In this section, we analyze in detail the problem of reasoning within an ABox. The basic reasoning task for retrieving information from an ABox

	OR-source absent	OR-source present
AND-source absent	$\mathcal{AL}, \mathcal{ALN}$ PTIME	$\mathcal{ALU}, \mathcal{ALUN}$ coNP-complete
AND-source present	$\mathcal{ALE}, \mathcal{ALR}, \mathcal{ALER}$ NP-complete	all other $\mathcal{AL}$-languages but $\mathcal{ALEN}$ PSPACE-complete

TABLE 3 Complexity of Subsumption in $\mathcal{AL}$-languages

is Instance Checking. We consider the Instance Checking problem in the case of an empty TBox, that amounts to checking whether the ABox implies that an individual is an instance of a given concept.

In particular, we address the question of whether Instance Checking can be solved by means of Subsumption algorithms. The basic technique underlying most of the approaches to check whether $\Sigma \models C(a)$ consists in retrieving all the assertions in the knowledge base Σ relevant to a given individual a, collecting them into a single concept, called *most specific concept* of a ($MSC(a)$), and then checking whether C subsumes $MSC(a)$. This technique, called Abstraction/Subsumption, has been broadly exploited in actual systems (see Kindermann 1990, Quantz and Kindermann 1990, Nebel 1990a).

The Abstraction/Subsumption idea is applicable to a number of languages. However, there are cases where, in order to check whether $\Sigma \models D(a)$, it is necessary to consider assertions about other objects in the knowledge base different from a, and the above method is no longer applicable. We discuss these cases in Sections 4.2 and 4.3.

Subsequently, we show in Section 4.4 how to enrich concept languages with an epistemic operator, so as to distinguish the knowledge about the world and knowledge about the state of the knowledge base. In particular, we demonstrate that the epistemic operator introduces a sophisticated query mechanism that can be used to decrease the complexity of Instance Checking.

4.1 Complexity Measures

The complexity of a problem is generally measured with respect to the size of its whole input. For instance, in Section 3 the complexity of Subsumption has been measured w.r.t. the sum of the size of the candidate subsumer and that of the candidate subsumee. However, in Instance Checking, two inputs of different kind are given, namely, the ABox and the query. For this reason, different kinds of complexity measures can be considered, similarly to what has been suggested by Vardi (1982) in database query answering.

Definition 4.1 Given an ABox Σ, a concept D, an individual a, we call:

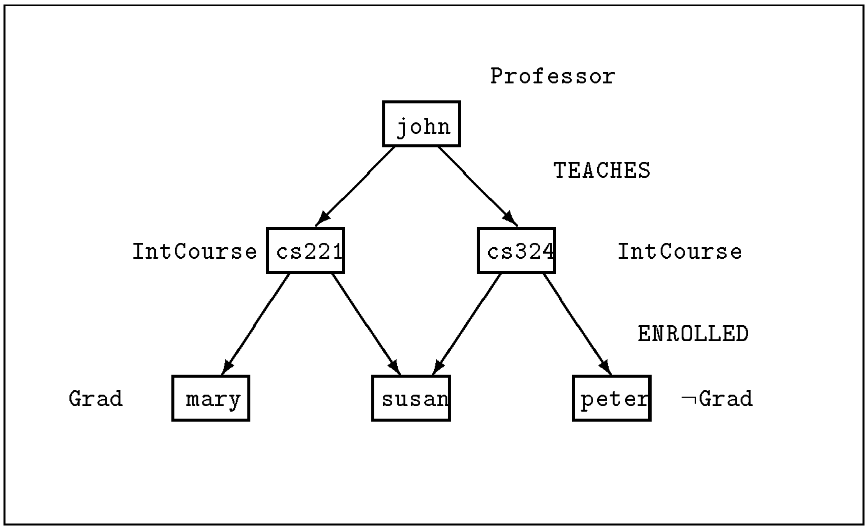

FIGURE 1 A pictorial representation of the ABox Σ

- *data complexity* of Instance Checking the complexity of checking if $\Sigma \models D(a)$ with respect to $|\Sigma|$;
- *combined complexity* of Instance Checking the complexity of checking if $\Sigma \models D(a)$ with respect to $|\Sigma| + |D|$

Combined complexity is the one usually considered in the literature in concept languages. It is a good measure of the actual cost of Instance Checking in the cases in which the ABox and the query are comparable in size.

On the other hand, data complexity is important when the size of the query is negligible with respect to the size of the ABox. Although such a measure is widely accepted in the database community, in the framework of concept languages the assumption that the size of the query is negligible is not always reasonable.

It is worth noticing that combined complexity is always higher or equal to data complexity. In fact, the complexity with respect to $|\Sigma|$ and $|D|$ obviously includes as a particular case that in which $|D|$ is small with respect to $|\Sigma|$.

4.2 Instance Checking in $\mathcal{ALE}$

We start our analysis on ABox-reasoning by studying Instance Checking in the particular language $\mathcal{ALE}$, which shows interesting properties. $\mathcal{ALE}$ is rich enough to express both significant classes of assertions in the ABox and meaningful queries. In particular, by virtue of qualified existential quantification it is possible to express queries requiring some form of navigation through the assertions on roles in the ABox. For example, the query: "find the individuals who teach a course in which a graduate student is enrolled" can be expressed by the $\mathcal{ALE}$-concept $\exists\mathtt{TEACHES}.\exists\mathtt{ENROLLED}.\mathtt{Grad}$.

We first consider data complexity of Instance Checking in $\mathcal{ALE}$. To this regard, it is easy to show that such complexity is at least as high as the complement of Satisfiability, and therefore is NP-hard. We now prove that it is coNP-hard too. Since Subsumption in $\mathcal{ALE}$ is NP-complete, a consequence of this result is that (assuming NP $\neq$ coNP) Instance Checking for $\mathcal{ALE}$ is strictly harder than Subsumption.

This unexpected result shows that Instance Checking in $\mathcal{ALE}$ suffers from a new *source of complexity*, which does not show up when reasoning with concept expressions (see Section 3.4). This source of complexity is related to the use of qualified existential quantification in the concept representing the query, which makes the behavior of the individuals heavily dependent on other individuals in the ABox. Let us illustrate this point by means of an example.

Example 4.2 Let Σ be the following ABox:

$$\begin{aligned}\Sigma = \{ & \mathtt{Professor(john)}, \mathtt{TEACHES(john, cs221)}, \mathtt{TEACHES(john, cs324)},\\ & \mathtt{IntCourse(cs221)}, \mathtt{IntCourse(cs324)},\\ & \mathtt{ENROLLED(cs221, mary)}, \mathtt{ENROLLED(cs221, susan)},\\ & \mathtt{ENROLLED(cs324, susan)}, \mathtt{ENROLLED(cs324, peter)},\\ & \neg\mathtt{Grad(peter)}, \mathtt{Grad(mary)}\}\end{aligned}$$

The ABox Σ is also shown in graphical form in Figure 1. It can be easily verified that Σ is satisfiable and that it has indeed several different models. In fact, it does not have complete knowledge about the represented world. For example, Σ doesn't know whether Susan is a graduate or not.

Consider the following query to the ABox Σ:

Query 1 $\Sigma \models \exists\mathtt{TEACHES}.(\exists\mathtt{ENROLLED}.\mathtt{Grad} \sqcap \exists\mathtt{ENROLLED}.\neg\mathtt{Grad})(\mathtt{john})$?

Query 1 asks whether John teaches a course in which both a graduate and an undergraduate are enrolled. At a superficial reading of the query, it might seem that the answer should be `NO`. The answer `NO` is supported by the fact that none of the courses taught by John is known to be in the requested conditions, i.e. Σ logically implies neither $\exists\mathtt{ENROLLED}.\mathtt{Grad} \sqcap \exists\mathtt{ENROLLED}.\neg\mathtt{Grad}(\mathtt{cs221})$ nor $\exists\mathtt{ENROLLED}.\mathtt{Grad} \sqcap \exists\mathtt{ENROLLED}.\neg\mathtt{Grad}(\mathtt{cs324})$. Nevertheless, the correct answer is `YES`, and in order to get it, one must reason by *case analysis*: As we have already remarked, the knowledge base does not provide the information as to whether Susan is a graduate or an undergraduate; however, in every model she must be either one or the other. This fact ensures that in every model for Σ either $\mathtt{Grad(susan)}$ or $\neg\mathtt{Grad(susan)}$ holds. Consider now the set of models for Σ in which $\mathtt{Grad(susan)}$ holds. In each of these models, the course CS324 is taken by both a graduate (Susan) and an undergraduate (Peter). Similarly, consider the set of the remaining models for Σ, i.e. the ones in which $\neg\mathtt{Grad(susan)}$ holds. It is easy to see that in every model for this set the course CS221, this time, is taken by both a graduate (Mary) and an undergraduate (Susan).

In conclusion, in every model for Σ either CS324 or CS221 is in the extension of $\exists\mathtt{ENROLLED.Grad} \sqcap \exists\mathtt{ENROLLED.}\neg\mathtt{Grad}$. It follows that in every model for Σ, the assertion σ is true proving that the correct answer is $\mathtt{YES}$.

We now show that the kind of reasoning required in the example makes instance checking in $\mathcal{ALE}$ coNP-hard with respect to data complexity.

The proof is based on a reduction from a suitable variation of the propositional satisfiability problem (SAT) to Instance Checking in $\mathcal{ALE}$. We define 2+2-CNF formula on an alphabet P as a CNF formula F such that each clause of F has exactly four literals: two positive and two negative ones, where the propositional letters are elements of $P \cup \{true, false\}$. Furthermore, we call 2+2-SAT the problem of checking whether a 2+2-CNF formula is satisfiable. The problem 2+2-SAT is proved NP-complete by Schaerf (1993).

Given a 2+2–CNF formula $F = C_1 \wedge C_2 \wedge \cdots \wedge C_n$, where $C_i = L^i_{1+} \vee L^i_{2+} \vee \neg L^i_{1-} \vee \neg L^i_{2-}$, we associate with it an ABox Σ_F and a concept Q as follows. Σ_F has one individual l for each letter L in F, one individual c_i for each clause C_i, one individual f for the whole formula F, plus two individuals $true$ and $false$ for the corresponding propositional constants. The roles of Σ_F are Cl, P_1, P_2, N_1, N_2, and the only concept name is A.

$$
\begin{aligned}
\Sigma_F \quad = \{ \quad & A(true), \neg A(false), \\
& Cl(f, c_1), Cl(f, c_2), \ldots, Cl(f, c_n), \\
& P_1(c_1, l^1_{1+}), P_2(c_1, l^1_{2+}), N_1(c_1, l^1_{1-}), N_2(c_1, l^1_{2-}), \\
& \ldots \\
& P_1(c_n, l^n_{1+}), P_2(c_n, l^n_{2+}), N_1(c_n, l^n_{1-}), N_2(c_n, l^n_{2-}) \ \}
\end{aligned}
$$

$$
Q \quad = \quad \exists Cl.(\exists P_1.\neg A \sqcap \exists P_2.\neg A \sqcap \exists N_1.A \sqcap \exists N_2.A).
$$

Intuitively, the membership to the extension of A or $\neg A$ corresponds to the truth values $true$ and $false$ respectively and checking if $\Sigma_F \models Q(f)$ corresponds to checking if in every truth assignment for F there exists a clause whose positive literals are interpreted as false, and whose negative literals are interpreted as true, i.e. a clause that is not satisfied.

Schaerf (1993) proved that a 2+2–CNF formula F is unsatisfiable if and only if $\Sigma_F \models Q(f)$. Since for any 2+2–CNF formula F, Q is fixed independently of F and Σ_F can be computed in polynomial time with respect to the size of F, such result proves that Instance Checking in $\mathcal{ALE}$ is coNP-hard in the size of the knowledge base.

Schaerf (1993) also shows that Instance Checking in $\mathcal{ALE}$ is in Π_2^p with respect to data complexity. We conjecture that the problem is actually complete for Π_2^p (i.e. it is Π_2^p-hard too). Instead, if combined complexity is considered, Donini et al. (1994b) show that Instance Checking in $\mathcal{ALE}$ is PSPACE-complete.

These results single out several interesting properties:

1. Reasoning in $\mathcal{ALE}$ suffers from an additional source of complexity which makes Instance Checking *not* polynomially reducible to Subsumption, and (unlike Subsumption in $\mathcal{ALE}$) not solvable by means of one single level of nondeterministic choice.
2. The problem of Instance Checking in $\mathcal{ALE}$ is in Π_2^p if data complexity is considered, and is PSPACE-complete if combined complexity is considered. This shows that in order to have a significant complexity measure of reasoning problems in DL-systems, we must pay attention to which is the crucial data of the application.
3. With respect to combined complexity, $\mathcal{ALE}$ is in the same class (PSPACE) as more complex languages (e.g. $\mathcal{ALCNR}$, see Buchheit et al. 1995). Therefore, whenever the expressivity of $\mathcal{ALE}$ is required, other constructors can be added without any increase of the (worst-case) computational complexity.
4. The source of complexity identified in this section is not related to the possibility of nesting an arbitrary number of existential and universal quantifiers—like the one pointed out in Section 3.4 for Subsumption in $\mathcal{ALE}$. In fact, it leads to intractability even using only two nested existential quantifiers and no universal ones. This is an important observation, since long chains of nested quantifiers seem not to appear frequently in practice.

With regard to Point 1, one may object that there are cases in which the reasoning process underlying this source of complexity is not in the intuition of the user, as pointed out by Query 1. Therefore, such process should be ruled out by the reasoner (and by the semantics). On the contrary, there are cases in which this kind of reasoning seems to agree with the intuition and therefore it must be taken into account. In Section 4.4 we address the issue of the appropriate semantics for queries involving existential quantifications. In particular, we show how is possible to achieve a more sophisticated control on the semantics, by means of an epistemic operator in the query language.

4.3 Complexity of Instance Checking and Consistency

An extensive analysis of the complexity of ABox-reasoning in several languages is carried out by Donini et al. (1994b). The analysis singles out several interesting properties. Firstly, Consistency is shown to be in the same complexity class as Concept Satisfiability for all the considered languages. Secondly, different results are obtained for Instance Checking depending upon the language used to express the ABox. In particular:

1. Languages with polynomial Subsumption, such as $\mathcal{AL}$ and $\mathcal{ALN}$, preserve their tractability, although, in general, some additional work with respect to Subsumption is required.
2. The new source of complexity singled out for $\mathcal{ALE}$ in the previous section does not arise in the other languages in which Subsumption is NP-complete, such as $\mathcal{ALR}$. In fact, although $\mathcal{ALR}$ can simulate qualified existential quantification in ABox assertions (Section 3.4), it cannot express qualified existential quantification in the query concept, neither explicitly nor implicitly. It follows that Instance Checking in $\mathcal{ALR}$ is NP-complete, exactly like Subsumption in $\mathcal{ALR}$.
3. With regard to other languages, Instance Checking is again in the same complexity class as Subsumption. This happens either because no navigation through the knowledge base is allowed (e.g. $\mathcal{ALU}$), or because the language is rich enough (e.g. $\mathcal{ALC}$), so that Subsumption is already computationally demanding.

An analysis of the complexity of Instance Checking for languages including the constructor $\mathcal{O}$ has been started by Lenzerini and Schaerf (1991) and has been carried out extensively by Schaerf (1994b).

The analysis shows that, in languages with $\mathcal{O}$, all reasoning services have the same complexity. This is due to the fact that individuals can now appear in concept expressions, hence the source of complexity related to their treatment has already an impact when reasoning on concept expressions.

The corresponding complexity results also show that reasoning with the construct $\mathcal{O}$ is generally hard. When $\mathcal{O}$ is added to languages such as $\mathcal{AL}$,$\mathcal{ALE}$, complexity of reasoning increases. Even in those cases in which reasoning in a language including $\mathcal{O}$ is in the same complexity class of reasoning in the language without $\mathcal{O}$ (e.g., $\mathcal{ALCO}$, $\mathcal{ALC}$), the algorithms taking into account $\mathcal{O}$ are generally more complex (in terms of both time and space) than the algorithms that do not deal with $\mathcal{O}$.

Finally, we remark that results presented here for $\mathcal{O}$ are based on a standard first-order semantics. Semantics employed in actual systems can be a variant of pure first-order semantics (see e.g. Borgida and Patel-Schneider 1994). However, there is no general agreement on whether such non-standard semantics are appropriate and understandable by users.

4.4 Queries with an Epistemic Operator

One of the strengths of concept languages is that they are given a set-theoretic first-order semantics. However first-order semantics, for other aspects, is also their weakness, since it leaves out several aspects of practical systems, that are generally described in terms of procedural or non-monotonic mechanisms. Therefore, it seems appropriate to enrich such a semantics both to explore novel language features and to account for aspects that cannot be described in a standard first-order framework. The need of some enrichment of this sort is discussed in the literature (see for example Doyle and Patil 1991, Woods 1991) and can be easily recognized by looking at recent DL-systems such as CLASSIC and CLASP (Yen et al. 1991).

Although here we are not concerned with a full treatment of these issues, we present a non-first-order extension of concept languages that provides special forms of reasoning in the ABox. In (Donini et al. 1992b, Donini et al. 1994a, Donini et al. 1995a), it is proposed to enrich concept languages with an epistemic operator defined in the style of (Levesque 1984, Reiter 1990, Lifschitz 1991). In particular, we focus our attention on the use of the epistemic operator to define a more powerful query language.

We first provide some examples that show how the epistemic query language allows one to address both aspects of the external world as represented in the knowledge base, and aspects of what the knowledge base knows about the external world. We then discuss the complexity of reasoning.

4.4.1 Semantics

In this section we present *epistemic concept languages* which are concept languages extended with an epistemic operator. In details, an epistemic concept language includes the concept constructor $\mathbf{K}C$ and the role constructor $\mathbf{K}P$, where C and P are a concept and a role, respectively. As a notation, we call $\mathcal{LK}$ the concept language obtained by adding the above constructors to a concept language $\mathcal{L}$. Intuitively, $\mathbf{K}C$ denotes the set of individuals which are *known* to be instances of the concept C, i.e. those individuals that are in the extension of C in *every* model for the knowledge base (and similarly for $\mathbf{K}P$).

The semantics of the epistemic operator is an adaptation to the framework of concept languages of the one proposed in (Lifschitz 1991).

An *epistemic interpretation* is a pair $(\mathcal{I}, \mathcal{W})$ where $\mathcal{I}$ is an interpretation and $\mathcal{W}$ is a set of interpretations such that

1. concept and role names are interpreted independently of $\mathcal{W}$; that is $A^{\mathcal{I},\mathcal{W}} = A^{\mathcal{I}}$ and $P^{\mathcal{I},\mathcal{W}} = P^{\mathcal{I}}$.
2. the interpretation of the non-epistemic constructors is obtained from

Figures 1 and 2 by replacing $\mathcal{I}$ with $\mathcal{I}, \mathcal{W}$ and $\Delta^{\mathcal{I}}$ with Δ (for example $(C \sqcap D)^{\mathcal{I},\mathcal{W}} = C^{\mathcal{I},\mathcal{W}} \cap D^{\mathcal{I},\mathcal{W}}$).

3. for the epistemic constructor the following equations are satisfied

$$\begin{aligned}(\mathbf{K}C)^{\mathcal{I},\mathcal{W}} &= \bigcap_{\mathcal{J}\in\mathcal{W}} (C^{\mathcal{J},\mathcal{W}}) \\ (\mathbf{K}P)^{\mathcal{I},\mathcal{W}} &= \bigcap_{\mathcal{J}\in\mathcal{W}} (P^{\mathcal{J},\mathcal{W}}).\end{aligned}$$

Some issues typical of first-order modal systems arise. Such issues concern the interpretation structures and are dealt with by the following assumptions:

- every interpretation is defined over a fixed domain, called Δ (Common Domain Assumption), that is $\Delta^{\mathcal{I}} = \Delta$ for any $\mathcal{I}$;
- for every interpretation the mapping from the individuals into the domain elements, called γ, is fixed (Rigid Term Assumption), that is $a^{\mathcal{I},\mathcal{W}} = \gamma(a)$.

Notice that, since the domain is fixed independently of the interpretation, it is meaningful to refer to the conjunction of the extensions of a concept in different interpretations. It follows that $\mathbf{K}C$ is interpreted in $\mathcal{W}$ as the set of objects that are instances of C in every interpretation belonging to $\mathcal{W}$. In this sense, $\mathbf{K}C$ represents those objects known to be instances of C in $\mathcal{W}$.

An ABox Σ in a concept language $\mathcal{L}$ logically implies an assertion σ in $\mathcal{LK}$, written $\Sigma \approx\!\!\!\mid\ \sigma$, if σ is true in every pair $(\mathcal{I}, \mathcal{M}(\Sigma))$, where $\mathcal{M}(\Sigma)$ the set of models of Σ and $\mathcal{I}$ is an element of $\mathcal{M}(\Sigma)$.

Given an $\mathcal{L}$-ABox Σ, an $\mathcal{LK}$-concept C, and an individual a, the *answer* to the query $C(a)$ posed to Σ, is `YES` if $\Sigma \mid\!\approx C(a)$, `NO` if $\Sigma \mid\!\approx \neg C(a)$, and `UNKNOWN` otherwise.

Our goal here is to show that the use of epistemic operators in queries allows for a more sophisticated interaction with the DL-system. For this purpose we consider the ABox Σ of Example 4.2 and we provide various kinds of queries that can be posed to it using the language $\mathcal{ALEK}$. In particular, in order to understand the role of the epistemic operator $\mathbf{K}$, we consider both $\mathcal{ALE}$ queries and modified versions of them including $\mathbf{K}$. The comparison between their respective semantics highlights the role of $\mathbf{K}$ in the query language.

Example 4.3 We consider two queries that involve universal quantifiers:

Query 2 $\Sigma \mid\!\approx \forall$`TEACHES.IntCourse(john)`? *Answer:* `UNKNOWN`.

Query 3 $\Sigma \mid\!\approx \forall\mathbf{K}$`TEACHES.`$\mathbf{K}$`IntCourse(john)`? *Answer:* `YES`.

Query 2 asks whether every course taught by John is an intermediary one. The answer is `UNKNOWN` because there are models for Σ in which John teaches only intermediary courses as well as models in which he teaches also courses that are not intermediary.

Query 3, instead, asks whether every course that is known to be taught by John is also known to be an intermediary course. Since the only courses taught by John are CS221 and CS324, and they are indeed intermediary courses, the answer to Query 3 is `YES`. Note that this is a form closed-world reasoning, since the universal quantification takes into account only known individuals.

Example 4.4 We now consider two queries involving nested quantifiers: Query 1 has been already discussed in Section 4.2 (we report it here for convenience) and Query 4 is its modified version with the epistemic operator.

Query 1 $\Sigma \models\!\!\approx \exists\mathtt{TEACHES}.(\exists\mathtt{ENROLLED}.\mathtt{Grad} \sqcap \exists\mathtt{ENROLLED}.\neg\mathtt{Grad})(\mathtt{john})$?

Answer : `YES`.

Query 4 $\Sigma \models\!\!\approx \exists\mathbf{K}\mathtt{TEACHES}.\mathbf{K}(\exists\mathtt{ENROLLED}.\mathtt{Grad} \sqcap \exists\mathtt{ENROLLED}.\neg\mathtt{Grad})(\mathtt{john})$?

Answer: `NO`.

As we already said, the answer of Query 1 is `YES` due to a case analysis. On the other hand, Query 4 asks whether John is known to teach a course that is known to be an instance of the concept $\exists\mathtt{ENROLLED}.\mathtt{Grad} \sqcap \exists\mathtt{ENROLLED}.\neg\mathtt{Grad}$. The courses known to be taught by John are CS221 and CS324, and therefore none of them is within the conditions required by the query.

Query 1 shows how, in some cases, the first-order semantics might not agree with the intuitive reading of a query. In fact, most people tend to read Query 1 as requiring the reasoning pattern that is actually associated with the semantics of Query 4. For this reason, in our opinion, it is important to have the operator **K**, which allows us to express both types of queries.

Several other queries and other possible uses of the epistemic operator are discussed in (Donini et al. 1992b, Donini et al. 1994a).

4.4.2 Complexity Results

In this section we address the computational complexity of answering epistemic queries. To this aim, we realize an interesting relation between reasoning with epistemic concepts and reasoning with concepts involving collections of individuals.

The intuition underlying such relation is that a concept of the form $\mathbf{K}C$ can be considered equivalent to the concept $\{a_1, \ldots, a_n\}$, where $a_1, \ldots, a_n$ are exactly the individuals for which $\Sigma \models C(a_i)$ holds. However, reasoning with **K** is more complex than reasoning with $\mathcal{O}$ for two reasons:

1. when using $\mathcal{O}$ the set $\{a_1, \ldots, a_n\}$ is given, while when using **K**, such

a set must be computed (possibly in a recursive way, due to nested occurrences of $\mathbf{K}$)

2. when the $\mathbf{K}$ operator is in front of a role name, $\mathbf{K}P$, this is equivalent to a role of the form $\{(a_1, b_1), \ldots, (a_n, b_n)\}$. Such a role is not expressible even in languages with $\mathcal{O}$.

Despite the above differences, using a technique similar to the one used for reasoning in $\mathcal{ALCO}$, in (Donini et al. 1995a) we prove that Instance Checking in $\mathcal{ALC}$ using $\mathcal{ALCK}$ as query language is a PSPACE-complete problem.

The above result proves that answering $\mathcal{ALCK}$-queries is not (worst-case) computationally harder than answering $\mathcal{ALC}$-queries. This allows us to conclude that the use of epistemic operators in $\mathcal{ALC}$ does not substantially increase the complexity in query answering. This is extremely interesting, especially in light of the expressive power gained. Next we show also a case in which the epistemic operator decreases the complexity.

In the examples given above we show that the use of $\mathbf{K}$ may allow us to express queries ruling out the case analysis. In particular, this is done by replacing concepts of the form $\exists R.C$ with concepts of the form $\exists \mathbf{K}R.\mathbf{K}C$. In (Donini et al. 1995a) it is shown that such a replacement can lessen the complexity of reasoning. More precisely, considering an ABox in $\mathcal{AL}$, it is possible to answer $\mathcal{ALEK}$-queries in polynomial time w.r.t. $|\Sigma|$ (data complexity) provided that the only qualified existential quantifications are of the form $\exists \mathbf{K}R.\mathbf{K}E$. Conversely, recall that answering general $\mathcal{ALE}$ queries in coNP-hard, as shown in Section 4.2.

5 Reasoning with the TBox

In this section, we deal with the problem of reasoning with a set of TBox-statements. As we said in Section 2, TBox-statements are used to specify the intensional component of a knowledge base, i.e., properties of classes and relationships between classes. The basic reasoning services to be provided on TBoxes are Concept Satisfiability and Subsumption. Here we address these two problems in the hypothesis that the knowledge base Σ has the form $\langle \mathcal{T}, \emptyset \rangle$, i.e., in the absence of ABox-assertions. In particular we consider three types of TBox-statements, and, correspondingly, we distinguish between three kinds of TBoxes: primitive TBoxes, simple TBoxes, and free TBoxes.

5.1 Primitive TBoxes

Recalling definitions of Section 2.2.1, a *primitive TBox* is a set of statements (called primitive concept specifications) of the form:

$$A \dot{\leq} C$$

where each statement is used to specify a necessary condition on the instances of the concept whose name is A, namely that they are instances of the concept C as well.

The semantics of a primitive concept specification is given in Section 2. Intuitively, each specification is used to restrict the set of admissible interpretations to those that obey the inclusion condition expressed in the specification.

Primitive TBoxes provide a characterization of Frames, Semantic Networks, and several Database Models. Indeed, the structure of the intensional component of a knowledge base expressed in each of these formalisms can be correctly represented in terms of a set of primitive concept specifications. Two recent papers (Buchheit et al. 1994a, Calvanese et al. 1994) address the problems of developing suitable reasoning techniques and studying the computational complexity of reasoning in primitive TBoxes.

In (Buchheit et al. 1994a), an architecture for DL-systems is presented, constituted by three components. One of these components, called schema, is just a set of primitive concept specifications in a language obtained from $\mathcal{FL}^-$ by adding the two constructors $(\leq\ 1\ P), (\geq\ 1\ P)$, where P is an atomic role. The paper shows that Subsumption between atomic concepts in a schema can be decided in polynomial time. However, if the schema language is enriched with inverse roles[3] or with the negation of atomic concepts, Subsumption becomes NP-hard. In the paper it is also observed that, if the DL-system architecture is carefully designed, primitive concept specifications on one hand are proved very useful in domain modeling, and on the other hand, do not increase the complexity of reasoning. This claim is supported by three case studies, where a primitive TBox in the style described above, is added to three working systems, namely ConceptBase, kris, and classic.

In (Calvanese et al. 1994), primitive TBoxes using more powerful languages are investigated. The main language considered in that paper, called $\mathcal{ALUNI}$, is obtained from $\mathcal{ALUN}$ by adding the constructor for inverse roles. Inverse roles are shown essential in order to express the modeling constructors used in Semantic and Conceptual Database Models. It is also shown that primitive TBoxes expressed in this language exhibit an interesting property: a concept may be satisfiable in a TBox $\mathcal{T}$ only in infinite models, i.e., models with infinite domains. In other words, (unrestricted) Concept Satisfiability is different from *Finite Concept Satisfiability*, i.e., the problem of checking if a concept has a nonempty extension in at least one finite model of the TBox. Observe that finite Satisfiability is important in

[3]Given a role R, its inverse is usually denoted as R^{-1} and its semantics is $(R^{-1})^\mathcal{I} = \{(d_1, d_2) \mid (d_2, d_1) \in R^\mathcal{I}\}$.

the Database setting, where models for the TBox correspond to Database states, and therefore are assumed to be finite.

5.2 Simple TBoxes

Recall that a concept definition is a statement of the form:

$$A \doteq C$$

(called the definition of A) and is intended to provide a definitional account of the concept whose name is A. A simple TBox $\mathcal{T}$ is a set of concept definitions, that obeys the following syntactic restriction: for every concept name A, there is at most one definition of A in $\mathcal{T}$.

Informally, defining a concept A means singling out a set of necessary and sufficient conditions for an object to belong to the extension of A. All definitions, however, must ultimately rely on some primitive notions that are left undefined. Therefore, in simple TBoxes, we distinguish between so-called primitive and defined concepts, where the former are those concepts whose meaning is assumed somehow understood (and are therefore left undefined in the TBox), whereas the latter are those concepts that have a characterization in the TBox in terms of the set of primitive concepts and roles. The form of the TBox determines the sets of primitive and defined concepts: defined (resp. primitive) concepts are those that do (not) have a definition in the TBox.

Intuitively, for any simple TBox $\mathcal{T}$, we would like to be able to assign a unique interpretation to the defined concepts in $\mathcal{T}$, once we have fixed the interpretation of the primitive ones. In order to capture such an intuition, we introduce the notion of initial interpretation. Let us call *initial* an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ whose interpretation function maps every atomic concept that is primitive in $\mathcal{T}$ to a subset of $\Delta^{\mathcal{I}}$, and every role to a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. Given an initial interpretation $\mathcal{I}$, we obtain an interpretation for $\mathcal{T}$ by extending the interpretation function $\cdot^{\mathcal{I}}$ to all concepts and roles in $\mathcal{T}$ (including complex and defined concepts, as well complex roles). The denotation of complex concepts and roles must obey the conditions reported in Tables 1 and 2, respectively. Moreover, it must assign the same denotation to A and C, for every definition $A \doteq C$ in $\mathcal{T}$. Now the question is:

Given an initial interpretation, is there a unique way to determine the interpretation of defined concepts?

We will see that the answer to this question depends on the form of the TBox, and in particular on whether the TBox contains cycles or not.

In order to precisely define the notion of cycle, observe that to a simple TBox $\mathcal{T}$ we can associate a total function $f_{\mathcal{T}}$ from atomic concepts to concept expressions, defined as follows: $f_{\mathcal{T}}(A) = C$, if $A \doteq C \in \mathcal{T}$, and $f_{\mathcal{T}}(A) = A$, if A is primitive in $\mathcal{T}$. Now, following Nebel (1991), we say that A *directly uses* an atomic concept B in $\mathcal{T}$, if the expression $f_{\mathcal{T}}(A)$ contains B. Let *uses* (in $\mathcal{T}$) denote the transitive closure of the relation *directly uses* (in $\mathcal{T}$). A simple TBox $\mathcal{T}$ is said to be *cyclic* if some atomic concept uses itself in $\mathcal{T}$, acyclic otherwise. Intuitively, cycles correspond to recursive definition of concepts.

We can now return to the above question, and analyze the two cases of acyclic and general simple TBoxes (or simply TBoxes), respectively.

5.2.1 Acyclic simple TBoxes

Acyclic simple TBoxes (or acyclic TBoxes for short) are ubiquitous in DL-systems. Indeed, almost all existing systems do not allow the specification of cyclic TBoxes. Nevertheless, formal properties of acyclic TBoxes have been investigated only recently. Nebel (1990b) shows that for acyclic TBoxes the answer to the above question is positive: Given an acyclic TBox $\mathcal{T}$, any initial interpretation can be extended to at most one model of $\mathcal{T}$. This means that there is no ambiguity in interpreting a defined concept, on the basis of the interpretation for primitive concepts. Moreover, it is easy to see that Subsumption in simple TBoxes can be reduced to Subsumption in an empty terminology. Indeed, if we want to check whether C is subsumed by D in an acyclic TBox $\mathcal{T}$, we can iteratively substitute every occurrence of any defined concept A in C and D by the corresponding expression $f_{\mathcal{T}}(A)$, obtaining the two expressions C', D'. Since $\mathcal{T}$ is acyclic, this process terminates in a finite number of iterations, and it holds that $\langle \mathcal{T}, \emptyset\rangle \models C \sqsubseteq D$ if and only if $\langle \emptyset, \emptyset\rangle \models C' \sqsubseteq D'$.

This result shows that reasoning with acyclic TBoxes can be reduced to reasoning with concept expressions. However, Nebel (1990b) also shows that the above method may lead to a cost of the Subsumption algorithm which is exponential in the size of the TBox. Surprisingly, the same paper shows that intractability does not stem from the method, but from the inherent complexity of the problem: determining Subsumption between C and D in an acyclic TBox $\mathcal{T}$ is co-NP-complete, even if the language used to express C, D and $\mathcal{T}$ is less expressive than $\mathcal{FL}^-$. The technique used to establish this result is interesting on its own, and is based on reducing equivalence between nondeterministic finite state automata to two Subsumption tests in a TBox encoding the two automata.

The reason why this result is surprising is that it was believed that Subsumption in a TBox is not harder than Subsumption of concept descriptions. Nebel (1990b) argues that indeed, the exponential worst case is unlikely to occur, and this would explain why Subsumption algorithms in acyclic simple TBoxes are well behaved in practice.

5.2.2 Cyclic simple TBoxes

Consider the following cyclic simple TBox $\mathcal{T}$

$$B \doteq A \sqcap \forall P.B,$$

and the initial interpretation $\mathcal{I}'$ such that: $\Delta^{\mathcal{I}'} = \{a, b, c, d\}$, $A^{\mathcal{I}'} = \{a, b, c, d\}$, and $P^{\mathcal{I}'} = \{(a, b), (c, d), (d, d)\}$. It is easy to see that there are two ways to extend $\mathcal{I}'$ to a model $\mathcal{I}$ for $\mathcal{T}$:

1. $\mathcal{I}'$ is extended to $\mathcal{I}$ such that $B^{\mathcal{I}} = \{a, b\}$
2. $\mathcal{I}'$ is extended to $\mathcal{I}$ such that $B^{\mathcal{I}} = \{a, b, c, d\}$.

This example shows that, given an initial interpretation for a cyclic TBox $\mathcal{T}$, there is no unique way to extend it to a model for $\mathcal{T}$, i.e., to determine the denotation of defined concepts.

In order to discuss the problems related to cyclic definitions, consider a TBox $\mathcal{T}$ constituted by a single cyclic definition $A \doteq C$, where C is an expression containing A. Semantically, $A \doteq C$ is a sort of equation specifying that, for any interpretation $\mathcal{I}$, the subsets of $\Delta^{\mathcal{I}}$ which can be tied to the concept A must be a *solution* of $A^{\mathcal{I}} = C^{\mathcal{I}}$.

Notice that we can associate to a definition statement an operator from subsets of $\Delta^{\mathcal{I}}$ to subsets of $\Delta^{\mathcal{I}}$, such that the solutions of the equation correspond to the fixpoints of the operator. Consider the following definition:

(5) $$A \doteq \exists \texttt{CHILD}.A$$

we can associate to it the operator: $\lambda S.\{a \in \Delta^{\mathcal{I}} \mid \exists b.(a, b) \in \texttt{CHILD}^{\mathcal{I}} \wedge b \in S\}$, for any interpretation $\mathcal{I}$. In what follows, we write $F(\cdot)$ to denote such an operator.

In (Baader 1990, Nebel 1991, De Giacomo and Lenzerini 1994a), three ways of interpreting cyclic definitions are discussed, according to one of the following semantics:

Descriptive Semantics. $A \doteq F(A)$ is a *constraint* stating that A has to be *some* solution of the corresponding equation. For example, Definition (5) states that the individuals in the class A have a child in the class A, and the individuals that have a child in the class A are themselves in the class A, where A is no further specified. In general, the definition statement $A \doteq C$ in the descriptive semantics is equivalent to $\{A \mathrel{\dot{\leq}} C,\ C \mathrel{\dot{\leq}} A\}$. It follows that the descriptive semantics is not appropriate for defining concepts. On the other hand, it allows one to express necessary and sufficient conditions for concepts, like in:

(6) $$\texttt{Human} \doteq \texttt{Mammal} \sqcap \exists \texttt{PARENT} \sqcap \forall \texttt{PARENT}.\texttt{Human}$$

Least Fixpoint Semantics. $A \doteq F(A)$ specifies that A is to be interpreted as the smallest solution (if it exists) of the corresponding equation. If the operator associated with the definition statement is monotonic, then for any interpretation $\mathcal{I}$ satisfying the definition statement, the corresponding equation singles out a *unique* subset of $\Delta^{\mathcal{I}}$, hence it *defines* the concept A. For Definition (5), the least fixpoint semantics leads to identify A with $\bot$. The same holds for Human in Definition (6). It is evident that the least fixpoint semantics does not help in the above examples. On the other hand, the least fixpoint semantics is particularly suitable for providing inductive definitions of concepts. Consider the case of a single-source directed acyclic graph (DAG):

1. The EMPTY-DAG is a DAG (base step).
2. A NODE that has connections and all connections are DAG, is a DAG (inductive step).
3. Nothing else is a DAG.

We can easily write a definition statement reflecting the first two conditions:

$$\texttt{Dag} \doteq \texttt{EmptyDag} \sqcup (\texttt{Node} \sqcap \exists\texttt{ARC} \sqcap \forall\texttt{ARC.Dag})$$

and, to enforce the third condition, we need to take the smallest solution of the corresponding equation.

Greatest Fixpoint Semantics. $A \doteq F(A)$ specifies that A is to be interpreted as the greatest solution (if it exists) of the corresponding equation. Again, if the operator associated with the definition statement is monotonic, then for any interpretation $\mathcal{I}$ satisfying the definition statement, the corresponding equation singles out a *unique* subset of $\Delta^{\mathcal{I}}$. For Definition (5), the greatest fixpoint semantics leads to interpret A as the class of *all* the individuals having a child in A. More generally, the greatest fixpoint semantics is suitable for defining classes of individuals whose structure is *circularly repeating*. As another example, the class FoB (FOrever Blond) of individuals who are blond and generation after generation have some descendant who is FoB is defined by

$$\texttt{FoB} \doteq \texttt{Blond} \sqcap \exists\texttt{CHILD.FoB}$$

With the above definition statement, we want to denote the class of *all* individuals satisfying the equation corresponding to the above definition statement, that is, we want its greatest solution. On the other hand, if we interpret the two definition statements

$$\begin{array}{lcl} \texttt{Human} & \doteq & \texttt{Mammal} \sqcap \exists\texttt{PARENT} \sqcap \forall\texttt{PARENT.Human} \\ \texttt{Horse} & \doteq & \texttt{Mammal} \sqcap \exists\texttt{PARENT} \sqcap \forall\texttt{PARENT.Horse} \end{array}$$

with the greatest fixpoint semantics, we obtain a rather unintuitive result: for any interpretation $\mathcal{I}$ satisfying the above definition statements, $\texttt{Human}^{\mathcal{I}} = \texttt{Horse}^{\mathcal{I}}$.

The computational properties of the three semantics are studied in (Baader 1990, Nebel 1991). It turns out that, for the language $\mathcal{FLN}^{-}$, Subsumption in cyclic TBoxes is PSPACE-complete with respect to both least and greatest fixpoint semantics. With regard to the descriptive semantics, Subsumption is both coNP-hard and in PSPACE, but it is not known if it is complete for PSPACE.

5.3 Free TBoxes

Free TBoxes are sets of concept inclusions ($C \dot{\leq} D$) and concept equations ($C \doteq D$) (see Section 2). Reasoning in free TBoxes requires different techniques than those studied for primitive and simple TBoxes. This is also confirmed by complexity results: for the language $\mathcal{FL}^-$, while reasoning in simple TBoxes adopting descriptive semantics is in PSPACE, reasoning in free TBoxes is EXPTIME-hard (McAllester 1991). However, the distinction disappears for languages containing union and negation, which can simulate a whole free TBox in one primitive concept specification interpreted under descriptive semantics (Buchheit et al. 1993).

For the sake of brevity, we do not delve into the details of reasoning techniques. We simply note that there are two main approaches. The first approach is based on an extension of the Tableaux calculus (see Buchheit et al. 1993), while the second one is based on the correspondence between description logics and logics of programs. Such a correspondence, first discussed in (Schild 1991), has been proved extremely useful for devising Subsumption algorithms in free TBoxes expressed in very powerful languages. Recent work on this subject can be found in (De Giacomo and Lenzerini 1994a, Schild 1994).

6 Reasoning in a complete DL-system

The ultimate goal of the research about reasoning in Description Logics is to reason on a complete DL-system $\Sigma = \langle \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{T}$ is a TBox, and $\mathcal{A}$ is an ABox, both expressed in a concept language $\mathcal{L}$. Although this is the most general problem related to reasoning in Description Logics, research on this subject has developed only recently. Similarly to the case of free TBoxes, there are two main approaches to the development of suitable techniques for reasoning in complete DL-systems. One of the approaches is based on extending the tableaux-based calculus presented in Section 3.3 to deal with both intensional and extensional knowledge, and the other one is based on exploiting the correspondence between Description Logics and logics of programs. The interested reader is referred to (Buchheit et al. 1993) and (De Giacomo and Lenzerini 1994a) for an account of these approaches.

7 Conclusions

We have surveyed the most important issues related to reasoning problems arising in the construction of knowledge representation systems based on Description Logics. We have characterized such DL-systems as composed by two parts: axioms about concepts in the TBox representing intensional knowledge, and assertions about individuals in the ABox representing extensional knowledge. We arranged the presentation by considering different TBox-ABox settings, and looking at corresponding reasoning problems.

However, research on Description Logics has made several other contributions, as the reader can find in a series of workshops specifically dedicated to this subject (Nebel et al. 1991, MacGregor et al. 1992, Baader et al. 1994, Borgida et al. 1995). In the following we briefly mention some of the issues, which were outside the scope of this survey.

Implementation issues. The implementation of DL-systems involves many other aspects besides the computational analysis of reasoning problems. Engineering of knowledge representation systems is illustrated for example in (Brachman 1992). More specifically, with regard to the implementation of reasoning procedures, the tableaux-based reasoning techniques which we have considered in this survey should be equipped with specialized strategies for rule selection and application. Another approach to the characterization of reasoning methods is to look at existing implementations and then try to characterize the computations done by the system, possibly using a non-standard semantics (Borgida and Patel-Schneider 1994).

Non-first-order features. The analysis developed in this paper was limited to deal with the first-order formalization of knowledge representation systems based on classes, which means that we have addressed neither nonmonotonic nor behavioral features.

There is a large body of research on nonmonotonic reasoning within taxonomies, starting from the use of defaults in Semantic Networks and Frames. Some of this research specifically addresses this issue in the framework of Description Logics (see for example Baader and Hollunder 1992, Quantz and Royer 1992, Padgham and Nebel 1993, Baader and Hollunder 1993, Straccia 1993, Padgham and Zhang 1993, Donini et al. 1995b).

Knowledge representation systems typically provide several forms of behavioral features. Some of them, such as methods or daemons, closely follow the object-oriented programming paradigm, others take the form of procedural rules, and are borrowed from production rule systems. The use of production rules and forward reasoning can be nicely formalized in Description Logics by using the epistemic operator defined in Section 4 to characterize what is known by the system (Donini et al. 1992b, Donini et al. 1994a).

Applications. We have not discussed the fields of application where the representation and reasoning techniques discussed so far can be profitably applied. Description Logics as well as their predecessors Semantic Networks and Frames have been initially developed for applications in natural language understanding. However, Description Logics can be effectively used in the construction of knowledge-based applications in several domains (see for example Wright et al. 1993), like configuration and software engineering. Recently we have seen attempts to use them in the fields of machine learning (see for example Cohen and Hirsh 1994) and planning (see for example Devambu and Litman 1991, Weida and Litman 1992, Artale and Franconi 1994, where the basic formalism is extended to deal with time and action).

Related fields. Finally, there are close relationships between Description Logics and formalisms developed in other areas of computer science. In particular, there is a strict relationship with logics of programs, originally studied for program verification and analysis (see Schild 1991, Schild 1994, De Giacomo and Lenzerini 1994a). As we said in Sections 5 and 6, this correspondence has provided new useful tools for studying description logics equipped with a very expressive concept language, as well as insights on the definitional mechanisms of the TBox. Moreover, there are strong links with the field of Databases and in particular with Semantic and Object-oriented Data Models (see for example Bergamaschi and Nebel 1993, Bergamaschi and Sartori 1992, Borgida 1992, Calvanese et al. 1994). Research in this direction can be significant not only for the exchange of techniques and results, but may have a strong impact on the development of tools for database specification and design.

Acknowledgements

This work has been supported by the Esprit Basic Research Action 6810 (Compulog 2) and by the Italian National Research Council as part of the Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo, Sottoprogetto 7, LdR Ibridi.

References

Abrial, J. R. 1974. Data Semantics. In *Data Base Management*, ed. J. W. Klimbie and K. L. Koffeman. 1–59. North-Holland Publ. Co., Amsterdam.

Artale, Alessandro, and Enrico Franconi. 1994. A computational Account for a Description Logic of time and action. In *Proc. of the 4th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-94)*, ed. J. Doyle, E. Sandewall, and P. Torasso, 3–14. Bonn. Morgan Kaufmann, Los Altos.

Attardi, Giuseppe, and Maria Simi. 1981. Consistency and Completeness of OMEGA, a Logic for Knowledge Representation. In *Proc. of the 7th Int. Joint Conf. on Artificial Intelligence (IJCAI-81)*, 504–510. Vancouver, British Columbia.

Baader, Franz. 1990. Terminological Cycles in KL-ONE-based Knowledge Representation Languages. Technical Report RR-90-01. Kaiserslautern, Germany: Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI). An abridged version appeared in *Proc. of the 8th Nat. Conf. on Artificial Intelligence AAAI-90*, pp. 621–626.

Baader, Franz, Hans-Jürgen Bürkert, Jochen Heinsohn, Bernhard Hollunder, Jürgen Müller, Bernard Nebel, Werner Nutt, and Hans-Jürge Profitlich. 1991. Terminological Knowledge Representation: A Proposal for a Terminological Logic. Technical Report TM-90-04. Kaiserslautern, Germany: Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI).

Baader, Franz, and Bernhard Hollunder. 1991. A Terminological Knowledge Representation System with Complete Inference Algorithm. In *Proc. of the Workshop on Processing Declarative Knowledge, PDK-91*, 67–86. Lecture Notes In Artificial Intelligence, No. 567. Springer-Verlag.

Baader, Franz, and Bernhard Hollunder. 1992. Embedding Defaults into Terminological Knowledge Representation Formalisms. In *Proc. of the 3rd Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-92)*, 306–317. Morgan Kaufmann, Los Altos.

Baader, Franz, and Bernhard Hollunder. 1993. How to Prefer More Specific Defaults in Terminological Default Logic. In *Proc. of the 13th Int. Joint Conf. on Artificial Intelligence (IJCAI-93)*, 669–674. Chambery, France. Morgan Kaufmann, Los Altos.

Baader, Franz, Bernhard Hollunder, Bernhard Nebel, Hans-Jürgen Profitlich, and Enrico Franconi. 1992. An Empirical Analisys of Optimization Techniques for Terminological Representation Systems. In *Proc. of the 3rd Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-92)*, 270–281. Morgan Kaufmann, Los Altos.

Baader, Franz, Maurizio Lenzerini, Werner Nutt, and Peter F. Patel-Schneider (ed.). 1994. *Working Notes of the 1994 Description Logics Workshop*. Bonn, Germany. Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI), D-94-10.

Beck, H. W., S. K. Gala, and S. B. Navathe. 1989. Classification as a Query Processing Technique in the CANDIDE Semantic Data Model. In *Proc. of the 5th IEEE Int. Conf. on Data Engineering (ICDE-89)*.

Bell, John L., and Moshe Machover. 1977. *A Course in Mathematical Logic*. North-Holland Publ. Co., Amsterdam.

Bergamaschi, Sonia, and Bernhard Nebel. 1993. Acquisition and Validation of Complex Object Database Schemata Supporting Multiple Inheritance. *Applied Intelligence* 4(2):185–203.

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

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

Borgida, Alexander, Ronald J. Brachman, Deborah L. McGuinness, and Lori Alperin Resnick. 1989. CLASSIC: A Structural Data Model for Objects. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, 59–67.

Borgida, Alexander, Maurizio Lenzerini, Daniele Nardi, and Bernhard Nebel (ed.). 1995. *Working Notes of the 1995 Description Logics Workshop*. Rome, Italy. Technical Report, Università di Roma "La Sapienza", Dipartimento di Informatica e Sistemistica, RAP 07.95.

Borgida, Alexander, and 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.

Brachman, Ronald J. 1979. On the Epistemological Status of Semantic Networks. In *Associative Networks*, ed. Nicholas V. Findler. 3–50. Academic Press.

Brachman, Ronald J. 1992. "Reducing" CLASSIC to Practice: Knowledge Representation Meets Reality. In *Proc. of the 3rd Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-92)*, 247–258. Morgan Kaufmann, Los Altos.

Brachman, Ronald J., and Hector J. Levesque. 1984. The Tractability of Subsumption in Frame-Based Description Languages. In *Proc. of the 4th Nat. Conf. on Artificial Intelligence (AAAI-84)*, 34–37.

Brachman, Ronald J., and Hector J. Levesque. 1985. *Readings in Knowledge Representation*. Morgan Kaufmann, Los Altos.

Brachman, Ronald J., Victoria Pigman Gilbert, and Hector J. Levesque. 1985. An Essential Hybrid Reasoning System: Knowledge and Symbol Level Accounts in KRYPTON. In *Proc. of the 9th Int. Joint Conf. on Artificial Intelligence (IJCAI-85)*, 532–539. Los Angeles.

Brachman, Ronald J., and James G. Schmolze. 1985. An Overview of the KL-ONE Knowledge Representation System. *Cognitive Science* 9(2):171–216.

Buchheit, Martin, Francesco M. Donini, Werner Nutt, and Andrea Schaerf. 1994a. Terminological Systems Revisited: Terminology = Schema + Views. In *Proc. of the 12th Nat. Conf. on Artificial Intelligence (AAAI-94)*, 199–204. Seattle, USA.

Buchheit, Martin, Francesco M. Donini, Werner Nutt, and Andrea Schaerf. 1995. A Refined Architecture for Terminological Systems: Terminology = Schema + Views. Technical Report RR-95-09. Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI).

Buchheit, Martin, Francesco M. Donini, and Andrea Schaerf. 1993. Decidable Reasoning in Terminological Knowledge Representation Systems. *Journal of Artificial Intelligence Research* 1:109–138.

Buchheit, Martin, Manfred A. Jeusfeld, Werner Nutt, and Martin Staudt. 1994b. Subsumption between Queries to Object-Oriented Databases. *Information Systems* 19(1):33–54. Special issue on Extending Database Technology, EDBT'94.

Calvanese, Diego, Maurizio Lenzerini, and Daniele Nardi. 1994. A Unified Framework for Class Based Representation Formalisms. In *Proc. of the 4th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-94)*, ed. J. Doyle, E. Sandewall, and P. Torasso, 109–120. Bonn. Morgan Kaufmann, Los Altos.

Cohen, William W., and Haym Hirsh. 1994. Learning the CLASSIC Description Logics: Theorethcal and Experimental Results. In *Proc. of the 4th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-94)*, ed.

J. Doyle, E. Sandewall, and P. Torasso, 121–133. Bonn. Morgan Kaufmann, Los Altos.

De Giacomo, Giuseppe, and Maurizio Lenzerini. 1994a. Boosting the Correspondence between Description Logics and Propositional Dynamic Logics. In *Proc. of the 12th Nat. Conf. on Artificial Intelligence (AAAI-94)*, 205–212. AAAI Press/The MIT Press.

De Giacomo, Giuseppe, and Maurizio Lenzerini. 1994b. Concept Language with Number Restrictions and Fixpoints, and its Relationship with μ-Calculus. In *Proc. of the 11th European Conf. on Artificial Intelligence (ECAI-94)*, 411–415.

Devambu, Premkumar, and Diane Litman. 1991. Plan-Based Terminological Reasoning. In *Proc. of the 2nd Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-91)*, ed. James Allen, Richard Fikes, and Erik Sandewall, 128–138. Morgan Kaufmann, Los Altos.

Donini, Francesco M., Bernhard Hollunder, Maurizio Lenzerini, Alberto Marchetti Spaccamela, Daniele Nardi, and Werner Nutt. 1992a. The Complexity of Existential Quantification in Concept Languages. *Artificial Intelligence Journal* 2–3:309–327.

Donini, Francesco M., Maurizio Lenzerini, Daniele Nardi, and Werner Nutt. 1991a. The Complexity of Concept Languages. In *Proc. of the 2nd Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-91)*, ed. James Allen, Richard Fikes, and Erik Sandewall, 151–162. Morgan Kaufmann, Los Altos.

Donini, Francesco M., Maurizio Lenzerini, Daniele Nardi, and Werner Nutt. 1991b. Tractable Concept Languages. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI-91)*, 458–463. Sydney.

Donini, Francesco M., Maurizio Lenzerini, Daniele Nardi, and Werner Nutt. 1995. The Complexity of Concept Languages. Technical Report RR-95-07. Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI).

Donini, Francesco M., Maurizio Lenzerini, Daniele Nardi, Werner Nutt, and Andrea Schaerf. 1992b. Adding Epistemic Operators to Concept Languages. In *Proc. of the 3rd Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-92)*, 342–353. Morgan Kaufmann, Los Altos.

Donini, Francesco M., Maurizio Lenzerini, Daniele Nardi, Werner Nutt, and Andrea Schaerf. 1994a. Queries, Rules and Definitions as Epistemic Sentences in Concept Languages. In *Proc. of the ECAI Workshop on Knowledge Representation and Reasoning*, 113–132. Lecture Notes In Artificial Intelligence, No. 810. Springer-Verlag.

Donini, Francesco M., Maurizio Lenzerini, Daniele Nardi, Werner Nutt, and Andrea Schaerf. 1995a. Adding Epistemic Operators to Concept Languages. Technical report. Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza". In preparation.

Donini, Francesco M., Maurizio Lenzerini, Daniele Nardi, and Andrea Schaerf. 1994b. Deduction in Concept Languages: From Subsumption to Instance Checking. *Journal of Logic and Computation* 4(4):423–452.

Donini, Francesco M., Daniele Nardi, and Riccardo Rosati. 1995b. Non-first-order features in concept languages. In *Proc. of the 4th Conf. of the Italian Association for Artificial Intelligence (AI*IA-95)*, 91–102. Lecture Notes In Artificial Intelligence, No. 992. Springer-Verlag.

Doyle, Jon, and Ramesh S. Patil. 1991. Two Theses of Knowledge Representation: Language Restrictions, Taxonomic Classification, and the utility of representation Services. *Artificial Intelligence Journal* 48:261–297.

Fitting, Melvin. 1990. *First-Order Logic and Automated Theorem Proving.* Springer-Verlag.

Johnson, D. S. 1990. A Catalog of Complexity Classes. In *Handbook of Theoretical Computer Science*, ed. J. van Leeuwen. Chap. 2. Elsevier Science Publishers B. V. (North Holland).

Kaczmarek, Thomas S., Raymond Bates, and Gabriel Robins. 1986. Recent Developments in NIKL. In *Proc. of the 5th Nat. Conf. on Artificial Intelligence (AAAI-86)*, 978–985.

Kindermann, Carsten. 1990. Class Instances in a Terminological Framework–an Experience Report. In *Proc. of the German Workshop on Artificial Intelligence*, 48–57. Springer-Verlag.

Lehmann, Fritz. 1992. Semantic Networks. In *Semantic Networks in Artificial Intelligence*, ed. Fritz Lehmann. 1–50. Pergamon Press.

Lenzerini, Maurizio, and Andrea Schaerf. 1991. Concept Languages as Query Languages. In *Proc. of the 9th Nat. Conf. on Artificial Intelligence (AAAI-91)*, 471–476.

Levesque, Hector J. 1984. Foundations of a Functional Approach to Knowledge Representation. *Artificial Intelligence Journal* 23:155–212.

Levesque, Hector J., and Ron J. Brachman. 1987. Expressiveness and Tractability in Knowledge Representation and Reasoning. *Computational Intelligence* 3:78–93.

Lifschitz, Vladimir. 1991. Nonmonotonic Databases and Epistemic Queries. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI-91)*, 381–386. Sydney.

MacGregor, Robert. 1994. A Description Classifier for the Predicate Calculus. In *Proc. of the 12th Nat. Conf. on Artificial Intelligence (AAAI-94)*. To appear.

MacGregor, Robert, and R. Bates. 1987. The Loom Knowledge Representation Language. Technical Report ISI/RS-87-188. Marina del Rey, Cal.: University of Southern California, Information Science Institute.

MacGregor, Robert, Deborah McGuinness, Eric Mays, and Tom Russ (ed.). 1992. *Working Notes of the AAAI Fall Symposium on Issues on Description Logics: Users meet Developers.* Boston, USA.

McAllester, David. 1991. Unpublished Manuscript.

Minsky, Marvin. 1981. A Framework for Representing Knowledge. In *Mind Design*, ed. J. Haugeland. The MIT Press. A longer version appeared in *The Psychology of Computer Vision* (1975). Republished in Brachman and Levesque 1985.

Mylopoulos, J., P. A. Bernstein, and E. Wong. 1980. A Language Facility for Designing Database-Intensive Applications. *ACM Transactions on Database Systems* 5(2):185–207.

Nebel, Bernhard. 1988. Computational Complexity of Terminological Reasoning in BACK. *Artificial Intelligence Journal* 34(3):371–383.

Nebel, Bernhard. 1990a. *Reasoning and Revision in Hybrid Representation Systems.* Lecture Notes In Artificial Intelligence, No. 422. Springer-Verlag.

Nebel, Bernhard. 1990b. Terminological Reasoning is Inherently Intractable. *Artificial Intelligence Journal* 43:235–249.

Nebel, Bernhard. 1991. Terminological Cycles: Semantics and Computational Properties. In *Principles of Semantic Networks*, ed. John F. Sowa. 331–361. Morgan Kaufmann, Los Altos.

Nebel, Bernhard, Christof Peltason, and Kai von Luck (ed.). 1991. *International Workshop on Terminological Logics.* Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI).

Padgham, Lin, and Bernhard Nebel. 1993. Combining Classification and Non-Monotonic Inheritance Reasoning: A First Step. In *Proc. of the 7th Int. Sym. on Methodologies for Intelligent Systems (ISMIS-93)*, ed. J. Komorowski and Z. W. Raś.

Padgham, Lin, and Tingting Zhang. 1993. A Terminological Logic with Defaults: A Definition and an Application. In *Proc. of the 13th Int. Joint Conf. on Artificial Intelligence (IJCAI-93)*, 662–668. Chambery, France. Morgan Kaufmann, Los Altos.

Patel-Schneider, P. F. 1989. Undecidability of Subsumption in NIKL. *Artificial Intelligence Journal* 39:263–272.

Patel-Schneider, P. F., and Bill Swartout. 1993. Working Version (Draft): Description Logic Specification from the KRSS Effort, June. Unpublished Manuscript.

Patel-Schneider, Peter F. 1984. Small Can Be Beautiful in Knowledge Representation. In *Proc. of the IEEE Workshop on Knowledge-Based Systems.* An extended version appeared as Fairchild Tech. Rep. 660 and FLAIR Tech. Rep. 37, October 1984.

Patel-Schneider, Peter F., Ronald J. Brachman, and Hector J. Levesque. 1984. ARGON: Knowledge Representation Meets Information Retrieval. In *Proc. of the 1st Conf. on Artificial Inteligence Applications.* Also available as Fairchild Technical Report 654 and FLAIR Technical Report 29.

Quantz, Joachim, and Carsten Kindermann. 1990. Implementation of the BACK System Version 4. Technical Report KIT-Report 78. Berlin, Germany: FB Informatik, Technische Universität Berlin.

Quantz, Joachim, and Veronique Royer. 1992. A Preference Semantics for Defaults in Terminological Logics. In *Proc. of the 3rd Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-92)*, 294–305. Morgan Kaufmann, Los Altos.

Quillian, M. Ross. 1967. Word Concepts: A Theory and Simulation of Some Basic Capabilities. *Behavioral Science* 12:410–430. Republished in Brachman and

Levesque 1985.

Reiter, R. 1990. On Asking What a Database Knows. In *Symposium on Computational Logics*, ed. J. W. Lloyd, 96–113. Springer-Verlag, ESPRIT Basic Research Action Series.

Rich, Charles (ed). 1991. Special Issue on Implemented Knowledge Representation and Reasoning Systems. *SIGART Bulletin* 2(3).

Salomone, Sandro. 1992. An $O(n \log n)$ algorithm for Subsumption in $\mathcal{FL}^-$. In *Proceedings of the Fourth Italian Conference on Theoretical Computer Science*, ed. A. Marchetti Spaccamela, P. Mentrasti, and M. Venturini Zilli, 125–139. World Scientific Publishing Co., October.

Schaerf, Andrea. 1993. On the Complexity of the Instance Checking Problem in Concept Languages with Existential Quantification. *Journal of Intelligent Information Systems* 2:265–278.

Schaerf, Andrea. 1994a. *Query Answering in Concept-Based Knowledge Representation Systems: Algorithms, Complexity, and Semantic Issues.* Doctoral dissertation, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza".

Schaerf, Andrea. 1994b. Reasoning with Individuals in Concept Languages. *Data and Knowledge Engineering* 13(2):141–176.

Schild, Klaus. 1988. Undecidability of Subsumption in $\mathcal{U}$. Technical Report KIT-Report 67. Berlin, Germany: FB Informatik, Technische Universität Berlin.

Schild, Klaus. 1991. A Correspondence Theory for Terminological Logics: Preliminary Report. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI-91)*, 466–471. Sydney.

Schild, Klaus. 1994. Terminological Cycles and the Propositional μ-Calculus. In *Proc. of the 4th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-94)*, ed. J. Doyle, E. Sandewall, and P. Torasso, 509–520. Bonn. Morgan Kaufmann, Los Altos.

Schmidt-Schauß, Manfred. 1989. Subsumption in KL-ONE is Undecidable. In *Proc. of the 1st Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-89)*, ed. Ron J. Brachman, Hector J. Levesque, and Ray Reiter, 421–431. Morgan Kaufmann, Los Altos.

Schmidt-Schauß, Manfred, and Gert Smolka. 1991. Attributive Concept Descriptions with Complements. *Artificial Intelligence Journal* 48(1):1–26.

Straccia, Umberto. 1993. Default Inheritance Reasoning in Hybrid KL-ONE-style Logics. In *Proc. of the 13th Int. Joint Conf. on Artificial Intelligence (IJCAI-93)*, 676–681. Chambery, France. Morgan Kaufmann, Los Altos.

Vardi, Moshe Y. 1982. The Complexity of Relational Query Languages. In *Proc. of the 14th ACM SIGACT Sym. on Theory of Computing (STOC-82)*, 137–146.

Weida, Robert, and Diane Litman. 1992. Terminological Reasoning with Constraint Networks and an Application to Plan Recognition. In *Proc. of the 3rd Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-92)*, 282–293. Morgan Kaufmann, Los Altos.

Woods, William A. 1975. What's in a Link: Foundations for semantic Networks. In *Representation and Understanding: Studies in Cognitive Science*, ed. D. G. Bobrow and A. M. Collins. 35–82. Academic Press. Republished in Brachman and Levesque 1985.

Woods, William A. 1991. Understanding Subsumption and Taxomony: A Framework for Progress. In *Principles of Semantic Networks*, ed. J. F. Sowa. 45–94. Morgan Kaufmann, Los Altos.

Woods, William A., and James G. Schmolze. 1992. The KL-ONE Family. In *Semantic Networks in Artificial Intelligence*, ed. F. W. Lehmann. 133–178. Pergamon Press. Published as a special issue of *Computers & Mathematics with Applications*, Volume 23, Number 2–9.

Wright, Jon R., Elia S. Weixelbaum, Gregg T. Vesonder, Karen E. Brown, Stephen R. Palmer, Jay I. Berman, and Harry H. Moore. 1993. A Knowledge-Based Configurator that Supports Sales, Engineering, and Manufacturing at AT&T Network Systems. *AI Magazine* 14(3):69–80.

Yen, John, Robert Neches, and Robert MacGregor. 1991. CLASP: Integrating Term Subsumption Systems and Production Systems. *IEEE Transactions on Knowledge and Data Engineering* 3(1):25–31.