INFORMATION AND COMPUTATION **134**, 1–58 (1997)
ARTICLE NO. IC972625

The Complexity of Concept Languages*

Francesco M. Donini, Maurizio Lenzerini, and Daniele Nardi

Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza," via Salaria 113, I-00198 Rome, Italy
E-mail: {donini,lenzerini,nardi}@dis.uniroma1.it

and

Werner Nutt†

German Research Center for Artificial Intelligence, DFKI GmbH, Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany
E-mail: nutt@dfki.uni-sb.de

A basic feature of Terminological Knowledge Representation Systems is to represent knowledge by means of taxonomies, here called terminologies, and to provide a specialized reasoning engine to do inferences on these structures. The taxonomy is built through a representation language called a *concept language* (or *description logic*), which is given a well-defined set-theoretic semantics. The efficiency of reasoning has often been advocated as a primary motivation for the use of such systems. The main contributions of the paper are: (1) a complexity analysis of concept satisfiability and subsumption for a wide class of concept languages; (2) algorithms for these inferences that comply with the worst-case complexity of the reasoning task they perform.

1. INTRODUCTION

Among computer systems based on artificial intelligence technologies, the distinguishing feature of knowledge-based systems (KBS) is that knowledge is explicitly represented, via a suitable language, and that new knowledge can be inferred from the existing one by an inference engine, tailored to the representation language employed: e.g., a forward reasoner for rules, or a classifier for taxonomies.

The information stored in language expressions, plus the inference engine, is usually considered as a knowledge representation system (KRS) in its own rights, which can be regarded as the core of a KBS. The communication between the KRS

* This is an extended and revised version of a paper presented at the 2nd International Conference on Principles of Knowledge Representation and Reasoning, Cambridge, MA, 1991.

† Address from February 15, 1997, to January 31, 1998: Institute of Computer Science, Ross Building, Givat Ram Campus, Hebrew University of Jerusalem, Jerusalem 91904, Israel.

0890-5401/97 $25.00

and the rest of the KBS is realized via queries and answers to queries. The type of language used to represent knowledge, and the inferences drawn from it characterize the KRS (Levesque, 1984).

In this paper we are concerned with terminological knowledge representation systems (TKRSs), whose basic feature is to represent knowledge by means of taxonomies, here called *terminologies*, and to provide a specialized reasoning engine to do inferences on these structures. Such TKRSs have their roots in the structured inheritance networks proposed by Brachman and realized in the KL-ONE system (Brachman, 1979). This novel paradigm was motivated by the shortcomings of semantic networks, which have been criticized for their lack of *formal* semantics, which results in ambiguity and contradictions (Woods, 1975; Brachman 1985). During the last ten years, this line of research has led to the development of various TKRSs (see for example (Woods and Schmolze, 1992; SIGART, 1991; Baader *et al.*, 1994)). These systems make a clear distinction between intensional knowledge, or general knowledge about the problem domain, and extensional knowledge, which is specific to a particular problem. Typically, the intensional knowledge takes the form of a taxonomy, which is built through a representation language that is called *concept language* (or *description logic*), and is given well-defined set-theoretic semantics. The efficiency of reasoning has often been advocated as a primary motivation for the use of such systems. Among the consequences of the formalization of the representation language, there is the possibility of studying the computational properties of inference by means of a complexity analysis, as first proposed in (Brachman and Levesque, 1984). Deduction methods and computational properties of reasoning problems in concept languages are the subject of this paper. We use standard notions from complexity theory (see for example (Johnson, 1990)). In this Introduction we first describe concept languages and the reasoning services of a TKRS informally; then we illustrate the motivations and main results of the paper.

1.1. *Concepts as Structured Set Descriptions*

At the heart of a TKRS is a *concept language* for specifying concepts, which are general descriptions of the classes in the domain of interest. A concept expression is formed by means of several constructors, some of which specify relationships to other concepts (role links). Role links can be qualified in various ways (e.g., value restrictions, number restrictions, co-references). By analyzing concept expressions the system organizes concepts into a hierarchy according to their specificity. This hierarchical structure is the basis for the realization of reasoning services, such as inheritance computation. Depending on which constructors are allowed, different structures can be built. Such structures can also be described pictorially (as in early proposals of semantic networks), however in the paper we almost always use language expressions.

The basic building blocks of concept languages are atomic concepts, which can be thought of as unary predicates, and atomic roles, which can be thought of as binary predicates. They are combined to build complex concepts and roles using constructors that are characteristic of the language employed. Concept languages

are given a Tarski-style semantics where concepts are interpreted as subsets of a domain and roles as binary relations (cf. Section 2).

For the sake of example, let us suppose that Female, Person, and Woman are atomic concepts and that has-child and has-female-relative are atomic roles. Using the operators *intersection*, *union*, and *complement*, interpreted as their set theoretical counterparts, we can describe the class of "persons that are not female" and the class of "individuals that are female or male" by the expressions

$$\mathsf{Person} \sqcap \neg \mathsf{Female} \qquad \text{and} \qquad \mathsf{Female} \sqcup \mathsf{Male}.$$

An alternative notation for the first concept, proposed in (Patel–Schneider and Swartout, 1993), is (AND Person (NOT Female)).

Most languages provide *existential* and *universal quantification over roles* that allow one to describe the classes of "individuals having a female child" and of "individuals whose children are all female" by the concepts

$$\exists \mathsf{has\text{-}child}.\mathsf{Female} \qquad \text{and} \qquad \forall \mathsf{has\text{-}child}.\mathsf{Female}.$$

In a KL-ONE-like notation (Brachman and Schmolze, 1985), the latter concept corresponds to a link via the role has-child, and a value restriction to Female of this link.

Number restrictions on roles denote classes of individuals having at least or at most a certain number of fillers for a role. For instance, the concept

$$(\geqslant 3\ \mathsf{has\text{-}child}) \sqcap (\leqslant 2\ \mathsf{has\text{-}female\text{-}relative})$$

represents the class of "individuals having at least three children and at most two female relatives."

Role intersection is a role forming construct. Intuitively, has-child $\sqcap$ has-female-relative yields the role "daughter," so the concept

$$\mathsf{Woman} \sqcap (\leqslant 2\ (\mathsf{has\text{-}child} \sqcap \mathsf{has\text{-}female\text{-}relative})$$

denotes the class of "women having at most two daughters."

In this paper we consider a family of concept languages that we call $\mathscr{AL}$-languages and that are defined as extensions of a simple language called $\mathscr{AL}$, where concepts are formed using intersection, complement, universal role quantification, unqualified existential quantification (written $\exists R.\top$, which simply requires the existence of some role filler) and primitive complement (i.e., complement only applied to atomic concepts). The languages of the $\mathscr{AL}$-family are then obtained by extending the constructs of $\mathscr{AL}$ with all possible combinations of union, qualified existential role quantification, number restrictions and role conjunction.

1.2. *Reasoning about Concept Taxonomies*

In a TKRS, the user can give names to concept expressions and use these names inside other concept expressions, thus specifying a *terminology*. This name comes

historically from the use of concept names as *terms* in the language of the external KBS. The module of the KRS containing the terminology is known as terminological box, or simply TBox (Brachman *et al.*, 1983).

As an example, we discuss a terminology whose pictorial representation is given in Fig. 1. Here, we have the concept Parent defined by

$$\mathsf{Parent} := \mathsf{Person} \sqcap \exists \mathsf{has\text{-}child}.\mathsf{Person} \sqcap \forall \mathsf{has\text{-}child}.\mathsf{Person}.$$

"A parent is a person having at least one child, who is a person, and all of his/her children are persons." When a concept is defined as an intersection of some other concepts, it inherits the properties of these: for example, the concept

$$\mathsf{Mother} := \mathsf{Female} \sqcap \mathsf{Parent}$$

inherits from Parent the links to Person through the role has-child (i.e., also mothers are persons, have at least one child, who is a person, and all of their children are persons).

Observe that there may be implicit inclusions between concepts. For example, if we consider

$$\mathsf{Woman} := \mathsf{Person} \sqcap \mathsf{Female}$$

we have that every Mother is a Woman. Such dependencies (usually more complex than this one) should be detected automatically by the KRS.

The inference engine that is necessary to build and to use the terminology relies on two basic inferences:

- *concept satisfiability*, which is the problem of checking whether a concept expression does not always denote the empty concept;
- *subsumption*, which is the problem of checking whether one concept (the *subsumer*, e.g., Woman in Fig. 1) is considered more general than a second one (the *subsumee*, e.g., Mother in Fig. 1); in other words, subsumption checks whether the first concept always denotes a superset of the set denoted by the second.

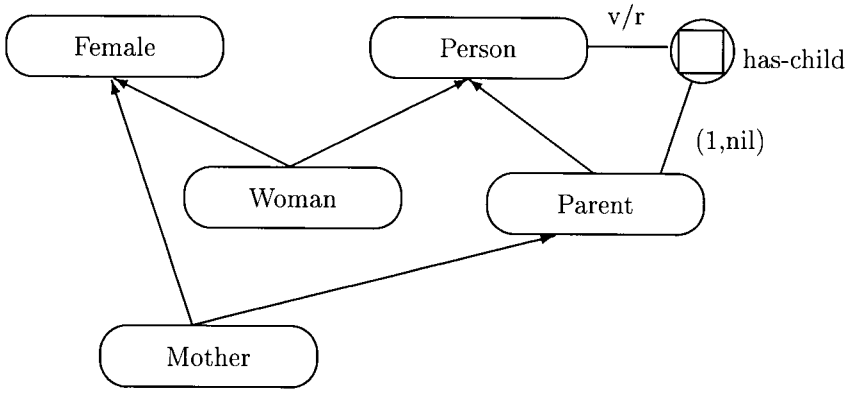

FIG. 1. A KL-ONE-like picture of an example terminology.

In fact, concept satisfiability is a special case of nonsubsumption (with the second concept being the empty concept) and therefore subsumption is the basic reasoning task to be accomplished by the TKRS.

In theory, terminologies play a role when looking at complexity issues, since Nebel (Nebel, 1990) showed that expanding definitions gives rise to an unavoidable source of complexity. In practice, however, definitions increasing the complexity of reasoning do not occur. Moreover, a common requirement on terminologies is that the meaning of a concept "can be completely understood in terms of the meaning of its parts and the way these are composed" (Schmolze and Brachman, 1982). Consequently, systems usually admit only terminologies that satisfy the following conditions:

- there is no more than one definition for a concept name;
- the definitions are *acyclic* in the sense that concepts are neither defined in terms of themselves nor in terms of other concepts that refer to them via a chain of definitions.

In this case, every defined concept can be expanded in a unique way into a complex concept containing only atomic concepts by replacing every defined concept with the right-hand side of its definition. Under these assumptions the computational complexity of inferences can be studied by abstracting from the terminology and by considering all given concepts as fully expanded expressions. Therefore, in this paper we are concerned with reasoning on concept expressions and, in particular, with concept satisfiability and subsumption.

1.3. *Motivation and Scope*

Although in early work the meaning of concepts had already been specified with a logical semantics, the design of inference procedures was influenced for a long time by the tradition of semantic networks, where concepts were viewed as nodes and roles as edges in a graph. Subsumption had been recognized as the key inference and the basic idea of the first subsumption algorithms was to transform two input concepts into labeled graphs and test whether one could be embedded into the other; the embedded graph would correspond to the more general concept (the subsumer) (see (Lipkis, 1982)). This method is called *structural comparison*, and the relation it computes between concepts is called *structural subsumption*. However, a careful analysis of the algorithms for structural subsumption shows that they are *sound* but not always *complete* in terms of the logical semantics: whenever they return "yes" the answer is correct, but when they report "no" the answer may be incorrect. In other words, structural subsumption is in general weaker than logical subsumption.

The study of sound and complete algorithms for reasoning in concept languages and the systematic use of complexity analysis to characterize their computational properties originated with the influential paper by Brachman and Levesque (Brachman and Levesque, 1984). They provided a complete polynomial algorithm for a very limited language, called $\mathcal{FL}^-$, and showed that for the seemingly

slightly more expressive language $\mathcal{FL}$ subsumption is co-NP-hard (for a definition of $\mathcal{FL}^-$ and $\mathcal{FL}$ see Subsection 4.3). Nebel (Nebel, 1988) identified other constructs that give rise to co-NP-hard subsumption problems. Other work identified languages with undecidable subsumption problem (Patel-Schneider, 1989; Schmidt-Schauß, 1989; Schild, 1988). However, neither (Brachman and Levesque, 1984; Levesque and Brachman, 1987) nor (Nebel, 1988) give algorithms for the co-NP-hard languages.

The first nonstructural complete subsumption algorithm was devised by Schmidt-Schauß and Smolka (Schmidt-Schauß and Smolka, 1991), for an extension of $\mathcal{FL}$ by a construct for complements of concepts, called $\mathcal{ALC}$. In fact, they changed the paradigm for designing algorithms in that, using complements, they reformulated subsumption problems as satisfiability problems. Perceiving satisfiability of concepts as a constraint-solving problem, they developed a calculus on constraints by which satisfiability—and thus subsumption—of concepts in $\mathcal{ALC}$ can be decided. Moreover, they proved satisfiability and subsumption in $\mathcal{ALC}$ to be PSPACE-complete and identified a sublanguage with a co-NP-complete unsatisfiability problem.

The aim of the paper is to provide a complete computational analysis of the inference problems for concept description languages obtained by combining the most frequently used concept forming constructs. Therefore, the paper ideally fulfills one of the goals of the research started after Brachman and Levesque's paper (Brachman and Levesque, 1984), namely determining the computational complexity of inference in a concept language characterized in terms of its concept-forming constructs.

The results of our study have both theoretical and practical significance. First of all, the study of the computational behavior of concept languages has led to a clear understanding of the properties of the language constructs and their interaction. This is not only valuable from a theoretical viewpoint, but gives insights to the designer of a TKRS (see for example Brachman, 1992)), with clear indications of the language constructs or their combinations that are difficult to deal with and general methods to cope with them.

Second, the complexity results have been obtained by exploiting a general technique for satisfiability checking, which relies on a form of tableaux calculus that 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 the language constructs. For many $\mathcal{AL}$-languages, the algorithms for concept satisfiability and subsumption obtained from the calculus on constraints represent the only known complete deduction procedures. They have been the basis of actual implementations (Baader and Hollunder, 1991; Baader *et al.*, 1994) where they have been tuned by special control strategies.

Third, the analysis of intractable cases has led to discover cases of incompleteness of the algorithms developed for implemented systems, and can be used in the definition of worst-case test sets for verifying implementations. For example, the comparison of implemented systems described in (Heinsohn *et al.*, 1994) has benefited from the results of the complexity analysis.

There are at least two other areas of research that have a relationship with concept description languages and the associated inference problems: modal logic and query containment in databases.

Schild (Schild, 1991) pointed out that certain concept languages are notational variants of certain propositional modal logics. Among the languages we consider, $\mathcal{ALC}$ has a modal-logic counterpart, namely the multimodal version of the logic K (see Halpern and Moses, 1992)). Actually, $\mathcal{ALC}$-concepts and formulas in multimodal K can immediately be translated into each other. Moreover, a concept is satisfiable if and only if the corresponding formula is satisfiable. Research in the complexity of the satisfiability problem for modal propositional logics started a long time before the complexity of concept languages was investigated. Ladner (Ladner, 1977) showed that satisfiability in unimodal K—a logic whose counterpart is the set of all $\mathcal{ALC}$-concepts with at most one role symbol—is PSPACE-complete. Interestingly, in order to prove membership in PSPACE, Ladner, too, exhibited a tableaux-like algorithm. Halpern and Moses (Halpern and Moses, 1992) showed, again using a tableaux calculus, that in multimodal K satisfiability of formulas can also be decided with polynomial space. However, the results we found on the complexity of reasoning in concept languages are new if translated into the framework of modal logic. In particular, the language $\mathcal{ALE}$ ($\mathcal{AL}$ plus existential quantification on roles) corresponds to the fragment of multimodal K consisting of those formulas whose negation normal form does not contain disjunction. Number restrictions correspond to a numerically parameterized modal operator, and our results can be transferred to reasoning in such non-standard modal logics (de Rijke and van der Hoek, 1995).

With regard to query containment in databases, we observe that the main difference between database query languages and concept languages is that concept languages have a variable-free syntax. This, on one hand, limits the expressiveness of concept languages as query languages. In fact, database query languages may create arbitrary coreferences between different parts of a query by using the same variable name. On the other hand, query containment in relational query languages is undecidable (Kanellakis, 1990, p. 1083), while subsumption is decidable for most concept languages (see results in this paper). It is interesting to observe that subsumption is undecidable in concept languages with a construct expressing arbitrary coreferences (Patel-Schneider, 1989; Schmidt-Schauß, 1989).

1.4. *Main Results and Organization of the Paper*

The goal of the paper is to provide algorithms that perform various kinds of reasoning about concepts and to study the complexity of such tasks. To this end we introduce the family of $\mathcal{AL}$-languages as a ramework for investigating how the interplay of the most common constructs in concept languages affects the complexity of reasoning and how to devise adequate inference techniques. Each element in the $\mathcal{AL}$-family is obtained by adding to the core language $\mathcal{AL}$ a combination of the four constructs union, full existential quantification, number restriction, and role intersection. Therefore, we obtain 16 different concept languages.

The main contributions of the paper are:

1. a complexity analysis of concept satisfiability and subsumption in $\mathcal{AL}$-languages;

2. the algorithms for these inferences that comply with the worst-case complexity of the reasoning task they perform.

For our work we employ a calculus of logical constraints that has first been introduced in (Hollunder and Nutt, 1990) and that significantly extends the one in (Schmidt-Schauß and Smolka, 1991). In fact, it covers a more expressive language, and it employs a more concise notation, which points out its similarity to the tableaux calculus for first-order predicate logic. We use the calculus in two ways. The analysis of its behavior leads to proofs of worst-case complexity. In addition, we obtain our algorithms by modifying the rules of the calculus and submitting them to a suitable control strategy.

Since concept satisfiability is a special case of subumption, the complexity analysis is accomplished as follows:

1. For each $\mathcal{AL}$-language we give an upper bound on the complexity of subsumption—and hence of unsatisfiability—by proving that it is in one of the classes P, NP, co-NP, or PSPACE. For a given language, the proof consists in devising an algorithm that complies with the resource bounds by which the respective class is defined.

2. For the languages where subsumption is not polynomial, we give a lower bound for the complexity of unsatisfiability—and hence of subsumption—by proving that is hard for one of the classes NP, co-NP, or PSPACE.

With the exception of one language, upper and lower bounds coincide. This means that from the viewpoint of worst-case complexity our algorithms are optimal for the problems they solve.

The paper is organized as follows. In Section 2, we formally introduce syntax and semantics of $\mathcal{AL}$-languages, and define the properties of concepts we want to infer. In Section 3, we introduce the calculus and summarize the results concerning its soundness and completeness. In Sections 4 top 7 we discuss the computational properties of $\mathcal{AL}$-languages, structuring the presentation according to complexity classes. Thus we have sections on languages for which reasoning is PSPACE-complete, NP-complete, co-NP-complete, or polynomial. In Section 8 we summarize our complexity results, and Section 9 concludes the paper.

2. CONCEPT LANGUAGES AND THEIR INFERENCE PROBLEMS

In this section, we introduce the syntax and semantics of the $\mathcal{AL}$-family of concept languages, discuss their relationship to first-order predicate logic, and formally define the kinds of inference to be performed on concepts. For the syntax of concepts, we employ a notation that is similar to that in (Schmidt-Schauß and Smolka, 1991). The specification of the semantics follows that in (Brachman and Levesque, 1984, Schmidt-Schauß and Smolka, 1991).

2.1. *Syntax and Semantics of Concepts*

In the simplest $\mathscr{AL}$-language, called $\mathscr{AL}$, *concepts* (denoted by C, D) are built out of *atomic concepts* (denoted by A) and *atomic roles* (denoted by P) according to the syntax rule

$$\begin{array}{rcll} C, D & \rightarrow & A \mid & \text{(atomic concept)} \\ & & \top \mid & \text{(universal concept)} \\ & & \bot \mid & \text{(empty concept)} \\ & & \neg A \mid & \text{(atomic negation)} \\ & & C \sqcap D \mid & \text{(intersection)} \\ & & \forall R.C \mid & \text{(universal role quantification)} \\ & & \exists R.\top & \text{(restricted existential role quantification),} \end{array}$$

where R denotes a *role*, which in $\mathscr{AL}$ is always atomic (more general languages provide a constructor for role intersection). A *subconcept* of a concept C is a substring of C that is a concept.

An *interpretation* $\mathscr{I} = (\Delta^{\mathscr{I}}, \cdot^{\mathscr{I}})$ consists of a set $\Delta^{\mathscr{I}}$ (the *domain* of $\mathscr{I}$) and a function $\cdot^{\mathscr{I}}$ (the *interpretation function* of $\mathscr{I}$) that maps every concept to a subset of $\Delta^{\mathscr{I}}$ and every role to a subset of $\Delta^{\mathscr{I}} \times \Delta^{\mathscr{I}}$ such that

$$\begin{aligned} \top^{\mathscr{I}} &= \Delta^{\mathscr{I}} \\ \bot^{\mathscr{I}} &= \varnothing \\ (C \sqcap D)^{\mathscr{I}} &= C^{\mathscr{I}} \cap D^{\mathscr{I}} \\ (\neg A)^{\mathscr{I}} &= \Delta^{\mathscr{I}} \backslash A^{\mathscr{I}} \\ (\forall R.C)^{\mathscr{I}} &= \{a \in \Delta^{\mathscr{I}} \mid \forall b.\, (a, b) \in R^{\mathscr{I}} \rightarrow b \in C^{\mathscr{I}}\} \\ (\exists R.\top)^{\mathscr{I}} &= \{a \in \Delta^{\mathscr{I}} \mid \exists b.\, (a, b) \in R^{\mathscr{I}}\}. \end{aligned}$$

Observe that an interpretation function is already determined by the way it interprets atomic concepts and roles.

More general languages are obtained by adding to $\mathscr{AL}$ the following constructors:

- *union* of concepts (indicated by the letter $\mathscr{U}$), written as $C \sqcup D$, and interpreted as

$$(C \sqcup D)^{\mathscr{I}} = C^{\mathscr{I}} \cup D^{\mathscr{I}};$$

- *full existential quantification* (indicated by the letter $\mathscr{E}$), written as $\exists R.C$, and interpreted as

$$(\exists R.C)^{\mathscr{I}} = \{a \in \Delta^{\mathscr{I}} \mid \exists b.\ (a, b) \in R^{\mathscr{I}} \wedge b \in C^{\mathscr{I}}\}$$

(note that $\exists R.C$ differs from $\exists R.\top$ in that arbitrary concepts are allowed to occur in the scope of the existential quantifier);

• *complement* of nonatomic concepts (indicated by the letter $\mathscr{C}$), written as $\neg C$, and interpreted as

$$(\neg C)^{\mathscr{I}} = \Delta^{\mathscr{I}} \setminus C^{\mathscr{I}};$$

• *number restrictions* (indicated by the letter $\mathscr{N}$), written as $(\geqslant nR)$ and $(\leqslant nR)$, where n ranges over the nonnegative integers, and interpreted as

$$(\geqslant nR)^{\mathscr{I}} = \{a \in \Delta^{\mathscr{I}} \mid \text{card}\{b \mid (a, b) \in R^{\mathscr{I}}\} \geqslant n\},$$

and

$$(\geqslant nR)^{\mathscr{I}} = \{a \in \Delta^{\mathscr{I}} \mid \text{card}\{b \mid (a, b) \in R^{\mathscr{I}}\} \geqslant n\},$$

respectively, where card $\cdot$ denotes the cardinality of sets;[1]

• *intersection of roles* (indicated by the letter $\mathscr{R}$), written as $Q \sqcap R$, where Q and R are arbitrary roles, and interpreted as

$$(Q \sqcap R)^{\mathscr{I}} = Q^{\mathscr{I}} \cap R^{\mathscr{I}}.$$

Extending $\mathscr{AL}$ by a subset of the above constructs yields a particular $\mathscr{AL}$-language. We name each $\mathscr{AL}$-language by a string of the form

$$\mathscr{AL}[\mathscr{U}][\mathscr{E}][\mathscr{N}][\mathscr{R}],$$

where a letter in the name stands for the presence of the corresponding construct. For instance, $\mathscr{ALNR}$ is the extension of $\mathscr{AL}$ by number restrictions and intersection of roles.

Observe that the semantics enforces the following equivalences (among others): $C \sqcup D \equiv \neg(\neg C \sqcap \neg D)$, $\exists R.C \equiv \neg\forall R.\neg C$. Hence, union and full existential quantification can be expressed using complements, and conversely, the combination of union and full existential quantification gives us the possibility of expressing complements of concepts (through their equivalent negation normal form, see Section 3.1). Therefore, without loss of generality we assume that union and full existential quantification are available in every language that contains complements, and vice versa, and in language names we use the letter $\mathscr{C}$ instead of $\mathscr{UE}$, e.g., $\mathscr{ALUEN} = \mathscr{ALCN}$. It follows that there are sixteen pairwise nonequivalent $\mathscr{AL}$-languages, which form a lattice, whose bottom element is $\mathscr{AL}$ and whose top element is $\mathscr{ALCNR}$.

Throughout the paper, if no particular language is specified, we refer to $\mathscr{ALCNR}$-concepts simply as "concepts."

[1] From a semantic viewpoint, the actual coding of n is immaterial; however, since the coding does matter in complexity analysis, we will specify in each case whether we assume that n is coded in unary form (i.e., the integer n is represented by a string of length n) or not.

In addition to $\mathscr{AL}$-languages, in Section 4.3 we consider also the concept language $\mathscr{FL}$ (Brachman and Levesque, 1984), whose concepts can be expressed as particular $\mathscr{ALCNR}$-concepts.

2.2. *Concepts as Predicate Logic Formulas*

The semantics of concepts identifies concept languages as fragments of first-order predicate logic. Since an interpretation $\mathscr{I}$ assigns to every atomic concept or role a unary or binary relation over $\Delta^{\mathscr{I}}$, respectively, one can think of atomic concepts and roles as unary and binary predicates. Then, any concept expression C can be translated effectively into a predicate logic formula $\phi_C(x)$, which has one free variable, such that for every interpretation $\mathscr{I}$ the set of elements of $\Delta^{\mathscr{I}}$ satisfying $\phi_C(x)$ is just $C^{\mathscr{I}}$. This can easily be seen as follows: an atomic concept A is translated into the formula $A(x)$; the concept forming constructs intersection, union, and negation are expressed through conjunction, disjunction, and negation, respectively; if C is already translated into $\phi_C(x)$ and R is an atomic role, then role quantification and number restrictions are captured by the formulas

$$
\begin{aligned}
\phi_{\exists R.C}(x) &= \exists y.\, R(x, y) \wedge \phi_C(y) \\
\phi_{\forall R.C}(x) &= \forall y.\, R(x, y) \rightarrow \phi_C(y) \\
\phi_{(\geqslant nR)}(x) &= \exists y_1, \ldots, y_n.\left(\bigwedge_{i \neq j} y_i \neq y_j\right) \wedge R(x, y_1) \wedge \cdots \wedge R(x, y_n) \\
\phi_{(\leqslant nR)}(x) &= \forall y_1, \ldots, y_{n+1}.\, R(x, y_1) \wedge \cdots \wedge R(x, y_{n+1}) \rightarrow \bigvee_{i \neq j} y_i \doteq y_j;
\end{aligned}
$$

finally, if $R = P_1 \sqcap \cdots \sqcap P_k$ is the intersection of the atomic roles $P_1, \ldots, P_k$, each occurrence of an atom $R(x, y)$ in the above formulas has to be replaced with $P_1(x, y) \wedge \cdots \wedge P_k(x, y)$.

Note that equality "$\doteq$" between variables is needed to express number restrictions, while concepts without number restrictions can be translated into equality-free formulas. We shall see this reflected in our calculus.

Although concepts can be translated into predicate logic in principle, the variable-free syntax of concept expressions is more readable and lends itself more easily to the development of algorithms. Moreover, characterizing syntactically which first-order formulas express concepts of a given language, and which do not, would lead to unnatural descriptions. Therefore, the use of a concept-language syntax seems to us preferrable.

2.3. *Inferences with Concepts*

As argued in Section 1, a main reasoning task with concepts is the check for satisfiability, i.e., the test whether a concept can be interpreted as a nonempty set. Other possible types of inferences are checking whether, given two concepts, the sets denoted by them are always the same, always disjoint, or whether one is always a subset of the other. The semantics of concepts allows us to define them formally as follows:

Satisfiability. A concept C is *satisfiable* if there exists an interpretation $\mathcal{I}$ such that $C^{\mathcal{I}}$ is nonempty. In this case we say that $\mathcal{I}$ is a *model* of C.

Subsumption. A concept C is *subsumed* by a concept D if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for every interpretation $\mathcal{I}$.

Equivalence. Two concepts C and D are *equivalent* if $C^{\mathcal{I}} = D^{\mathcal{I}}$ for every interpretation $\mathcal{I}$. In this case we write $C \equiv D$.

Disjointness. Two concepts C and D are *disjoint* if $C^{\mathcal{I}} \cap D^{\mathcal{I}} = \varnothing$ for every interpretation $\mathcal{I}$.

Using the complement of concepts, we can reduce subsumption, equivalence, and disjointness of concepts to satisfiability problems (see also (Smolka, 1988)):

PROPOSITION 2.1. *Let C and D be concepts. Then*:

1. *C is subsumed by D iff $C \sqcap D$ is not satisfiable*;
2. *C and D are equivalent iff $(C \sqcap \neg D)$ and $(\neg C \sqcap D)$ are not satisfiable*;
3. *C and D are disjoint iff $C \sqcap D$ is not satisfiable.*

Because of the above proposition, it is sufficient to develop algorithms that decide the satisfiability of concepts if one is interested in decision procedures for any of the other three inferences. However, when studying the complexity of the above inferences in a particular $\mathcal{AL}$-language, it is not sufficient to restrict oneself to satisfiability, since subsumption and equivalence problems for a language without full complement give rise to satisfiability problems for concepts not contained in the language. Nonetheless, we show later on that for such languages too, algorithms obtained through this approach are optimal with respect to the complexity of the problem they solve. In this paper, we will only be interested in the complexity of satisfiability and of subsumption, since from the viewpoint of worst-case complexity these are the most specific and the most general kind of inferences, respectively. In fact, as stated more precisely in the following propositions, all problems can be rephrased as subsumption problems, while unsatisfiability is a special case of each problem.

PROPOSITION 2.2 (Reduction to Subsumption). *Let C, D be concepts. Then*

1. *C is unsatisfiable iff C is subsumed by $\bot$*;
2. *C and D are equivalent iff C is subsumed by D and D is subsumed by C*;
3. *C and D are disjoint iff $C \sqcap D$ is subsumed by $\bot$.*

COROLLARY 2.3. *For each $\mathcal{AL}$-language, an upper bound for the complexity of the subsumption problem yields an upper bound for the complexity of the unsatisfiability, the equivalence, and the disjointness problem.*

Hence, whenever we prove that subsumption between concepts of a given $\mathcal{AL}$-language is a problem in P, NP, co-NP, or PSPACE, then so are the other problems.

PROPOSITION 2.4 (Reducing Unsatisfiability). *Let C be a concept. Then the following are equivalent*:

1. *C is unsatisfiable*;
2. *C is subsumed by* $\bot$;
3. *C and* $\bot$ *are equivalent*;
4. *C and* $\top$ *are disjoint.*

COROLLARY 2.5. *For each $\mathcal{AL}$-language, a lower bound for the unsatisfiability problem yields a lower bound for the subsumption, the equivalence, and the disjointness problem.*

Hence, whenever we prove that the unsatisfiability problem is NP-hard, co-NP-hard, or PSPACE-hard, then so are the other problems.

3. A CALCULUS FOR CHECKING SATISFIABILITY

In this section we introduce a calculus for checking the satisfiability of concepts. The calculus appeared first in (Hollunder *et al.*, 1990) and is discussed in full detail in (Hollunder and Nutt, 1995). It is influenced by the *completion calculus* in (Schmidt-Schauß and Smolka, 1991), but employs a much simpler notation that emphasizes its similarity to the tableaux calculus for first-order predicate logic (see (Smullyan, 1968; Bell and Machover, 1977; Fitting, 1990)).

The calculus consists of inference rules that decompose complex concepts according to the top-level construct. Actually, the rules can be simulated by applications of several rules of the tableaux calculus using a special control strategy. The strategy is essential to guarantee termination, i.e., to exclude infinite chains of rule applications.

The data structures underlying our calculus are *constraints* which state either (1) that an individual is a member of a concept, or (2) that two individuals are related through a role, or (3) that two individuals are distinct. The rules operate on sets of constraints that are understood as conjunctions of their elements. If we start with a set of constraints corresponding to a concept C, every derivation terminates after finitely many steps. If all terminal sets of constraints that are derivable contain an "obvious contradiction"—a notion that has to be defined precisely—then C is unsatisfiable. Otherwise we can conclude that C is satisfiable, since a terminal set that is free of obvious contradictions describes a model of C.

To keep the number of inference rules small we assume that concepts are in a normal form, which is similar to negation normal form of logical formulas. This normal form is described in Subsection 3.1. Then, in Subsection 3.2 we introduce the data structures—sets of constraints—and the rules. In Subsection 3.3 we review the soundness and completeness of the calculus, and in Subsection 3.4 we discuss its computational properties.

3.1. *Normal Forms of Concepts*

We say a concept is *in negation normal form* or *simple* if it contains only complements of the form $\neg A$ where A is a primitive concept. Arbitrary concepts can be rewritten to simple concepts by the following equivalence-preserving rules:

$$
\begin{array}{rcl}
\neg\top & \rightarrow & \bot \\
\neg\bot & \rightarrow & \top \\
\neg(C \sqcap D) & \rightarrow & \neg C \sqcup \neg D \\
\neg(C \sqcup D) & \rightarrow & \neg C \sqcap \neg D \\
\neg\neg C & \rightarrow & C \\
\neg(\forall R.C) & \rightarrow & \exists R.\neg C \\
\neg(\exists R.C) & \rightarrow & \forall R.\neg C \\
\neg(\leqslant nR) & \rightarrow & (\geqslant n+1 R) \\
\neg(\geqslant nR) & \rightarrow & \begin{cases} \forall R.\bot & \text{if } n=1 \\ (\leqslant n-1\ R) & \text{if } n>1. \end{cases}
\end{array}
$$

If C' is a simple concept that has been obtained from C using the above rules, then we say that C' is *the* negation normal form of C. The idea to consider only concepts in negation normal form appeared before in (Smolka, 1988; Schmidt-Schauß and Smolka, 1991).

PROPOSITION 3.1. *For any concept one can compute in linear time its negation normal form, which is equivalent to the original concept.*

3.2. *The Completion Rules*

Now we introduce the expressions on which our calculus operates. We assume that there exists an alphabet of variable symbols (ranged over by x, y, z). A *constraint* (ranged over by c) is a syntactic object of one of the forms

$$x : C, \qquad xPy, \qquad x \not\doteq y,$$

where C is a simple concept and P is a primitive role. Intuitively, $x : C$ says that x has to be interpreted as an element of C, xPy says that x and y have to be interpreted as individuals related by P, and $x \not\doteq y$ that x and y have to be interpreted as distinct individuals. Note that we do not have constraints for equality, but only for inequality, although the logical formulas corresponding to concepts with number restrictions contain both positive *and* negated equations. The reason is that our calculus will implicitly take into account the fact that number restrictions force two variables to be equal by substituting one variable for the other. Observe also that we allow complex concepts in constraints, but only primitive roles. Since every role is an intersection of finitely many primitive roles, it is not necessary to allow constraints of the form xRy, where $R = P_1 \sqcap \cdots \sqcap P_k$. We will use instead the constraints $xP_1\, y$, $xP_2\, y$, ..., $xP_k\, y$.

We extend the semantics of concepts to constraints. Let $\mathcal{I}$ be an interpretation. An *$\mathcal{I}$-assignment* is a function α that maps every variable to an element of $\Delta^{\mathcal{I}}$. We say that α *satisfies*

$$x : C \quad \text{if} \quad \alpha(x) \in C^{\mathcal{I}},$$
$$xPy \quad \text{if} \quad (\alpha(x), \alpha(y)) \in P^{\mathcal{I}},$$
$$x \neq y \quad \text{if} \quad \alpha(x) \neq \alpha(y).$$

A constraint c is *satisfiable* if there are an interpretation $\mathcal{I}$ and an $\mathcal{I}$-assignment α such that α satisfies c. A *constraint system* S is a finite, nonempty set of constraints. An $\mathcal{I}$-assignment α *satisfies* a constraint system S if α satisfies every constraint in S. A constraint system S is *satisfiable* if there is an interpretation $\mathcal{I}$ and an $\mathcal{I}$-assignment α such that α satisfies S.

The following proposition is an immediate consequence of the above definitions.

PROPOSITION 3.2. *A simple concept C is satisfiable if and only if the constraint system $\{x : C\}$ is satisfiable.*

Let us introduce some notation to help us specify the rules of the calculus. Let S be a constraint system and $R = P_1 \sqcap \cdots \sqcap P_k$ be a role. We say that y is an *R-successor of x in S* if $xP_1 y, ..., xP_k y$ are in S. We say that y is a *successor* of x in S if for some role R y is an R-successor of x. If S is clear from the context we simply say that y is an R-successor or a successor of x. With $S[y/z]$ we denote the constraint system obtained from S by replacing each occurrence of y by z. We say that x and y are *separated in S* if the constraint $x \neq y$ is in S.

The *completion calculus* is given by the rules in Fig. 2. Note that there is a rule for every concept-forming construct, except for complement. Since concepts occurring in constraints are supposed to be in negation normal form, we do not need rules that handle concepts with complement as outermost symbol.

The rules are designed in such a way that, intuitively, their application can be understood as an attempt to construct a model by generating new individuals as required by the constraints. In this sense, they will be used to *complete* an initial system $\{x : C\}$ by additional constraints. Note that the rule conditions guarantee that a rule can be applied to a constraint system only if its application changes the system. In the following, we discuss the action of the rules in more detail.

The $\rightarrow_{\sqcap}$-rule and the $\rightarrow_{\sqcup}$-rule decompose constraints of the form x: $C_1 \sqcap C_2$, and x: $C_1 \sqcup C_2$, respectively.

A constraint $x : \exists R.C$ states that x has an R-successor which is in the interpretation of C. To satisfy this constraint, if such an R-successor does not yet exist, the $\rightarrow_{\exists}$-rule generates a a new R-successor of x and constrains it to C.

The $\rightarrow_{\forall}$-rule is applied to a combination of constraints of the form $x : \forall R.C$ and $xP_1 y, ..., xP_k y$ where $R = P_1 \sqcap \cdots \sqcap P_k$. It is different from the analogous tableaux rule for universally quantified formulas, which allows one to instantiate a universally quantified variable by an arbitrary term, in that it is only applied to R-successors of x. If one applied the tableaux calculus to a formula corresponding to a concept without this restriction, one could generate nonterminating derivations.

Intuitively, a constraint $x : (\geqslant n R)$ requires x to have at least n distinct R-successors. By application of the $\rightarrow_{\geqslant}$-rule this condition is satisfied in that n new R-successors of x are generated together with additional constraints stating that they are pairwise distinct.

Intersection

$(\rightarrow_{\sqcap})$ $S \rightarrow_{\sqcap} \{x{:}\,C_1,\ x{:}\,C_2\} \cup S$

if $x{:}\,C_1 \sqcap C_2$ is in S, and $x{:}\,C_1$ and $x{:}\,C_2$ are not both in S

Union

$(\rightarrow_{\sqcup})$ $S \rightarrow_{\sqcup} \{x{:}\,D\} \cup S$

if $x{:}\,C_1 \sqcup C_2$ is in S, neither $x{:}\,C_1$ nor $x{:}\,C_2$ is in S,
and $D = C_1$ or $D = C_2$

Existential Quantification

$(\rightarrow_{\exists})$ $S \rightarrow_{\exists} \{xP_1y, \ldots, xP_ky,\ y{:}\,C\} \cup S$

if $x{:}\,\exists R.C$ is in S, $R = P_1 \sqcap \cdots \sqcap P_k$,
there is no z such that z is an R-successor of x and $z{:}\,C$ is in S,
and y is a new variable

Universal Quantification

$(\rightarrow_{\forall})$ $S \rightarrow_{\forall} \{y{:}\,C\} \cup S$

if $x{:}\,\forall R.C$ is in S, y is an R-successor of x in S,
and $y{:}\,C$ is not in S

At-least Restriction

$(\rightarrow_{\geq})$ $S \rightarrow_{\geq} \{xP_1y_i, \ldots, xP_ky_i \mid i \in 1..n\} \cup \{y_i \not\equiv y_j \mid i, j \in 1..n,\ i \neq j\} \cup S$

if $x{:}\,(\geq n\, R)$ is in S, $R = P_1 \sqcap \cdots \sqcap P_k$,
the number of R-successors of x in S is less than n,
and $y_1, \ldots, y_n$ are new variables

At-most Restriction

$(\rightarrow_{\leq})$ $S \rightarrow_{\leq} S[y/z]$

if $x{:}\,(\leq n\, R)$ is in S, x has more than n R-successors in S,
and y, z are two R-successors of x that are not separated

FIG. 2. The completion rules.

If a constraint system S contains the constraint $x : (\leqslant nR)$ and there are more than n R-successors of x then every assignment satisfying S must interpret at least two R-successors by the same individual. The $\rightarrow_{\leqslant}$-rule takes this fact into account by nondeterministically identifying two R-successors that are not constrained to be distinct.

EXAMPLE 3.3. Consider the concept $C := \exists P.\neg A \sqcap \forall P.(A \sqcup B)$, which is simple. Applying the completion rules, one can obtain the following constraint systems from $S_0 := \{x : C\}$:

$$\begin{aligned} S_0 \quad &\rightarrow_{\sqcap} \quad S_1 = S_0 \cup \{x : \exists P.\neg A, x : \forall P.(A \sqcup B)\} \\ &\rightarrow_{\exists} \quad S_2 = S_1 \cup \{xPy, y : \neg A\} \\ &\rightarrow_{\forall} \quad S_3 = S_2 \cup \{y : (A \sqcup B)\} \\ &\rightarrow_{\sqcup} \quad S_4 = S_3 \cup \{y : A\}. \end{aligned}$$

Alternatively, one can apply the $\rightarrow_{\sqcup}$-rule such that

$$S_3 \quad \rightarrow_{\sqcup} \quad S_3 \cup \{y : B\}.$$

We distinguish two kinds of completion rules, the *deterministic* rules $\rightarrow_{\sqcap}$, $\rightarrow_{\exists}$, $\rightarrow_{\forall}$, and $\rightarrow_{\geqslant}$, and the *nondeterministic* rules $\rightarrow_{\sqcup}$ and $\rightarrow_{\leqslant}$. The nondeterministic rules handle concept-forming constructs that contain disjunction when translated into predicate logic. Obviously, the union of concepts is the analogue of the disjunction of formulas. Moreover, also the formula $\phi_{(\leqslant nR)}(x)$ contains disjunction, as shown in Subsection 2.2.

The following proposition shows that the completion rules keep the satisfiability of constraint systems invariant. The proof follows straightforwardly from the soundness of the tableaux calculus (see (Nutt, 1993)).

THEOREM 3.4. (Invariance). *Let S and S' be constraint systems.*

1. *If S' is obtained from S by application of a deterministic rule, then S is satisfiable if and only if S' is satisfiable.*

2. *If S' is obtained from S by application of a nondeterministic rule then S is satisfiable if S' is satisfiable. Conversely, if S is satisfiable and a nondeterministic rule is applicable to a constraint c in S, then it can be applied to c in such a way that it yields a satisfiable constraint system.*

3.3. *Soundness and Completeness*

In order to check with the completion calculus whether a simple concept C is satisfiable one starts with a constraint system $\{x : C\}$ and adds new constraints by application of the completion rules until no more rules apply. Up to variable renaming, only finitely many such systems—which we call *complete*—can be derived. These systems can easily be checked for satisfiability by looking for certain obvious contradictions, called *clashes*, that will be defined later on. If each of the derived systems contains a clash, then C is unsatisfiable; otherwise it is satisfiable.

The calculus is *sound*, in the sense that if a clash-free complete constraint system can be derived from $\{x : C\}$, then $\{x : C\}$ is satisfiable, and that it is *complete* in

the sense that if $\{x : C\}$ is satisfiable, then a complete system without clash can be derived. The soundness is shown by proving that a complete system without clash is satisfiable, which immediately yields the claim, since $\{x : C\}$ is contained in any complete system derived from it. Completeness is shown by proving that there is no infinite chain of rule applications issuing from $\{x : C\}$, and then using the Invariance Theorem 3.4. Below we outline the proofs and describe the underlying ideas. The full proofs are given in (Nutt, 1993).

A constraint system is *complete* if no completion rule applies to it. A complete system that has been derived from a system S is also called a *completion* of S. A *clash* is a constraint system having one of the following forms:

- $\{x : \bot\}$;
- $\{x : A, x : \neg A\}$, where A is a primitive concept;
- $\{x : (\leqslant nR)\} \cup \{xP_1\, y_i, ..., xP_k\, y_i \mid i \in 1..n+1\}$
  $\cup \{y_i \neq y_j \mid i, j \in 1..(n+1), i \neq j\}$,

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

The third form of clash comprises those systems where a variable x is constrained to have at most n fillers for a role, but the system contains $n+1$ successors which are explicitly stated to be distinct.

Clashes allow one to identify unsatisfiable constraint systems.

PROPOSITION 3.5. *If a constraint system contains a clash, then it is not satisfiable.*

The notion of clash can be used to characterize satisfiable complete constraint systems.

PROPOSITION 3.6. *A complete constraint system is satisfiable if and only if it does not contain a clash.*

The "only if" part of the above proposition follows from Proposition 3.5. The "if" part is proved by constructing from a complete constraint system S an interpretation $\mathcal{I}$ and an $\mathcal{I}$-assignment α such that α satisfies S. The domain $\Delta^{\mathcal{I}}$ is the set of all variables appearing in the constraint system, and the interpretation is defined on the basis of all constraints in S of the form $x : A$ and xPy involving only atomic concepts and roles.

EXAMPLE 3.7. From the clash-free complete constraint system S_4 of Example 3.3 we construct the following interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}}) : \Delta^{\mathcal{I}} = \{x, y\}$, $A^{\mathcal{I}} = \{y\}$, $B^{\mathcal{I}} = \varnothing$, and $P^{\mathcal{I}} = \{(x, y)\}$. The $\mathcal{I}$-assignment α is defined by $\alpha(x) := x$ and $\alpha(y) := y$. It is easy to check that α satisfies every constraint in S_4.

The above propositions imply the soundness of the completion calculus.

THEOREM 3.8 (Soundness). *A simple concept C is satisfiable if there exists a clash-free completion of $\{x : C\}$.*

Our next goal is to show the completeness of the calculus. To this end, we first prove the termination of the completion calculus, which is done by exhibiting a suitable termination order (see (Nutt, 1993)).

PROPOSITION 3.9 (Termination). *Let C be a simple concept. Then there is no infinite chain of completion steps issuing from $\{x : C\}$.*

THEOREM 3.10 (Completeness). *Let C be a simple concept. If C is satisfiable, then from $\{x : C\}$ one can derive with finitely many steps a complete constraint system that does not contain a clash.*

Proof. By the Invariance Theorem 3.4, there is a chain of completion steps issuing from $\{x : C\}$ that (1) preserves satisfiability and (2) is either infinite or ends with a complete system. By Proposition 3.9 this chain cannot be infinite. Since the last constraint system in this chain is satisfiable and complete, we know by Proposition 3.6 that it does not contain a clash. ∎

On the basis of the above results, it is straightforward to turn the calculus into a decision procedure. In order to check a concept C for satisfiability, one transforms it into its negation normal form C' and then generates all complete constraint systems derivable from $\{x : C'\}$, which are, up to variable renaming, finitely many. If all these systems contain a clash, then C is unsatisfiable, otherwise it is satisfiable.

THEOREM 3.11 (Decidability). *Satisfiability of $\mathcal{ALCNR}$-concepts is decidable.*

Since a concept C is subsumed by a concept D if and only if $C \sqcap \neg D$ is unsatisfiable, the calculus can be applied to subsumption checking as well.

COROLLARY 3.12. *Subsumption of $\mathcal{ALCNR}$-concepts is decidable.*

3.4. *The Computational Properties of Constraint Systems*

In this section we discuss the computational properties of the completion calculus outlined in the previous section, and we show how they can be used to highlight the computational properties of deduction problems for $\mathcal{AL}$-languages.

We have shown how to check concept satisfiability and subsumption by nondeterministically computing completions of constraint systems. The deterministic version of this process can be seen as the process of generating all possible completions by applying the completion rules, and verifying, for every generated completion, if it contains a clash. Hence, the computational complexity of the whole method depends on the following factors:

1. the complexity of selecting a rule and applying it;
2. the complexity of checking for the presence of a clash in a completion;
3. the number of different completions to be generated;
4. the number of rule applications leading to a single completion.

In what follows, we suppose that the numbers occurring in number restrictions are coded in unary notation. From the form of the completion rules, one can see that the time for selecting and applying a rule to a constraint system S is polynomial with respect to the size of S. Observe that without the assumption on the unary coding of numbers, this statement would not hold for the $\rightarrow_{\geqslant}$-rule and the $\rightarrow_{\leqslant}$-rule.

Also, checking whether a complete constraint system S contains a clash can be achieved in time polynomial in the size of S. This is obvious for the first two kinds

of clashes. To see that it is also true for the third kind of clash, it is sufficient to observe that a complete system S contains such a clash, involving, say, a constraint $x : (\leqslant nR)$, if and only if x has at least $n+1$ R-successors in S. Observe that for *complete* systems, we do not need to check that the $n+1$ successors are pairwise separated—they must be so, otherwise the $\rightarrow_{\leqslant}$-rule would be applicable, and S would not be complete. Instead, for arbitrary constraint systems, deciding the presence of a clash of this kind is NP-hard, since deciding whether a graph has a clique of size at least $n+1$ can be easily reduced to clash checking.

Therefore the crucial parameters for measuring the complexity of the method are the number of different completions, and the number of different rule applications leading to a single completion. Indeed, it is easy to see that the application of a rule can have a twofold effect:

1. Increasing the number of different completions to be inspected; this happens when several applications of a nondeterministic rule are possible.

2. Increasing the size of the constraint system; this happens in the applications of all the rules except for the $\rightarrow_{\leqslant}$-rule.

We first discuss the increase in the number of completions, and then turn to the increase in the size of a single completion.

The different choices of applying a nondeterministic rule (namely, $\rightarrow_{\sqcup}$ and $\rightarrow_{\leqslant}$) to a constraint lead to different completions. For example, each application of the $\rightarrow_{\sqcup}$-rule to a constraint $x: C_1 \sqcup C_2$ nondeterministically chooses between two different constraint systems, one containing $x : C_1$ and the other containing $x : C_2$. In general, exponentially many different completions may be generated due to applications of the $\rightarrow_{\sqcup}$-rule. Therefore, we can identify a first source of exponential complexity (or simply complexity) in the calculus, caused by the presence of the union construct in a concept language. Recall from Section 2.2 that also the $(\leqslant nR)$ construct implicitly contains a disjunction.

We now turn our attention to the size of each completion. The completion rules can be classified into generating rules and non-generating rules. The $\rightarrow_{\exists}$-rule and the $\rightarrow_{\geqslant}$-rule are called *generating* rules, because they introduce new variables, whereas the other rules ($\rightarrow_{\sqcap}$, $\rightarrow_{\sqcup}$, $\rightarrow_{\forall}$, $\rightarrow_{\leqslant}$) do not, and are therefore called *nongenerating* rules. Intuitively, the latter are harmless with respect to the size of the resulting constraint system. Indeed, given a constraint system S, each non-generating rule may only introduce new constraints $y : D$ such that the variable y is already present in S and D is a subconcept of a concept appearing in S. Since a concept has only a number of subconcepts that is linear with respect to its size, the size of a constraint system obtained by exhaustive application of non-generating rules is polynomial with respect to the size of S.

Conversely, consider the constraint system $\{x : \exists P.C_1,\ x : \exists P.C_2,\ x : \forall P.C_3\}$. The application of the $\rightarrow_{\exists}$-rule adds the constraints xPy, $y : C_1$, xRz, $z : C_2$. To this system, the $\rightarrow_{\forall}$-rule is applicable twice, adding $y : C_3$, $z : C_3$. Observe that now the concept C_3 occurs twice in the resulting constraint system. Generalizing this argument, one can show that the completion of the constraint system $\{x : C\}$, where C is the concept

$$
\begin{aligned}
&\exists R_1 . C_{11} \sqcap \\
&\exists R_1 . C_{12} \sqcap \\
&\forall R_1 . (\exists R_2 . C_{21} \sqcap \\
&\qquad \exists R_2 . C_{22} \sqcap \\
&\qquad \forall R_2 . (\ldots (\exists R_n . C_{n1} \sqcap \\
&\qquad\qquad \exists R_n . C_{n2} \sqcap \\
&\qquad\qquad \forall R_n . D) \ldots)),
\end{aligned}
$$

contains distinct variables $y_1, \ldots, y_{2^n}$ and one constraint of the form $y_i : D$ for every $i \in 1..2^n$. Hence, generating rules may produce completions whose size is exponentially bigger than the one of the initial constraint system. Therefore, we can identify in the calculus a second source of complexity, which is caused by those constructs in a concept language that are treated by the generating rules, namely the constructs $\exists R . C$ and $(\geqslant n R)$.

A natural question arises: *are the two sources of complexity just properties of the calculus, or do they single out inherent properties of the deduction problems associated with concept languages*?

One of the main achievements of the research presented in this paper is to provide the answer to this question. In Section 1.3 we mentioned some previous results about the computational complexity of satisfiability and subsumption. Let us briefly review these results in the light of the two sources of complexity.

In (Schmidt-Schauß and Smolka, 1991), concept satisfiability in $\mathcal{ALC}$ was proved to be PSPACE-complete. The language $\mathcal{ALC}$ contains both union and qualified existential quantification. The PSPACE-hardness result can be intuitively explained with the fact that both sources of complexity are present in the language, and this gives rise both to an exponential number of completions and to an exponential size of each completion in the worst case. Nevertheless, in (Schmidt-Schauß and Smolka, 1991) it was shown that the calculus can work by keeping in memory just a polynomial portion of a completion. This observation yields a polynomial-space algorithm for checking the satisfiability of $\mathcal{ALC}$-concepts. Subsumption can be computed by reducing it to unsatisfiability: C is subsumed by D if and only if the $\mathcal{ALC}$-concept $C \sqcap \neg D$ is unsatisfiable. Hence subsumption too is PSPACE-complete.

In (Donini *et al.*, 1992), it was shown that unsatisfiability and subsumption in the language $\mathcal{ALE}$, obtained by adding qualified existential quantification to $\mathcal{AL}$, is NP-complete, thus proving that existential quantification indeed represents a source of complexity in the language. We will show in Section 4.4 that qualified existential quantification can appear implicitly through combinations of other constructs, like existential quantification and conjunction of roles.

The union construct as a source of complexity has been studied in (Schmidt-Schauß and Smolka, 1991), where the language $\mathcal{ALU}$—obtained by adding union to $\mathcal{AL}$—was investigated. Any propositional formula can be translated into a corresponding $\mathcal{ALU}$-concept that is satisfiable if and only if the propositional formula is. Hence, satisfiability in $\mathcal{ALU}$ is NP-hard. Contrast this result with the

one on $\mathcal{ALE}$: there, *unsatisfiability* is NP-hard—hence satisfiability is co-NP-hard—while here *satisfiability* is NP-hard. Note that disjunction can appear also implicitly through combinations of other constructs, as shown in (Nebel, 1988) for the language $\mathcal{ALNR}$.

Finally, the language $\mathcal{AL}$, which does not include any of the two sources of complexity, was investigated in (Schmidt-Schauß and Smolka, 1991), and it was proved that satisfiability in this language can be checked by means of a polynomial time algorithm.

Taking into account these results about the computational complexity of concept languages, we can now classify the $\mathcal{AL}$-languages according to the presence or absence of each source of complexity, and by this classification give a road map to the rest of the paper.

The topmost language $\mathcal{ALCNR}$, being a superlanguage of $\mathcal{ALC}$ contains both sources of complexity. Therefore, concept satisfiability is PSPACE-hard. In Section 4, we will exhibit a satisfiability algorithm working in polynomial space, thus matching the PSPACE-hardness lower bound of $\mathcal{ALC}$, which proves that satisfiability in $\mathcal{ALCNR}$ is PSPACE-complete. Subsumption can be computed by reducing it to unsatisfiability, and therefore it is PSPACE-complete too. In Section 4, we also deal with several other sublanguages of $\mathcal{ALCNR}$ whose inference problems are PSPACE-complete. With a slight abuse of complexity terminology, we call these languages "PSPACE-complete languages."

In Section 5, we consider languages that are affected only by the source of complexity caused by qualified existential quantification, and where unsatisfiability can be verified by generating just a polynomial number of completions. For these languages, unsatisfiability is NP-hard.

Subsumption can be rephrased in terms of unsatisfiability: C is subsumed by D if and only if $C \sqcap \neg D$ is unsatisfiable. However, this leads to a concept expression that does not belong to the initial class of languages, because $\neg D$—once it is rewritten in negation normal form—may in general contain unions. Nevertheless, such unions can be treated *ad hoc* in nondeterministic polynomial time, and therefore constraint systems again provide an optimal NP upper bound for subsumption in these languages. We refer to these languages as "NP-complete languages," since subsumption and satisfiability are NP-complete.

In Section 6, we address languages that are affected only by the other source of complexity—the one related to disjunction. Unsatisfiability is co-NP-hard in these languages. Again, reducing subsumption between C and D to unsatisfiability of $C \sqcap \neg D$ leads to a concept that does not belong to the initial class of languages, because $\neg D$ may in general contain qualified existential quantification. Nevertheless, we prove that in this context existential quantification does not lead to completions of exponential size. Based on this property, we show how nonsubsumption can be verified by a nondeterministic polynomial-time algorithm that guesses a clash-free completion of polynomial size, proving that subsumption is in co-NP. We refer to these languages as "co-NP-complete languages," as both unsatisfiability and subsumption are co-NP-complete.

Finally, in Section 7, we deal with languages where none of the two sources of complexity is present, and we devise a calculus that checks the satisfiability

of a concept by constructing a polynomial number of completions, each one of polynomial size. We devise techniques for reducing subsumption to concept unsatisfiability, while still avoiding both sources of complexity. Hence (un)satisfiability and subsumption can be checked in polynomial time for these languages, which we call "polynomial languages."

4. PSPACE-COMPLETE LANGUAGES

In this section we start with a control strategy for the completion calculus that allows one to check satisfiability of $\mathcal{ALCNR}$-concepts using polynomial space. Since $\mathcal{ALCNR}$ is the top element of the lattice of $\mathcal{AL}$-languages, this implies that the satisfiability and subsumption problems are in PSPACE for all $\mathcal{AL}$-languages. Then we will turn to $\mathcal{AL}$-languages for which satisfiability is PSPACE-hard. It has been shown (Schmidt-Schauß and Smolka, 1991) that $\mathcal{ALC}$ has this property. We will identify two other minimal PSPACE-hard $\mathcal{AL}$-languages, namely $\mathcal{ALUR}$ and $\mathcal{ALNR}$. Using an appropriate modification of the proof for $\mathcal{ALC}$, we can also show that subsumption in Brachman and Levesque's language $\mathcal{FL}$ is PSPACE-hard.

Throughout this section we assume that numbers occurring in number restrictions are coded as unary strings.

4.1. *A* PSPACE-*Algorithm for* $\mathcal{ALCNR}$

A straightforward implementation of the calculus developed in the previous section would generate for a given simple $\mathcal{ALCNR}$-concept C all completions of $\{x : C\}$ and would check whether they contain a clash. As pointed out in Section 3.4, even in the case of the smaller language $\mathcal{ALC}$ there may be exponentially many completions and each of them may be of exponential size.

For a more efficient method one therefore has to avoid generating more than one completion at a time and one also has to avoid storing entire completions. In the following we will show that one can apply the rules of the completion calculus in such a way that these requirements are met. To this end we will have a closer look at the structure of constraint systems.

Let S be a constraint system and y, z be variables occurring in S. We say that y is a *predecessor* of z if z is a successor of y.

LEMMA 4.1. *Let C be a simple concept and S be a constraint system derived from $\{x : C\}$. Then the "successor" relation on the variables in S is a tree with root x, that is, each variable in S, except x, has exactly one predecessor and x has no predecessor at all.*

Proof. The proof is by induction over the length of the derivation leading to S. If the last rule applied was the $\rightarrow_{\exists}$- or the $\rightarrow_{\geqslant}$-rule, then new variables have been created that have a unique predecessor. If it was the $\rightarrow_{\leqslant}$-rule then two successors of one variable have been identified. In either case a tree has been transformed into a tree. If any other rule has been applied the "successor" relation has not been affected. Obviously, no derivation step creates a predecessor of the start variable x. ∎

Suppose S has been derived from $\{x : C\}$. A finite sequence $\gamma = y_0 \cdots y_n$ of variables is a *chain in* S if y_i is a successor of y_{i-1} in S for $i \in 1..n$. We say that γ has *starting point* y_0, *end point* y_n, and *length* n. For any variable y occurring in S we say that the *level* of y is n, and write $\lambda(y) = n$, if there is a chain in S with starting point x and end point y. Since the "successor" relation is a tree, every variable has a unique level. It is easy to see that the level of a variable is not affected by applying a completion rule to S.

If D is a concept, then $\sharp D$ denotes the number of symbols occurring in D, i.e., the length of the string D, where primitive concepts and primitive roles are considered as strings of length 1.

LEMMA 4.2. *Let C be a simple concept and S be derived from $\{x : C\}$. Then for every constraint $y : D$ in S we have $\lambda(y) + \sharp D \leqslant \sharp C$.*

Proof. The proof is by induction over the length of derivations and makes a case analysis according to the form of the rule by which $y : D$ has been introduced. ∎

In order to illustrate the idea behind the control structure embodied in our algorithm let us remark two observations about constraint systems.

First, the clashes in a constraint system are "localized": Let us say that a constraint c *depends* on a variable y if c is of the form $y : D$ or yPz or $z \neq z'$, where z and z' are successors of y. Looking at the definition of clashes one realizes that the constraints in a clash always depend on a unique variable. Hence, one can look for clashes independently in the different successors of a variable.

Second, as shown above, the "successor" relation on the variables occurring a constraint system S derived from $\{x : C\}$ is a tree, whose root is the initial variable x. As seen in Subsection 3.4, this tree may be bushy, but it is not deep. More precisely, it follows from Lemma 4.2 that the length of a path in this tree is always bounded by the length of C.

These observations show that it is not necessary to store an entire completion S in order to check it for clashes but it suffices to explore S in such a way that one variable and the constraints depending on it are considered at a time. If the tree of successors of x is traversed depth-first, then one only needs to store constraints depending on a number of variables that is linear in the length of C.

For the simpler setting of checking the satisfiability of $\mathcal{ALC}$-concepts, this idea has been formally captured in (Schmidt-Schauß and Smolka, 1991) by the notion of a "trace." In our case, the situation is complicated by the presence of both number restrictions and role intersections that force us to generate for a given variable more than one successor at at time. We will use traces later on, when dealing with NP-complete languages.

In Fig. 3 we give a functional algorithm that implements a strategy of applying the completion rules along these ideas. It can be regarded as an extension of the algorithm in (Schmidt-Schauß and Smolka, 1991) for checking the satisfiability of $\mathcal{ALC}$-concepts. The function *sat* takes as input a variable y and a constraint system S. In particular, if the simple concept C is to be checked for satisfiability, *sat* is called with arguments x and $\{x : C\}$.

sat: *variable* × *constraint system* → *bool*

$sat(y, S) =$

if S contains a clash

then false

elsif $y{:}\, C \sqcap D \in S$ **and** $y{:}\, C \notin S$ **or** $y{:}\, D \notin S$

then $sat(y, S \cup \{y{:}\, C,\ y{:}\, D\})$

elsif $y{:}\, C \sqcup D \in S$ **and** $y{:}\, C \notin S$ **and** $y{:}\, D \notin S$

then $sat(y, S \cup \{y{:}\, C\})$ **or** $sat(y, S \cup \{y{:}\, D\})$

elsif $y{:}\, \exists R.C \in S$ **and** y has no R-successor z in S with $z{:}\, C \in S$

then $sat(y, S \cup \{yP_1z, \ldots, yP_kz,\ z{:}\, C\})$

where $P_1 \sqcap \cdots \sqcap P_k = R$ and z is a new variable

elsif $y{:}\, \forall R.C \in S$ **and** z is an R-successor of y in S with $z{:}\, C \notin S$

then $sat(y, S \cup \{z{:}\, C\})$

elsif $y{:}\, (\geq n\, R) \in S$

and y has less than n R-successors in S

then $sat(y, S \cup \{yP_1z_i, \ldots, yP_kz_i \mid i \in 1..n\}$

$\cup \{z_i \neq z_j \mid i, j \in 1..n,\ i \neq j\})$

where $P_1 \sqcap \cdots \sqcap P_k = R$ and $z_1, \ldots, z_n$ are new variables

elsif $y{:}\, (\leq n\, R) \in S$ **and** y has more than n R-successors in S

then there exist non-separated R-successors z, z' in S such that

$sat(y, S[z/z'])$

else for all successors z of y

$sat(z, S)$

FIG. 3. A SPACE-Algorithm for $\mathcal{ALCNR}$.

Essentially, *sat* explores an AND–OR graph, where an AND-node corresponds to the (independent) check of all successors of a variable, while an OR-node corresponds to the application of a nondeterministic rule. More precisely, when called with arguments y and S, *sat* first checks whether S contains a clash and if so it returns **false**. If S is clash-free and there is in S a constraint depending on y to which a rule is applicable, then the rule is applied. If the rule is deterministic then it yields one constraint system, say S_0, and *sat* is called again with y and S_0. If it is nondeterministic, then rule application may result in two systems S_1, S_2 as for the $\rightarrow_{\sqcup}$-rule or finitely many systems S_1, ..., S_n as for the $\rightarrow_{\leqslant}$-rule. In such a case, *sat* is called recursively with y and each of the S_i's and returns **true** if one of the

recursive calls returns **true**. For the case of the $\rightarrow_{\leqslant}$-rule this is expressed with the **there exist** construct. If no rule is applicable to a constraint depending on y then for each successor z of y the function *sat* expands the constraints that depend on z. It returns **true** if all these recursive calls return **true**, which is the case, in particular, if there are no successors.

PROPOSITION 4.3. *Let C be a simple concept. Then*:

1. *sat*$(x, \{x : C\})$ *terminates*;
2. *sat*$(x, \{x : C\})$ *returns* **true** *if and only if C is satisfiable.*

Proof. 1. The function *sat* implements a strategy of applying the completion rules. The only difference to the original completion calculus is that *sat* branches for nondeterministic rules and explores all possible outcomes of applying such a rule to a given constraint. As seen above, the number of possible outcomes is always finite. Together with the termination property of the calculus (3.9) this yields termination of *sat* using König's Lemma.

2. The claim follows from the correctness and completeness of the completion calculus if we take into account the following observations. Suppose *sat* has been called with arguments x and $\{x : C\}$, which has led to the recursive call *sat*(y, S). First, if no rule is applicable to a constraint depending on a variable situated on the chain between x and y then rules will not become applicable to such a constraint after working on constraints depending on successors of y. Second, working on the constraints depending on one successor of y will never create constraints depending on a different successor. These two observations justify *sat*'s strategy of never returning to a variable if at some stage of the computation no rule is applicable to a constraint depending on it and of expanding the constraints depending on each successor independently of other successors. ∎

Let us now estimate how much space is needed to execute *sat*, if it is implemented in a straightforward manner. To simplify the discussion, we will assume that *sat* is implemented in a straightforward manner.

Let C be a simple concept of length n. Suppose that *sat* has been started with arguments x and $\{x : C\}$ and is now called for some y and S. Let us find out how often *sat* will apply a rule to constraints depending on y. Using the fact that the number of subconcepts of the initial concept C is bounded by n, one concludes that at any time of the computation there exist at most n constraints of the form $y : D$. Hence the $\rightarrow_{\sqcap}$-, $\rightarrow_{\sqcup}$-, $\rightarrow_{\exists}$-, and the $\rightarrow_{\geqslant}$-rule are applied at most n times to constraints depending on y. Note that only the $\rightarrow_{\exists}$- and the $\rightarrow_{\geqslant}$-rule create successors of y. Since we assume that numbers are represented as unary strings, they generate at most n successors. The $\rightarrow_{\leqslant}$-rule can be applied no more times than there are successors of y, that is, at most n times. The $\rightarrow_{\forall}$-rule can be applied at most once for every constraint of the form $y : \forall R.D$ and every successor of y. Hence, it is applied at most n^2 times. Summing up we get that for a variable y there are at most $2n + n^2$ calls to *sat* with first argument y on the recursion stack. Since the length of a chain of successors is bounded by the size of C, the whole recursion stack contains at most $O(n^3)$ calls.

Next, let us consider how much space is needed to store one call. Since our algorithm expands per level in the "successor" tree only the constraints depending on a single variable, S contains at most n variables per level. Thus, S contains no more than n^2 variables on the whole because there are at most n levels. Since the number of constraints per variable is bounded by the length of C, we conclude that S contains at most n^3 constraints, which implies that the size of S is at most $O(n^4)$.

PROPOSITION 4.4. *For any simple concept C the call $sat(x, \{x : C\})$ can be executed in polynomial space.*

Proof. As seen above, the recursion stack for a call $sat(x, \{x : C\})$ has depth at most $O(n^3)$, where n is the length of C. Since a call covers at most $O(n^4)$ space, $sat(x, \{x : C\})$ can be executed with $O(n^7)$ space. ∎

A much more detailed and implementation-oriented analysis shows that the space needed is in fact $O(n^2)$. In any case, the above analysis also supports the following conclusion.

THEOREM 4.5. *Satisfiability and subsumption of $\mathcal{ALCNR}$-concepts can be decided in polynomial space.*

4.2. *Unsatisfiability in $\mathcal{ALC}$*

As mentioned before, Schmidt-Schauß and Smolka (Schmidt-Schauß and Smolka, 1991) have shown that the unsatisfiability problem in $\mathcal{ALC}$ is PSPACE-hard. From results in later sections of this paper it follows that for the strict sublanguages of $\mathcal{ALC}$, i.e., $\mathcal{ALE}$, $\mathcal{ALU}$, and $\mathcal{AL}$, unsatisfiability and subsumption do not have this property (provided NP is different from PSPACE). In the rest of this section we will also identify $\mathcal{ALUR}$ and $\mathcal{ALNR}$ as minimal $\mathcal{AL}$-languages with PSPACE-hard unsatisfiability problem. We complement the results on $\mathcal{AL}$-languages by a proof that subsumption in Brachman and Levesque's language $\mathcal{FL}$ is PSPACE-hard. The proofs will consist either in appropriate modifications of the reduction that has been used in (Schmidt-Schauß and Smolka, 1991) for $\mathcal{ALC}$ or in a reduction of the satisfiability problem for $\mathcal{ALC}$ itself.

To make the paper self-contained we will start by resuming Schmidt-Schauß and Smolka's reduction for $\mathcal{ALC}$, which consists in associating to each quantified boolean formula an $\mathcal{ALC}$-concept such that the formula is valid if and only if the concept is satisfiable. Then we will show that this reduction can be modified so as to yield PSPACE-hardness of subsumption in $\mathcal{FL}$. The proof for $\mathcal{ALUR}$ will consist in translating $\mathcal{ALC}$-concepts into $\mathcal{ALUR}$-concepts in such a way that satisfiability is preserved. Finally, we will use a reduction of the validity problem for quantified boolean formulas to show PSPACE-hardness of $\mathcal{ALNR}$. In the following we will introduce the validity problem for quantified boolean formulas and show how it can be translated into the satisfiability problem for $\mathcal{ALC}$-concepts.

Quantified Boolean Formulas. We will first introduce quantified boolean formulas in an intuitive way and then give a formalization that is more appropriate for our purposes.

A quantified boolean formula is built up of propositional variables (such as x, y, z) and quantifiers ($\forall, \exists$). It consists of a finite sequence P of quantifications over variables, called the *prefix*, and a propositional logic formula M in conjunctive normal form, called the *matrix*. An example of a quantified boolean formula is $P.M := \forall x\, \exists y.\, (\neg x \vee \neg y) \wedge (x \vee y)$. Quantified boolean formulae are interpreted over the set of boolean values $\mathscr{B} := \{\mathbf{true}, \mathbf{false}\}$ as follows. We transform the matrix M into a formula M' of first order predicate logic by replacing each occurrence of variable x with the atom *is-true*(x). The predicate *is-true* is interpreted over $\mathscr{B}$ in the obvious way. Now, $P.M$ is said to be *valid* if $P.M'$ is valid over $\mathscr{B}$. Deciding the validity of quantified boolean formulas is a PSPACE-complete problem (Garey and Johnson, 1979). The following reformulation of the problem is due to (Schmidt-Schauß and Smolka, 1991).

A *literal* is a nonzero integer. A *clause* is a nonempty finite set N of literals such that $l \in N$ implies $-l \notin N$. A *prefix* from m to n, where m and n are positive integers such that $m \leqslant n$, is a sequence

$$(Q_m m)(Q_{m+1} m+1) \cdots (Q_n n),$$

where each Q_i is either "$\forall$" or "$\exists$". A *quantified boolean formula* is a pair $P.M$, where, for some n, P is a prefix from 1 to n and M is a finite nonempty set of clauses containing only literals between $-n$ and n (called the *matrix* of the clause).

Let P be a prefix from m to n. A *P-assignment* is a mapping

$$\{m, m+1, ..., n\} \to \{\mathbf{true}, \mathbf{false}\}.$$

An assignment α *satisfies* a literal l if $\alpha(l) = \mathbf{true}$ if l is positive and $\alpha(-l) = \mathbf{false}$ if l is negative. An assignment *satisfies* a clause if it satisfies at least one literal of the clause.

Let P be a prefix from m to n. A set $\mathbf{A}$ of P-assignments is *canonical* for P if it satisfies the following conditions:

1. $\mathbf{A}$ is nonempty
2. if $P = (\exists m) P'$, then all assignments of $\mathbf{A}$ agree on m and, if P' is nonempty, $\{\alpha_{|\{m+1,...,n}\} \mid \alpha \in \mathbf{A}\}$ is canonical for P'
3. if $P = (\forall m) P'$, then
   (a) $\mathbf{A}$ contains an assignment that satisfies m and, if P' is nonempty, $\{\alpha_{|\{m+1,...,n\}} \mid \alpha \in \mathbf{A}$ and $\alpha(m) = \mathbf{true}\}$ is canonical for P'
   (b) $\mathbf{A}$ contains an assignment that satisfies $-m$ and, if P' is nonempty, $\{\alpha_{|\{m+1,...,n\}} \mid \alpha \in \mathbf{A}$ and $\alpha(m) = \mathbf{false}\}$ is canonical for P'.

A quantified boolean formula $P.M$ is *valid* if there exists a set $\mathbf{A}$ of P-assignments such that every assignment in $\mathbf{A}$ satisfies every clause of M.

The Reduction to Satisfiability in $\mathscr{ALC}$. Let $P.M$ be a quantified boolean formula, where $P = (Q_1 1) \cdots (Q_m m)$ is a prefix from 1 to m and $M = \{M_1, ..., M_n\}$. Then $P.M$ is translated into an $\mathscr{ALC}$-concept

$$C_{P.M} = D_1 \sqcap C_1^1 \sqcap \cdots \sqcap C_1^n.$$

Let A be a primitive concept and R be a primitive role. The concept D_1 is obtained from the prefix P using the equations

$$D_l := \begin{cases} (\exists R.A) \sqcap (\exists R.\neg A) \sqcap (\forall R.D_{l+1}) & \text{if } Q_l = \forall \\ \exists R.\top \sqcap (\forall R.D_{l+1}) & \text{if } Q_l = \exists \end{cases}$$

for $l \in 1..(m-1)$ and the equation

$$D_m := \begin{cases} (\exists R.A) \sqcap (\exists R.\neg A) & \text{if } Q_m = \forall \\ \exists R.\top & \text{if } Q_m = \exists .^2 \end{cases}$$

The concept C_1^i is obtained from the clause M_i as follows. Let

$$k := \max_{l \in M_i} |l|.$$

Then, for $l \in 1..(k-1)$, one defines

$$C_i^i := \begin{cases} \forall R.(A \sqcup C_{l+1}^i) & \text{if } l \in M_i \\ \forall R.(\neg A \sqcup C_{l+1}^i) & \text{if } -l \in M_i \\ \forall R.C_{l+1}^i & \text{if neither } l \text{ nor } -l \text{ is in } M_i, \end{cases}$$

and one defines

$$C_k^i := \begin{cases} \forall R.A & \text{if } k \in M_i \\ \forall R.\neg A & \text{if } -k \in M_i. \end{cases}$$

We will briefly discuss the idea underlying this construction. The concept $C_{P.M}$ is constructed in such a way that canonical sets of assignments for P satisfying the clauses of M can be translated into clash-free complete constraint systems obtained from $\{x : C_{P.M}\}$ and vice versa. Under this translation a correspondence is established between chains of variables (see Subsection 4.1) of length m with starting point x and assignments for P.

A variable y that occurs at position l in a chain and has the constraint $y : A$ or $y : \neg A$ indicates that the corresponding assignment satisfies the literal l or $-l$, respectively. The structure of the concept C_1^i, which encodes the ith clause, ensures that for every clash free complete constraint system derived from $\{x : C_{P.M}\}$ the assignments corresponding to chains satisfy the clause M_i.

[2] The definitions given in (Schmidt-Schauß and Smolka, 1991) are $D_l = (\exists R.A \sqcup \exists R.\neg A) \sqcap (\forall R.D_{l+1})$ if $Q_l = \exists$, and $D_m = (\exists R.A \sqcup \exists R.\neg A)$ if $Q_m = \exists$. It is easy to see that these definitions are equivalent to ours.

Conversely, one can show that a set of assignments which is canonical for P and whose elements satisfy every clause in M provides guidance for applying the completion rules in such a way that they transform $\{x : C_{P \bullet M}\}$ into a clash free complete constraint system. The following two results have been shown in (Schmidt-Schauß and Smolka, 1991).

LEMMA 4.6 ($\mathcal{ALC}$-Reduction). *A quantified boolean formula $P \bullet M$ is valid if and only if its translation $C_{P \bullet M}$ is satisfiable.*

THEOREM 4.7. *Satisfiability of $\mathcal{ALC}$-concepts is* PSPACE*-hard.*

4.3. *Subsumption in $\mathcal{FL}$*

In their paper (Brachman and Levesque, 1984), which initiated the complexity analysis of concept languages, Brachman and Levesque introduced the language $\mathcal{FL}^-$, which can be thought of as the minimal meaningful concept language, and proved that the subsumption problem in $\mathcal{FL}^-$ can be solved in quadratic time. A seemingly slight extension of $\mathcal{FL}^-$ by so-called role restrictions led to a language, called $\mathcal{FL}$, for which subsumption was shown to be co-NP-hard.

As already pointed out in (Schmidt-Schauß and Smolka, 1991), $\mathcal{FL}$ can be viewed as a sublanguage of $\mathcal{ALC}$, which gives membership in PSPACE as an upper bound for the complexity of subsumption. We will prove that subsumption in $\mathcal{FL}$ actually is PSPACE-hard, thus complementing the upper bound by a lower bound. Together, they precisely characterize the complexity of this problem as being PSPACE-complete. We now formally introduce the languages $\mathcal{FL}^-$ and $\mathcal{FL}$. In $\mathcal{FL}^-$ only atomic roles are allowed as role expressions, and concepts are formed according to the syntax rules

$$C, D \quad \rightarrow \quad A \mid C \sqcap D \mid \forall R \bullet C \mid \exists R \bullet \top.$$

Note that $\mathcal{FL}^-$ differs from $\mathcal{AL}$ in that it neither contains the concepts $\top$ and $\bot$ nor allows for any form of complement. As a consequence, every $\mathcal{FL}^-$-concept C is satisfiable. This can be seen by considering a completion of $\{x : C\}$, which due to the lack of any kind of complement does not contain a clash. Therefore, deciding the satisfiability of $\mathcal{FL}^-$-concepts is trivial. As said before, it has been shown (Brachman and Levesque, 1984) that subsumption of $\mathcal{FL}^-$-concepts can be decided in polynomial time. This result also follows from a more general result in Section 7 where subsumption in the more expressive language $\mathcal{ALN}$ is proved to be polynomial.

The language $\mathcal{FL}$ extends $\mathcal{FL}^-$ by more expressive roles. For a role R and a concept C the *restriction of R to C*, which is written as $R|_C$, is a role that denotes in an interpretation $\mathcal{I}$ the set

$$(R|_C)^{\mathcal{I}} = \{(a, b) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid (a, b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\}.$$

The syntax rules for $\mathcal{FL}$ are obtained from those for $\mathcal{FL}^-$ by adding the rules for roles:

$$R \quad \rightarrow \quad P \mid R|_C.$$

Every $\mathcal{FL}$-concept can be translated into an equivalent $\mathcal{ALC}$-concept, using the following equivalences:

$$\forall R|_C.D \quad \equiv \quad \forall R.(\neg C \sqcup D) \tag{1}$$

$$\exists R|_C.\top \quad \equiv \quad \exists R.C. \tag{2}$$

In order to enhance the readability of concept expressions we often adopt the notation on the right hand side for expressing role restrictions. It has been shown that, as for $\mathcal{FL}^-$, every $\mathcal{FL}$ – concept is satisfiable (Schmidt-Schauß and Smolka, 1991). Therefore, satisfiability is a trivial problem for this language too.

In the following we reduce the validity problem for quantified boolean formulas to the subsumption problem for $\mathcal{FL}$-concepts. As a first step in this reduction we show that the satisfiability problem for an intersection of $\mathcal{FL}$-concepts and negated $\mathcal{FL}$-concepts can be reduced to the subsumption problem in $\mathcal{FL}$. Since $\mathcal{FL}$ does not allow for complement, concepts of the form $\neg D$ are not in $\mathcal{FL}$.

LEMMA 4.8. *Let $D_1, ..., D_m$ and $D'_1, ..., D'_n$ be $\mathcal{FL}$-concepts. Then one can compute in linear time $\mathcal{FL}$-concepts E and E' such that the following are equivalent*:

- $\neg D_1 \sqcap \cdots \sqcap \neg D_m \sqcap D'_1 \sqcap \cdots \sqcap D'_n$ *is unsatisfiable*
- E *is subsumed by* E'.

Proof. Let A be an atomic concept that occurs neither in the D_i's nor in the D'_j's, and let P be an atomic role. Then we define $E := E_1 \sqcap E_2$, where

$$E_1 \quad := \quad (\forall P|_{D_1}.A) \sqcap \cdots \sqcap (\forall P|_{D_m}.A) \qquad \text{and}$$

$$E_2 \quad := \quad \forall P.(D'_1 \sqcap \cdots \sqcap D'_n).$$

Moreover, we define

$$E' \quad := \quad \forall P.A.$$

Next, we observe that

$$E_1 \quad \equiv \quad \forall P.(\neg D_1 \sqcup A) \sqcap \cdots \sqcap \forall P.(\neg D_m \sqcup A) \tag{3}$$

$$\equiv \quad \forall P.((\neg D_1 \sqcup A) \sqcap \cdots \sqcap (\neg D_m \sqcup A)) \tag{4}$$

$$\equiv \quad \forall P.((\neg D_1 \sqcap \cdots \sqcap \neg D_m) \sqcup A), \tag{5}$$

where (3) holds because of the equivalence (1), (4) holds because for all roles R and all concepts C_1, C_2 we have $\forall R.C_1 \sqcap \forall R.C_2 \equiv \forall R.(C_1 \sqcap C_2)$, and (5) holds because "$\sqcup$" distributes over "$\sqcap$".

Now, E is subsumed by E' if and only if $\neg E' \sqcap E$ is unsatisfiable if and only if $\neg E' \sqcap E_1 \sqcap E_2$ is unsatisfiable, that is, the concept

$$\begin{aligned} \exists P.\neg A \quad &\sqcap \quad \forall P.((\neg D_1 \sqcap \cdots \sqcap \neg D_m) \sqcup A) \\ &\sqcap \quad \forall P.(D'_1 \sqcap \cdots \sqcap D'_n) \end{aligned}$$

is unsatisfiable. Since A occurs neither in the D_i's nor in the D'_j's, the latter is the case if and only if $\neg D_1 \sqcap \cdots \sqcap \neg D_m \sqcap D'_1 \sqcap \cdots \sqcap D'_n$ is unsatisfiable. ▮

Next we show that in a simple $\mathcal{ALC}$-concept one can simulate complement of atomic concepts using negated $\mathcal{FL}$-concepts in such a way that satisfiability is preserved. Let C be a simple $\mathcal{ALC}$-concept and A be an atomic concept. Let B_A, P_A be an atomic concept and an atomic role not occurring in C. We define two $\mathcal{FL}$-concepts by

$$A_+ := \forall P_A.B_A \qquad \text{and} \qquad A_- := \exists P_A.\top.$$

Let A^+ and A^- be the negation normal form of $\neg A_+$ and $\neg A_-$, respectively, i.e., $A^+ := \exists P_A.\neg B_A$, and $A^- := \forall P_A.\bot$. Observe that A^+ and A^- are satisfiable, but $A^+ \sqcap A^- = \exists P_A.\neg B_A \sqcap \forall P_A.\bot$ is unsatisfiable. The concept that is obtained from C by first replacing every occurrence of $\neg A$ with A^- and then every occurrence of A with A^+ is denoted as $C|^A$.

LEMMA 4.9. *Let C be a simple $\mathcal{ALC}$-concept and A be an atomic concept. Then C is satisfiable if and only if $C|^A$ is satisfiable.*

Proof. Suppose that $C|^A$ has been obtained from C using the atomic role P_A and the atomic concept B_A. We first show that for every interpretation $\mathcal{I}$ there exists an interpretation $\mathcal{I}_A$ over the same domain such that $(C|^A)^{\mathcal{I}_A} = C^{\mathcal{I}}$. Let $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ be an interpretation. Then $\mathcal{I}_A = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}_A})$ is obtained from $\mathcal{I}$ by defining $B^{\mathcal{I}_A} := B^{\mathcal{I}}$ for all atomic concepts $B \neq B_A$, $P^{\mathcal{I}_A} := P^{\mathcal{I}}$ for all atomic roles $P \neq P_A$, $(B_A)^{\mathcal{I}_A} := \varnothing$, and $(P_A)^{\mathcal{I}_A} := \{(a, a) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid a \in A^{\mathcal{I}}\}$. Obviously, $(A^+)^{\mathcal{I}_A} = (\exists P_A.\neg B_A)^{\mathcal{I}_A} = A^{\mathcal{I}}$ and $(A^-)^{\mathcal{I}_A} = (\forall P_A.\bot)^{\mathcal{I}_A} = \Delta^{\mathcal{I}} \setminus A^{\mathcal{I}}$. Hence, $(C|^A)^{\mathcal{I}_A} = C^{\mathcal{I}}$. Thus, $C|^A$ is satisfiable if C is satisfiable.

Conversely, suppose $C|^A$ is satisfiable. Then one can derive from $\{x : C|^A\}$ a complete constraint system S that does not contain a clash. We turn S into a constraint system S' by replacing every occurrence of A^+ and A^- with A and $\neg A$, respectively. One easily verifies that S' is again complete and clash free. Since S' contains the constraint $x : C$, which originated from $x : C|^A$, it follows that C is satisfiable. ▮

Let $P.M$ be a quantified boolean formula, where P is a prefix from 1 to m and $M = \{M_1, ..., M_n\}$ is a set of clauses. Let $C_{P.M} = D_1 \sqcap C_1^1 \sqcap \cdots \sqcap C_1^n$ be the translation of $P.M$ into an $\mathcal{ALC}$-concept as defined in Subsection 4.2. Recall that $C_{P.M}$ is built up with an atomic concept A and an atomic role R.

The rest of the proof is structured as follows: First, we describe how one can transform $P.M$ in quadratic time into a concept

$$\bar{C}_{P.M} = \bar{D}^1 \sqcap \cdots \sqcap \bar{D}^m \sqcap \bar{C}^1 \sqcap \cdots \sqcap \bar{C}^n,$$

where each $\bar{D}^i$ and each $\bar{C}^j$ is the negation normal form of an intersection of $\mathcal{FL}$-concepts and negated $\mathcal{FL}$-concepts. Second, we show that $\bar{C}_{P.M}$ and $C_{P.M}|^A$ are equivalent. By virtue of Lemma 4.8 and Lemma 4.9 this will imply that $P.M$ is valid if and only if $\bar{C}_{P.M}$ is satisfiable.

The prefix $(Q_1 1) \cdots (Q_m m)$ is translated into the intersection $\bar{D}^1 \sqcap \cdots \sqcap \bar{D}^m$ as follows. If $Q_i = \forall$, then define the two $\mathcal{FL}$-concepts

$$D^i_+ := \underbrace{\exists R.\exists R.\cdots.\exists R}_{i-1 \text{ times}}.\forall R.A_+,$$

$$D^i_- := \underbrace{\exists R.\exists R.\cdots.\exists R}_{i-1 \text{ times}}.\forall R.A_-,$$

and let $\bar{D}^i$ be the negation normal form of $\neg D^i_+ \sqcap \neg D^i_-$. Note that we have the equivalence

$$\bar{D}^i \equiv \underbrace{\forall R.\forall R.\cdots.\forall R}_{i-1 \text{ times}}.(\exists R.A^+ \sqcap \exists R.A^-).$$

If $Q_i = \exists$, then define the $\mathcal{FL}$-concept

$$\bar{D}^i := \underbrace{\forall R.\forall R.\cdots.\forall R}_{i-1 \text{ times}}.\exists R.\top.$$

Let M_i be the ith clause and let

$$k := \max_{l \in M_i} |l|.$$

We recursively define $\mathcal{FL}$-concepts by

$$E^i_l := \begin{cases} \exists R.(A_+ \sqcap E^i_{l+1}) & \text{if} \quad l \in M_i \\ \exists R.(A_- \sqcap E^i_{l+1}) & \text{if} \quad -l \in M_i \\ \exists R.E^i_{l+1} & \text{if} \quad \text{neither } l \text{ nor } -l \text{ is in } M_i, \end{cases}$$

for $l \in 1..(k-1)$ and by

$$E_k^i := \begin{cases} \exists R.A_+ & \text{if} \quad k \in M_i \\ \exists R.A_- & \text{if} \quad -k \in M_i. \end{cases}$$

Let $\bar{C}^i$ be the negation normal form of $\neg E_1^i$.

LEMMA 4.10. *$\bar{C}_{P_\bullet M}$ and $C_{P_\bullet M}|^A$ are equivalent.*

Proof. It suffices to prove that $\bar{D} := \bar{D}^1 \sqcap \cdots \sqcap \bar{D}^m$ and $D_1|^A$ are equivalent and that for each $i \in 1..m$ the concepts $\bar{C}^i$ and $C_1^i|^A$ are equivalent.

One can check that rewriting $\bar{D}$ with the rule $\forall R.C_1 \sqcap \forall R.C_2 \to \forall R.(C_1 \sqcap C_2)$ yields the concept $D_1|^A$. Since this rule preserves equivalence, the first part of the claim follows.

Considering the definitions of the concepts E_l^i and C_l^i, one notices that the negation normal form of E_l^i is exactly $C_l^i|^A$. Hence, $\bar{C}^i = C_l^i|^A$, which yields the second part of the claim. ∎

THEOREM 4.11. *Subsumption in $\mathcal{FL}$* is PSPACE-*complete.*

4.4. *Unsatisfiability in $\mathcal{ALUR}$*

Now, we show that unsatisfiability in $\mathcal{ALUR}$ is PSPACE-hard by a reduction of the unsatisfiability problem for $\mathcal{ALC}$. The reduction is based on the observation that subroles can be used to simulate full existential quantification. Since we will use this technique in later sections as well, we define it for the entire language $\mathcal{ALCNR}$.

We transform every simple $\mathcal{ALCNR}$-concept C into an $\mathcal{ALUNR}$ concept $\tilde{C}$ such that $\tilde{C}$ is satisfiable if and only if C is satisfiable. Recall that by definition of $\mathcal{ALUNR}$ the concept $\tilde{C}$ must not contain subconcepts of the form $\exists RD$ where $D \neq \top$.

Let C be a simple $\mathcal{ALCNR}$-concept. Suppose that for every subconcept D of C there is a primitive role P_D that does not occur in C. We define a function $\tilde{\cdot}$ that maps every subconcept D of C to a concept $\tilde{D}$. The function $\tilde{\cdot}$ is given by the equations

$$\begin{aligned} \widetilde{\exists R.D} &= \exists (R \sqcap P_D).\top \sqcap \forall (R \sqcap P_D).\tilde{D} \\ \widetilde{\forall R.D} &= \forall R.\tilde{D} \\ \widetilde{D_1 \sqcap D_2} &= \tilde{D}_1 \sqcap \tilde{D}_2 \\ \widetilde{D_1 \sqcup D_2} &= \tilde{D}_1 \sqcup \tilde{D}_2 \end{aligned}$$

and by $\tilde{D} = D$ if D is neither an intersection nor a union nor a concept of the form $\forall R.D'$ or $\exists R.D'$. We say that $\tilde{C}$ is an *$\mathcal{E}$-simulation* of C.

PROPOSITION 4.12. *For every simple $\mathcal{ALCNR}$-concept C one can compute in polynomial time an $\mathcal{E}$-simulation $\tilde{C}$.*

LEMMA 4.13. *Let C be a simple $\mathcal{ALCNR}$-concept and $\tilde{C}$ be an $\mathcal{E}$-simulation of C. Then C is satisfiable if and only if $\tilde{C}$ is satisfiable.*

Proof. Suppose that $\tilde{C}$ is an $\mathcal{E}$-simulation obtained from C using primitive roles P_D.

If $\mathcal{I}$ is an interpretation, then it is easy to see that $\tilde{D}^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for every subconcept D of C. Therefore, C is satisfiable if $\tilde{C}$ is satisfiable.

Conversely, we show that for every interpretation $\mathcal{I}$ there exists an interpretation $\tilde{\mathcal{I}}$ over the same domain such that $\tilde{C}^{\tilde{\mathcal{I}}} = C^{\mathcal{I}}$. Let $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ be an interpretation. Then $\tilde{\mathcal{I}} = (\Delta^{\mathcal{I}}, \cdot^{\tilde{\mathcal{I}}})$ is obtained from $\mathcal{I}$ by defining $A^{\tilde{\mathcal{I}}} = A$ if A is a primitive concept, $P^{\tilde{\mathcal{I}}} = \{(a, b) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid b \in D^{\mathcal{I}}\}$ if $P = P_D$ for some subconcept D of C, and $P^{\tilde{\mathcal{I}}} = P^{\mathcal{I}}$ for the remaining primitive roles. Now it is straightforward to verify that $\tilde{D}^{\tilde{\mathcal{I}}} = D^{\mathcal{I}}$ for every subconcept D of C. Thus $\tilde{C}$ is satisfiable if C is satisfiable. ∎

COROLLARY 4.14. *Satisfiability of $\mathcal{ALUR}$-concepts is* PSPACE*-hard.*

Proof. Satisfiability of $\mathcal{ALC}$-concepts can be reduced to satisfiability of $\mathcal{ALUR}$-concepts, since for every simple $\mathcal{ALC}$-concept C the $\mathcal{E}$-simulation $\tilde{C}$ is in $\mathcal{ALUR}$. ∎

4.5. *Unsatisfiability in $\mathcal{ALNR}$*

We show that unsatisfiability in $\mathcal{ALNR}$ is PSPACE-hard. This sharpens a result by Nebel (Nebel, 1988), who showed that subsumption in $\mathcal{ALNR}$ is co-NP-hard. The proof is by a reduction of the validity problem for quantified boolean formulas (QBF). We first reduce QBF to satisfiability in $\mathcal{ALENR}$. Then using $\mathcal{E}$-simulations we conclude the claim.

Let $P.M$ be a quantified boolean formula where $P = (Q_1 1) \cdots (Q_m m)$ is a prefix from 1 to m and $M = \{M_1, ..., M_n\}$. We translate $P.M$ into an $\mathcal{ALENR}$-concept

$$C_{P.M} = D_1 \sqcap C_1^1 \sqcap \cdots \sqcap C_1^n.$$

Let A, B be atomic concepts, let $P_0, P_1, \ldots P_n$ be atomic roles, and let $R := P_0 \sqcap P_1 \sqcap \cdots \sqcap P_n$.

The concept D_1 is obtained from the prefix P using the equations

$$D_l := \begin{cases} (\exists R.A) \sqcap (\exists R.\neg A) \sqcap (\forall R.D_{l+1}) & \text{if } Q_l = \forall \\ (\exists R.\top) \sqcap (\forall R.D_{l+1}) & \text{if } Q_l = \exists \end{cases}$$

for $l \in 1..(m-1)$ and the equation

$$D_m := \begin{cases} (\exists R.A) \sqcap (\exists R.\neg A) & \text{if } Q_m = \forall \\ \exists R.\top & \text{if } Q_m = \exists. \end{cases}$$

The concept C_1^i is obtained from the clause M_i as follows. Let

$$k := \max_{l \in M_i} |l|.$$

Then, for $l \in 1..(k-1)$, define

$$C_l^i := \begin{cases} (\leqslant 2P_i) \sqcap \exists P_i \bullet (B \sqcap A) \sqcap \exists P_i \bullet (\neg B \sqcap C_{l+1}^i) & \text{if} \quad l \in M_i \\ (\leqslant 2P_i) \sqcap \exists P_i \bullet (B \sqcap \neg A) \sqcap \exists P_i \bullet (\neg B \sqcap C_{l+1}^i) & \text{if} \quad -l \in M_i \\ \forall P_i \bullet C_{l+1}^i & \text{otherwise} \end{cases}$$

and define

$$C_k^i := \begin{cases} \forall P_i \bullet A & \text{if} \quad k \in M_i \\ \forall P_i \bullet \neg A & \text{if} \quad -k \in M_i. \end{cases}$$

The basic idea in the definition of $C_{P \bullet M}$ is similar to the one underlying the reduction of QBF to satisfiability in $\mathscr{ALC}$. Comparing the two reductions, one realizes that in fact the concepts D_l are defined in the same way. The definitions of the concepts C_l^i differ, because $\mathscr{ALENR}$ does not provide an explicit disjunction through union. However, there is a hidden disjunction in the "at most" restrictions, which is reflected in the completion calculus by the nondeterministic behavior of the $\rightarrow_{\leqslant}$-rule, which identifies different successors of a variable. Intuitively, this nondeterminism only is a source of complexity if it matters which variables are identified. In other words, the hidden disjunction is only activated if different successors have different constraints. For this reason, the "at most" restriction in the definition of C_l^i is combined with existential quantifications.

As in the reduction for $\mathscr{ALC}$ the concept $C_{P \bullet M}$ is defined in such a way that canonical sets of assignments for P satisfying the clauses of M can be translated into clash-free complete constraint systems obtained from $\{x : C_{P \bullet M}\}$ and vice versa. We will explain this correspondence at an intuitive level before providing the proofs.

Let S be a constraint system and R a be role. A chain $\gamma = y_0 \cdots y_n$ is S is called an *R-chain* if y_i is an R-successor of y_{i-1} in S for $i \in 1..n$. Recall that y_0 is called the starting point of the chain, and n is the length of the chain.

Suppose that S is a clash-free complete constraint system obtained from $\{x : C_{P \bullet M}\}$. Consider any variable y such that the constraint $y : D_l$ occurs in S for some $l \in 1..m$. The variable x, for instance, is one such variable, since S contains $x : D_1$. The concept D_l encodes the lth quantifier Q_l in the prefix P. It is defined in such a way that, if $Q_l = \forall$, the completion rules create R-successors y', y'' of y with the constraints $y' : A$, $y'' : \neg A$ or, if $Q_l = \exists$, a single R-successor y' without specifying whether it belongs to A or $\neg A$. In addition, S contains $y' : D_{l+1}$ and $y'' : D_{l+1}$ if $l < m$. This shows that the set of R-chains of length m with starting point x has a structure similar to a set of assignments that is canonical for P.

The concept C_1^i encodes the ith clause M_i. We now point out how the implicit disjunction present in the "at most" restriction is exploited to mimic the disjunction in clauses. Let $\gamma = y_0 \cdots y_m$ be an R-chain of length m with starting point $y_0 = x$. Let y_{l-1} be a variable occurring in γ such that S contains the constraint $y_{l-1} : C_l^i$ and M_i contains l or $-l$. By definition of the C_l^i's this is the case if, for instance, l is the least number such that M_i contains l or $-l$. Without loss of generality we may assume that $l \in M_i$. Now, the completion rules create two P_i-successors z', z'' of y_{l-1} together with the constraints

$$y : (\leqslant 2P_i), \qquad z' : B, \qquad z' : A, \qquad z'' : \neg B, \qquad z'' : C_{l+1}^i.$$

No subsequent application of rules may identify z' and z'', because otherwise S would contain a clash. Since y_{l-1} occurs in γ, it also has an R-successor y_l. Since R is a subrole of P_i and S contains $y_{l-1} : (\leqslant 2P_i)$, either y_l and z' or y_l and z'' have been identified. The choice between z' and z'' corresponds to the logical disjunction in the clause M_i. If y_l and z' are identified, then y_l is constrained to A. This means that the assignment corresponding to γ satisfies the literal l. However, when identifying y_l and z'' we end up with the constraint $y_l : C^i_{l+1}$, which corresponds to postponing the choice of the literal to be satisfied.

Along the lines of the above discussion we will now prove that a quantified boolean formula $P \bullet M$ is valid if and only if the $\mathscr{ALENR}$-concept $C_{P \bullet M}$ is satisfiable. In the proof we will use the following definitions that make the relation between chains and assignments explicit.

Let A be the atomic concept and R be the role used in the definition of $C_{P \bullet M}$. Let k, l be positive integers with $k \leqslant l$ and S be a constraint system derived from $\{x : C_{P \bullet M}\}$. If γ is an R-chain of length $l-k+1$ in S, say $\gamma = y_{k-1} .. y_l$, then we associate to γ the assignment

$$\alpha^{k,l}_{\gamma} : \{k, k+1, ..., l\} \to \{\textbf{true}, \textbf{false}\}$$

that is defined by

$$\alpha^{k,l}_{\gamma}(i) := \begin{cases} \textbf{true} & \text{if} \quad y_i : A \text{ is in } S \\ \textbf{false} & \text{otherwise.} \end{cases}$$

For every variable y and all positive integers k, l we denote as $[S]^{k,l}_y$ the set of all $\alpha^{k,l}_{\gamma}$ obtained from R-chains γ of length $l-k+1$ in S that have starting point y.

LEMMA 4.15. *If $C_{P \bullet M}$ is satisfiable, then $P \bullet M$ is valid.*

Proof. Suppose that $C_{P \bullet M}$ is a satisfiable concept. Let x be a variable. Since $C_{P \bullet M}$ is satisfiable, the constraint system $\{x : C_{P \bullet M}\}$ has a clash-free completion S. We will show that

1. the set of assignments $[S]^{1,m}_x$ is canonical for P
2. every assignment in $[S]^{1,m}_x$ satisfies every clause of $P \bullet M$.

1. To show that $[S]^{1,m}_x$ is canonical for P, we prove that for all $k \in 1, ..., m$ the following claim holds:

> if y is a variable such that $y : D_k$ is in S, then $[S]^{k,m}_y$ is canonical for $(Q_k k) \cdots (Q_m m)$.

We prove the claim by induction on $m-k$.

Suppose that $k = m$. We distinguish the case $Q_m = \forall$ and the case $Q_m = \exists$. If $Q_m = \forall$, then by definition of D_m we have

$$D_{m-1} = (\exists R \bullet A) \sqcap (\exists R \bullet \neg A).$$

Since S is complete, the variable y has R-successors y', y'' such that the constraints $y' : A$ and $y'' : \neg A$ are in S. Hence, $[S]_y^{m,m}$ contains exactly two assignments α' and α'', which correspond to the two R-chains yy' and yy''. They satisfy $\alpha'(m) = \textbf{true}$ and $\alpha''(m) = \textbf{false}$. This implies that $[S]_y^{m,m}$ is canonical for $(Q_m m)$.

If $Q_m = \exists$, a similar argument applies.

Suppose the claim holds for some positive integer k. We show that it also holds for $k-1$. Let y be a variable such that the constraint $y : D_{k-1}$ is in S. As before, we distinguish between the case that $Q_{k-1} = \forall$ and the case that $Q_{k-1} = \exists$. If $Q_{k-1} = \forall$, then by definition of D_{k-1} we have

$$D_{k-1} = (\exists R.A) \sqcap (\exists R.\neg A) \sqcap (\forall R.D_k).$$

Since S is complete, y has R-successors y', y'' with constraints $y' : A$, $y' : D_k$ and $y'' : \neg A$, $y'' : D_k$ are in S. By the induction hypothesis, the sets of assignments $[S]_{y'}^{k,m}$ and $[S]_{y''}^{k,m}$ are canonical for $(Q_k k) \cdots (Q_m m)$. By definition of the set $[S]_y^{k-1,m}$ it follows that $[S]_y^{k-1,m}$ is canonical for $(Q_{k-1} k-1) \cdots (Q_m m)$.

If $Q_{k-1} = \exists$, a similar argument applies.

2. Let M_i be a clause in M such that $k = \max_{l \in M_i} |l|$. Furthermore, let α be an assignment in $[S]_x^{1,m}$. Then there exists in S a R-chain $\gamma = y_0 y_1 \cdots y_m$ with starting point $y_0 = x$ such that $\alpha = \alpha_\gamma^{1,m}$. We will prove that $\alpha_\gamma^{1,m}$ satisfies M_i.

To do so, observe that either $y_{k-1} : C_k^i$ is in S, or there is an $l \in 1..(k-1)$ such that $y_{l-1} : C_l^i$ is in S, but $y_l : C_{l+1}^i$ is not in S.

Suppose that $y_{k-1} : C_k^i$ is in S. Then by definition of C_k^i we have

$$C_k^i = \forall P_i.A',$$

where $A' = A$ if $k \in M_i$, and $A' = \neg A$ if $-k \in M_i$. Since R is a subrole of P_i and S is complete, the constraint $y_k : A'$ is in S. Hence, $\alpha_\gamma^{1,m}(k) = \textbf{true}$ if $k \in M_i$, and $\alpha_\gamma^{1,m}(k) = \textbf{false}$ if $-k \in M_i$. Thus $\alpha_\gamma^{1,m}$ satisfies M_i.

If there is an l such that $y_{l-1} : C_l^i$ is in S, but $y_l : C_{l+1}^i$ is not in S, then either l or $-l$ is in M_i. Otherwise, by definition of C_l^i we would have $C_l^i = \forall P_i.C_{l+1}^i$. Since R is a subrole of P_i and S is complete, this would imply that $y_l : C_{l+1}^i$ is in S, which yields a contradiction.

Suppose that $l \in M_i$. Then by definition of C_l^i we have

$$C_l^i = (\leqslant 2 P_i) \sqcap \exists P_i.(B \sqcap A) \sqcap \exists P_i.(\neg B \sqcap C_{l+1}^i).$$

Since S is complete, y_l has P_i-successors z', z'' which are constrained by $z' : B$, $z' : A$ and $z'' : \neg B$, $z'' : C_{l+1}^i$. Furthermore, z' and z'' are distinct because otherwise there would be a variable in S that is constrained by B and $\neg B$, which contradicts the fact that S is clash free.

Now, y_l is constrained to have at most two fillers of P_i, but the completion rules have generated two distinct P_i-successors z' and z'' and the R-successor y_{l+1}. Since R is a subrole of P_i, the variable y_{l+1} must have been identified either with z' or

with z''. If it has been identified with z'', then S contains the constraint $y_l : C^i_{l+1}$, which contradicts our assumption on l. Hence, it has been identified with z', which implies that S contains the constraint $y_l : A$. By definition of $\alpha^{1,m}_\gamma$, we have $\alpha^{1,m}_\gamma(l) = \textbf{true}$. Thus, $\alpha^{1,m}_\gamma$ satisfies M_i.

If $-l \in M_i$, a similar argument applies. ∎

LEMMA 4.16. *If $P_\bullet M$ is valid, then $C_{P_\bullet M}$ is satisfiable.*

Proof. If $P_\bullet M$ is valid, then there exists a set of assignments $\mathbf{A}$ such that $\mathbf{A}$ is canonical for P and every $\alpha \in \mathbf{A}$ satisfies every clause $M_i \in M$.

Let x be a variable. In order to show that $C_{P_\bullet M}$ is satisfiable it suffices to construct from $\mathbf{A}$ a complete and clash-free constraint system $\bar{S}$ that contains the constraint system $\{x : D_1, x : C^1_1, ..., x : C^n_1\}$. This is done in two steps:

1. we construct a complete and clash-free constraint system S^0 containing $\{x : D_1\}$ such that $[S^0]^{1,m}_x = \mathbf{A}$

2. for every $i \in 1..n$ we construct a complete and clash-free constraint system S^i that contains $S^0 \cup \{x : C^i_1\}$.

Since the roles occurring in C^i_1 and C^j_1 are different, we can assume that for $i \neq j$, every variable occurring in $S^i \cap S^j$ occurs also in S^0. Then $\bar{S} = \bigcup^n_{i=1} S^i$ is a constraint system with the desired properties.

We show Step 1. Let S' be a completion of $\{x : D_1\}$. Because of the structure of D_1, the system S' is clash-free.

Due to our encoding of existential quantifiers in P, not all variables in S' that have been introduced by the completion rules are constrained to A or to $\neg A$. We will add constraints to S' such that every variable, except x, is constrained either to A or $\neg A$. In this process, we exploit the structure of $\mathbf{A}$.

Observe that for every R-chain $\gamma = xy_1 \cdots y_m$ of length m in S' there exists exactly one assignment $\alpha \in \mathbf{A}$ such that $\alpha(l) = \textbf{true}$ if $y_l : A$ is in S' and $\alpha(l) = \textbf{false}$ if $y_l : \neg A$ is in S'. We denote this assignment as α_γ. Now we transform S' into S^0 as follows. For every variable $y \neq x$ occurring in S' that is neither constrained to A nor to $\neg A$ we choose an R-chain $\gamma = xy_1 \cdots y_m$ with $y = y_l$ for some l, and add to S' the constraint $y : A$ if $\alpha_\gamma(l) = \textbf{true}$ or $y : \neg A$ if $\alpha_\gamma(l) = \textbf{false}$, respectively. Note that this construction does not depend on which chain γ we choose for y.

By construction, S^0 is clash-free and $[S^0]^{1,m}_x = \mathbf{A}$.

We now show Step 2. We derive S^i from $S^0 \cup \{x : C^i_1\}$ using a modified set of completion rules that we obtain by replacing the $\rightarrow_{\leqslant}$-rule by the following $\rightarrow_{\leqslant 2}$-rule:

$$S \rightarrow_{\leqslant 2} S[y/z]$$

if no other completion rule applies to S,
there is a variable v such that $v : (\leqslant 2P_i)$ is in S,
y is an R-successor of v, z is an P_i-successor of v,
$y \neq z$, and neither $y : A$, $z : \neg A$ nor $y : \neg A$, $z : A$ are in S.

This rule applies, if y is an R-successor of v in S^0, and if for some $l \in 1..(k-1)$ such that $l \in M_i$ or $-l \in M_i$ the constraint $v : C_l^i$ is in S. In this case

$$C_l^i = (\leqslant 2P_i) \sqcap \exists P_i.(B \sqcap A') \sqcap \exists P_i.(\neg B \sqcap C_{l+1}^i),$$

where $A' = A$ if $l \in M_i$ and $A' = \neg A$ if $-l \in M_i$. The completion rules will introduce P_i-successors z', z'' of v together with the constraints $z' : B$, $z' : A'$, and $z'' : \neg B$, $z'' : C_{l+1}^i$. Now, the $\rightarrow_{\leqslant 2}$-rule applies and identifies the variables y and z' or y and z'' in such a way that no clash arises.

Let S^i be a completion of $S^0 \cup \{x : C_1^i\}$ obtained with the modified set of completion rules. By construction, S^i is clash-free. Since the only "at most" restriction occurring in S^i have the form $(\leqslant 2P_i)$, the system S^i is also complete with respect to the original set of rules. ∎

We now come to the main result of this subsection.

LEMMA 4.17. *Satisfiability in* $\mathcal{ALENR}$ *is* PSPACE-*hard.*

Proof. Computing for a quantified boolean formula its translation into an $\mathcal{ALENR}$-concept takes polynomial time. By Lemmas 4.15 and 4.16 the formula is valid if and only if the concept is satisfiable. Then there exists a polynomial-time reduction of the validity problem for quantified boolean formulas, which is PSPACE-hard, to the satisfiability problem in $\mathcal{ALENR}$. ∎

THEOREM 4.18. *Satisfiability in* $\mathcal{ALNR}$ *is* PSPACE-*hard.*

Proof. For every $\mathcal{ALENR}$-concept C the $\mathcal{E}$-simulation $\tilde{C}$ is in $\mathcal{ALNR}$. Since $\mathcal{E}$-simulations can be computed in polynomial time, Lemma 4.17 yields the claim. ∎

4.6. *Summary on* PSPACE-*Complete Languages*

Since there exists a polynomial space algorithm for subsumption in $\mathcal{ALCNR}$, which is the top element of the lattice of $\mathcal{AL}$-languages, we know that subsumption and satisfiability in all $\mathcal{AL}$-languages are in PSPACE. Moreover, satisfiability is PSPACE-hard in $\mathcal{ALC}$, $\mathcal{ALUR}$, and $\mathcal{ALNR}$. It follows that for all extensions of these three languages subsumption and satisfiability are PSPACE-complete.

THEOREM 4.19 (PSPACE-Completeness). *Subsumption and* (*un*)*satisfiability are* PSPACE-*complete problems for* $\mathcal{ALC}$, $\mathcal{ALUR}$, $\mathcal{ALNR}$, $\mathcal{ALCR}$, $\mathcal{ALCN}$, $\mathcal{ALENR}$, $\mathcal{ALUNR}$, *and* $\mathcal{ALCNR}$.

5. NP-COMPLETE LANGUAGES

In this section we consider languages that do not contain constructs expressing logical disjunction—whether explicitly, such as unions, or implicitly, such as "at most" restrictions. In the lattice of $\mathcal{AL}$-languages, $\mathcal{ALER}$ is the greatest element with this property.

We will show that reasoning in $\mathcal{ALER}$ is easier than in the languages considered in the previous section. We show that whenever an $\mathcal{ALER}$-concept is subsumed by another one, there exists in the completion calculus a proof for this fact which is of polynomial size. Since such a proof can be guessed in nondeterministic polynomial time, it follows that the problem of deciding subsumption between $\mathcal{ALER}$-concepts is in NP.

In (Donini *et al.*, 1992) unsatisfiability and subsumption in $\mathcal{ALE}$ were shown to be NP-complete; hence they are NP-complete for $\mathcal{ALER}$ too. We also show that $\mathcal{ALR}$ unsatisfiability is NP-hard, thus identifying a third NP-complete language.

5.1. *An* NP-*Algorithm for* $\mathcal{ALER}$

An $\mathcal{ALER}$-concept C is subsumed by the $\mathcal{ALER}$-concept D if $C \sqcap \neg D$ is unsatisfiable. In general, $C \sqcap \neg D$ is not in $\mathcal{ALER}$, since $\mathcal{ALER}$ does not allow for negation of arbitrary concepts. However, it is an element of $\mathcal{ALCR}$. If we use the completion calculus for checking the unsatisfiability of arbitrary $\mathcal{ALCR}$-concepts it may generate an exponential number of constraint systems each of which is of exponential size. In order to be sure that the concept is unsatisfiable we have to find a clash in each system. In the following we will change the completion calculus, which attempts to build up a model for a given concept, into a calculus that is tailored to finding clashes.

As a first step we introduce a modified completion calculus in which the applicability of the $\rightarrow_{\exists}$-rule is reduced. The *trace rules* (see also (Schmidt-Schauß and Smolka, 1991)) consist of the $\rightarrow_{\sqcap}$-, the $\rightarrow_{\sqcup}$-, and the $\rightarrow_{\forall}$-rule given in Subsection 3.2, together with the rule

$$S \quad \rightarrow_{T\exists} \quad \{xP_1\, y, \ldots, xP_k\, y, y : C\} \cup S$$

if there is no successor of x in S,
$x : \exists R.C$ is in S, $R = P_1 \sqcap \cdots \sqcap P_k$,
and y is a new variable.

The difference between the $\rightarrow_{\exists}$-rule and the $\rightarrow_{T\exists}$-rule is that the latter is applied only once for a variable x. We are thus compelled to make a nondeterministic choice among the constraints of the form $x : \exists R.C$.

Let C be a simple $\mathcal{ALCR}$-concept. A constraint system T is a *trace* of $\{x : C\}$ if T is obtained from $\{x : C\}$ by application of the trace rules. Obviously, the variables occurring in a trace T form a chain with starting point x. For this reason the size of a trace is polynomial in the size of C. If S is a completion of $\{x : C\}$ then for any variable y occurring in S there exists a trace T such that T contains all the constraints in S that depend on y. Since clashes involve only constraints that depend on a single variable, S contains a clash if and only if there is a trace $T \subseteq S$ that contains a clash. These observations are summarized in the following proposition.

PROPOSITION 5.1. *Let C be a simple $\mathcal{ALCR}$-concept. Then*:

1. *the size of a trace derived from $\{x : C\}$ is polynomial in the size of C;*
2. *C is unsatisfiable if and only if for every completion S of $\{x : C\}$ there exists a trace $T \subseteq S$ such that T contains a clash.*

In order to prove that C is unsatisfiable it is sufficient to generate traces instead of completions. However, it is necessary to generate enough traces to cover all completions. For this purpose we introduce the S-rule calculus, which operates on *sets* of traces and behaves nondeterministically only for existential quantification. It is interesting to observe that the S-rules provide an alternative method for deciding the satisfiability of $\mathcal{ALCR}$-concepts.

The S-*rules* are ($\mathcal{T}$ denotes a set of traces):

1. $\{T\} \cup \mathcal{T} \rightarrow^{S}_{*} \{T'\} \cup \mathcal{T}$
   if $T \notin \mathcal{T}$ and $T \rightarrow_{*} T'$ where $* \in \{\sqcap, T\exists, \forall\}$
2. $\{T\} \cup \mathcal{T} \rightarrow^{S}_{\sqcup} \{T', T''\} \cup \mathcal{T}$
   if $T \notin \mathcal{T}$, $x : C \sqcup C'$ is in T, neither $x : C$ nor $x : C'$ is in T and
   $T' = \{x : C\} \cup T$, $T'' = \{x : C'\} \cup T$.

The S-rules eliminate the nondeterminism introduced by unions, since the two traces that can be obtained by an application of the $\rightarrow_{\sqcup}$-rule are put both into the new set of traces. The nondeterminism in choosing a constraint to which the $\rightarrow_{T\exists}$-rule applies persists.

The following lemmata can be proved with arguments similar to those in (Donini *et al.*, 1992).

LEMMA 5.2. *Let C be a simple $\mathcal{ALCR}$-concept. Then*:

1. *every* S-*rule derivation starting with $\{\{x : C\}\}$ terminates;*
2. *C is unsatisfiable if and only if $\{\{x : C\}\}$ can be transformed by the* S-*rules into a set $\mathcal{T}$ such that each trace in $\mathcal{T}$ contains a clash.*

Part 2 of Lemma 5.2 yields the soundness and completeness of the S-calculus. Together with Part 1 it suggests a nondeterministic method for deciding the unsatisfiability of an $\mathcal{ALCR}$-concept C. The method generates a number of traces that is—in the general case—exponential in the size of C. This is not surprising, since unsatisfiability in $\mathcal{ALCR}$ is PSPACE-complete. However, when checking subsumption between $\mathcal{ALER}$-concepts D and C, which is equivalent to checking the unsatisfiability of $C \sqcap \neg D$, a better result can be achieved. Intuitively, the reason is that C contains no unions, the negation normal form of D, say D', contains no intersections, and the intersections in C and the unions in D' do not interact.

LEMMA 5.3. *Let C, D be $\mathcal{ALER}$-concepts and let D' be the negation normal form of D. Then any* S-*rule derivation starting with $\{\{x : C \sqcap D'\}\}$ leads to a set of traces whose cardinality is bounded by the size of D'.*

From this results we can conclude that subsumption between $\mathcal{ALER}$-concepts is in NP.

THEOREM 5.4. *Subsumption in $\mathcal{ALER}$ can be decided in nondeterministic polynomial time.*

Proof. Let C, D be $\mathcal{ALER}$-concepts and let D' be the negation normal form of D. By Lemma 5.1, traces of $\{x : C \sqcap D'\}$ are of polynomial size, and by Lemma 5.3, each set derived from $\{\{x : C \sqcap D'\}\}$ contains polynomially many traces. By Part 2 of Lemma 5.2, C is subsumed by D if and only if $\{\{x : C \sqcap D'\}\}$ can be transformed with the S-rules into a set of traces each of which contains a clash. By the above arguments, computing such a derivation takes time polynomial in the size of C and D. ∎

5.2. NP-*Hardness Results*

From (Donini *et al.*, 1992) we know that unsatisfiability in $\mathcal{ALE}$ is NP-hard. We use this result to conclude that unsatisfiability in $\mathcal{ALR}$ is NP-hard, too.

THEOREM 5.5. *Unsatisfiability in $\mathcal{ALR}$ is* NP*-hard.*

Proof. The claim follows by Lemma 4.13, since for every $\mathcal{ALE}$-concept C the $\mathcal{E}$-simulation $\tilde{C}$ is in $\mathcal{ALR}$. ∎

Moreover, since $\mathcal{ALE}$ is a sublanguage of $\mathcal{ALEN}$, the NP-hardness result for $\mathcal{ALE}$ yields a lower bound for the complexity of reasoning in $\mathcal{ALEN}$.

PROPOSITION 5.6. *Unsatisfiability in $\mathcal{ALEN}$ is* NP*-hard.*

5.3. *Summary on* NP*-Complete Languages*

We combine the results of the preceding subsections in the following theorem.

THEOREM 5.7 (NP-Completeness). *Subsumption and unsatisfiability are* NP*-complete problems for $\mathcal{ALE}$, $\mathcal{ALR}$, and $\mathcal{ALER}$.*

As discussed in Section 3.4, results on NP-completeness have a character different from former intractability results for concept languages. The work in (Brachman and Levesque, 1984; Nebel, 1988; Schmidt-Schauß and Smolka, 1991) identified the disjunctive constructs of role restriction, "at most" restriction, and union that together with concept intersection give rise to intractability. The NP-hardness of $\mathcal{ALE}$ and $\mathcal{ALR}$, however, shows that the interplay of universal and existential quantifiers is another unavoidable source of complexity. The different nature of the two sources of complexity is illustrated by the fact that the former makes subsumption co-NP-hard and the latter makes it NP-hard.

6. Co-NP-COMPLETE LANGUAGES

In $\mathcal{ALU}$, intersection, union, and complement of primitive concepts are available. Therefore, deciding the unsatisfiability of $\mathcal{ALU}$-concepts is at least as hard as deciding the unsatisfiability of formulas in propositional logic, which is known to be a co-NP-complete problem. Conversely, it has been shown that

universal quantification and restricted existential quantification over roles do not increase the complexity of the problem, i.e., unsatisfiability in $\mathcal{ALU}$ is co-NP-complete (Schmidt-Schauß and Smolka, 1991).

One might conjecture that subsumption in $\mathcal{ALU}$ is harder, since it is equivalent to unsatisfiability of certain concepts containing both unions and full existential quantification. These two constructs show up in subsumption problems, since C is subsumed by D if and only if $C \sqcap \neg D$ is unsatisfiable if and only if $C \sqcap D'$ is unsatisfiable, where D' is the negation normal form of $\neg D$. If D is an $\mathcal{ALU}$-concept containing intersection and universal role quantification, then D' contains union and full existential role quantification. In the sequel we will show that subsumption in an even larger language, namely $\mathcal{ALUN}$, has the same complexity as unsatisfiability in $\mathcal{ALU}$. This result still holds if we drop the assumption that numbers are coded in unary. Therefore, in the present and the following section we will assume that numbers occurring in concepts are written in binary notation (or, more generally, q-ary notation for some $q \geqslant 2$). Note that, as a consequence, a concept of the form $(\geqslant nP)$ has length $O(\log n)$, but every model of it has cardinality at least n, i.e., the cardinality is exponential in the length of the concept.

6.1. *An Optimized Calculus for* $\mathcal{ALCN}$

As discussed before, since the language $\mathcal{ALUN}$ is not closed under complements, subsumption checking in $\mathcal{ALUN}$ amounts to checking the unsatisfiability of certain $\mathcal{ALCN}$-concepts. Intuitively, the reason why the satisfiability of such concepts can be decided in nondeterministic polynomial time is that, even if the models of a concept may have a number of elements that is exponential in the length of the concept, in models generated by the completion calculus the elements can be grouped in polynomially many sets in such a way that elements belonging to one set share the same constraints. In order to make this more precise we will modify the general calculus given in Subsection 3.2 in such a way that "at least"-constraints only lead to the introduction of a single new variable. The modified calculus appeared first in (Hollunder *et al.*, 1990) and its soundness and completeness are proved in (Nutt, 1993). We present it here because it is the basis for our complexity studies.

The *quasi-completion rules* consist of the $\rightarrow_{\sqcap}$-, $\rightarrow_{\sqcup}$-, $\rightarrow_{\exists}$-, $\rightarrow_{\forall}$-, and $\rightarrow_{\leqslant}$-rules together with the following revised $\rightarrow_{\geqslant}$-rule:

$$S \quad \rightarrow_{\geqslant 1} \quad \{xPy\} \cup S$$

if no other completion rule applies to S,
$x : (\geqslant nP)$ is in S, x does not have a P-successor in S,
and y is a new variable.

Intuitively, this weak version of the $\rightarrow_{\geqslant}$-rule is sufficient because in the absence of role intersection, for any variable and primitive role P the completion rules will impose the same constraints on all P-successors generated during a $\rightarrow_{\geqslant}$-step. Note that, because of the applicability condition of the rule, the P-successor of x

generated by the $\rightarrow_{\geqslant 1}$-rule is the only P-successor of x and will remain the only one in all constraint systems derived from $S \cup \{xPy\}$.

Obviously, for the quasi-completion calculus an invariance theorem analogous to Theorem 3.4 holds. A constraint system is *quasi-complete* if no quasi-completion rule is applicable.

The original calculus detects contradictory number restrictions of the form $\{x : (\geqslant mR),\ x : (\leqslant nR)\}$ with $m > n$ because the $\rightarrow_{\geqslant}$-rule generates m pairwise separated R-successors of x that together with the constraint $x : (\leqslant nR)$ form a clash. In order to find contradictions of this kind in quasi-complete constraint systems we have to redefine the notion of a clash. A *simplified clash* is a constraint system having one of the following forms:

- $\{x : \bot\}$;
- $\{x : A, x : \neg A\}$;
- $\{x : (\geqslant mP), x : (\leqslant nP)\}$ where $m > n$.

Note that simplified clashes consist of contradictory membership constraints, but do not involve relationship constraints or disequations. It is easy to see that, in order to detect contradictions in constraint systems containing role intersection, it does not suffice to look only for simplified clashes. However, as the next lemma shows, the quasi-completion rules provide a sound and complete method for deciding the satisfiability of $\mathcal{ALCN}$-concepts. A proof of the lemma can be found in (Nutt, 1993).

LEMMA 6.1. *Let C be a simple $\mathcal{ALCN}$-concept and S be a quasi-complete constraint system derived from $\{x : C\}$ by the quasi-completion rules. Then S is satisfiable if and only if it contains no simplified clash.*

6.2. *Polynomial Length of Derivations*

In this subsection we will use the quasi-completion calculus to show that nonsubsumption of $\mathcal{ALUN}$-concepts can be decided in nondeterministic polynomial time. Let C, D be $\mathcal{ALUN}$-concepts and let D' be the negation normal form of D. By Lemma 6.1, C is not subsumed by D if and only if the constraint system $\{x : C \sqcap D'\}$ can be transformed by the quasi-completion rules into a quasi-complete system that does not contain a simplified clash. Our goal is to show that any derivation with the quasi-completion rules issuing from $\{x : C \sqcap D'\}$ has length $O(n^3)$ where $n = \sharp(C \sqcap D')$.

The fact that such derivations are of polynomial length may be surprising, because both universal and full existential quantification appear in $C \sqcap D'$ and we have identified the interplay of universal and existential quantifiers as an unavoidable source of complexity in Section 5. However, the interaction of quantifiers is rather limited in the present case, since all existentially quantified subconcepts of C have the form $\exists P.\top$ and all universally quantified subconcepts of D' have the form $\forall P.\bot$.

We assume in the following that $\mathcal{ALCN}$-concepts do not contain subconcepts of the form $\exists P.\top$. We can do so without loss of generality, since each such subconcept

can be replaced with $(\geqslant 1P)$ while preserving equivalence. Transforming a concept by making these replacements takes time linear in the length of the concept.

In this subsection it is crucial that to consider subconcepts of a concept that occur in different positions as distinct entities, even if they are syntactically identical. The reason is that we want to give upper bounds for the size of constraint systems by the following kind of argument (see Lemma 6.3):

> Let D be an $\mathcal{ALCN}$-concept in negation normal form that does not contain universal quantification and S be derived from $\{x : C\}$. Then for each subconcept E of D there is at most one constraint of the form $y : E$ in S.

Obviously, this argument does not hold for the concept $D = A \sqcap \exists P.A$, because we can derive from $x : D$ the constraint system

$$\{x : D, x : A, x : \exists P.A, xPy, y : A\},$$

which contains the two constraints $x : A$ and $y : A$ for the subconcept A. The presence of two constraints is due to the twofold occurrence of A in D.

In order to be able to distinguish two different occurrences of a subconcept as two distinct entities, we slightly modify the quasi-completion calculus. After sketching the behavior of such a modified calculus we will draw conclusions about the original one. The technical details of the modification are immaterial for our arguments. Therefore we describe rather informally what the constraints and rules of such a calculus look like.

Obviously, every $\mathcal{ALCN}$-concept D can be viewed as a (syntactic) tree. Let us assume that we have arbitrarily many colors. Then we paint the root of every subtree of D with a different color, except that we do not paint role symbols.

We call such a concept a *colored version* of D. For example, if $D = A \sqcap \exists P.A$ as above, a possible colored version is

$$\tilde{D} = A^{\text{yellow}} \sqcap^{\text{green}} \exists^{\text{red}} P.A^{\text{blue}},$$

where the superscript indicates the color of the corresponding symbol. The semantics of concepts remains unaffected by coloring.

The calculus will operate on constraint systems that contain constraints with colored concepts. In such a system, constraints $y : E_1$ and $y : E_2$, even if they are syntactically equal, are considered as distinct elements if E_1 and E_2 are colored differently. To continue our example from above, applying the completion rules to $\{x : \tilde{D}\}$ yields the system

$$\{x : \tilde{D}, x : A^{\text{yellow}}, x : \exists^{\text{red}} P.A^{\text{blue}}, xPy, y : A^{\text{blue}}\},$$

where the constraints $x : A^{\text{yellow}}$ and $y : A^{\text{blue}}$ involve two distinct concepts. Recall that the conditions of application of a rule to a given constraint system S require that S does not yet contain the constraints that the rule adds. In the colored

calculus we will ignore the coloring when checking the applicability of a rule. For example, the $\rightarrow_{\sqcap}$-rule is not applicable to the system $\{y : A^{\text{yellow}} \sqcap^{\text{red}} A^{\text{blue}}, y : A^{\text{green}}\}$. As a consequence, the colored calculus will behave similarly to the uncolored one.

LEMMA 6.2. *Let C be an $\mathcal{ALCN}$-concept and C' be a colored version of C. Then no derivation with the quasi-completion calculus issuing from $\{x : C\}$ is longer than the longest derivation with the colored calculus issuing from $\{x : C'\}$.*

Proof. Let $\mathcal{D}'$ be the set of derivations with the colored calculus issuing from $\{x : C'\}$ and let $\mathcal{D}$ be the set of derivations obtained from $\mathcal{D}'$ by forgetting about the colors. Obviously, any derivation with the quasi-completion calculus issuing from $\{x : C\}$ occurs as an element of $\mathcal{D}$. ∎

Next we want to derive an upper bound for the number of variables occurring in a constraint system derived from $\{x : C \sqcap D'\}$. In order to do so, we divide the variables in two groups. For the definition we need the following technicality.

Let E be a colored concept and F a subconcept of E. We say that F is *inside*-$\forall$ if E has a subconcept $\forall P.E'$ such that F is a subconcept of E', and *outside*-$\forall$ otherwise. Intuitively, F is outside-$\forall$ if it does not occur in the scope of a universal quantifier.

Now, suppose S has been derived from $\{x : C \sqcap D'\}$ by means of the colored quasi-completion rules. We say that a variable y is *outside*-$\forall$ *in* S if $C \sqcap D'$ has an outside-$\forall$ subconcept E such that S contains the constraint $y : E$ and that it is *inside*-$\forall$ *in* S otherwise. The idea underlying this definition is that a variable is outside-$\forall$ in S if it is x or if it has been generated through the $\rightarrow_{\exists}$-rule (because due to the particular structure of $C \sqcap D'$ no existential quantifier occurs in the scope of a universal quantifier) and it is inside-$\forall$ if it has been generated through the $\rightarrow_{\geqslant 1}$-rule.

As already pointed out before, the condition for applying the $\rightarrow_{\geqslant 1}$-rule guarantees that for a given variable y and role P either all P-successors are created through the $\rightarrow_{\exists}$-rule or there is a single P-successor created through the $\rightarrow_{\geqslant 1}$-rule. As a consequence, the $\rightarrow_{\leqslant}$-rule will only identify outside-$\forall$ variables. From this and the definition it follows that an outside-$\forall$ (inside-$\forall$) variable in S remains outside-$\forall$ (inside-$\forall$) in any system derived from S.

The next lemma gives an upper bound for the number of outside-$\forall$ variables.

LEMMA 6.3. *If S has been derived from $\{x : C \sqcap D'\}$ by means of the colored quasi-completion calculus, then*

1. *for every outside-$\forall$ subconcept E of $C \sqcap D'$ there is at most one constraint of the form $y : E$ in S;*
2. *S contains at most $\sharp(C \sqcap D')$ outside-$\forall$ variables.*

Proof. 1. The proof is by induction over the length of the derivation leading to S. Obviously, the claim is true for $\{x : C \sqcap D'\}$. Suppose the claim holds for S', and S is obtained from S' by the application of a quasi-completion rule. If the rule employed is the $\rightarrow_{\geqslant 1}$- or the $\rightarrow_{\leqslant}$-rule, then nothing has to be shown, since these

rules do not introduce any constraint of the form $y : E$. If it is the $\rightarrow_\forall$-rule, then a constraint $y : E$ has been added where E is inside-$\forall$.

Now, suppose that $S = S' \cup \{yPz, z : E\}$ has been obtained from S' by applying the $\rightarrow_\exists$-rule to the constraint $y : \exists P.E$ in S'. Since in $C \sqcap D'$ no existential quantifier occurs in the scope of a universal quantifier, the concepts $\exists P.E$ and E are outside-$\forall$. Assume that S contains a constraint $z' : E$ with $z' \neq z$. Then $z' : E$ is already contained in S'. Since concepts in S' are colored, this constraint must have been introduced by applying the $\rightarrow_\exists$-rule to a constraint $y' : \exists P.E$, i.e., a constraint containing the same concept $\exists P.E$. Since we can assume without loss of generality that y' has not been replaced with another variable by the $\rightarrow_\leqslant$-rule, we conclude that $y' : \exists P.E$ is in S'. But this contradicts the induction hypothesis that only one constraint of the form $y : \exists P.E$ is in S'. Thus, the assumption that S contains a constraint $z' : E$ was incorrect and the induction hypothesis holds also for S.

In case S is obtained using the $\rightarrow_\sqcap$- or the $\rightarrow_\sqcup$-rule a similar argument applies.

2. The claim follows immediately from (1), since D' has no more than $\sharp D'$ subconcepts. ∎

We now want to count the inside-$\forall$ variables.

Let y and z be variables in S. We say that z is an *all-inside-$\forall$ descendant of y* if z is an inside-$\forall$ descendant of y and the chain with starting point y and end point z contains only inside-$\forall$ variables, except possibly y. Observe that, since the "successor" relation on the variables in S is a tree with root x (see Lemma 4.1), for every inside-$\forall$ variable z there is a unique outside-$\forall$ variable y such that z is an all-inside-$\forall$ descendant of y. The variable y is the last outside-$\forall$ variable on the path from x to z.

We will count the inside-$\forall$ variables by counting for every outside-$\forall$ variable its all-inside-$\forall$ descendants.

LEMMA 6.4. *If S has been derived from $\{x : C \sqcap D'\}$ by means of the colored quasi-completion calculus and $n = \sharp(C \sqcap D')$, then*

1. *an outside-$\forall$ variable in S has at most $2n$ all-inside-$\forall$ descendants;*
2. *S contains at most $2n^2$ inside-$\forall$ variables.*

Proof. 1. Let us first show that S has the following property: if y is an outside-$\forall$ variable and F is a subconcept of $C \sqcap D'$ then y has at most one all-inside-$\forall$ descendant z such that $z : F$ is in S.

Again, the proof is by induction over the length of the derivation leading to S. Obviously, $\{x : C \sqcap D'\}$ has the above property. Now, suppose S' has this property and S has been obtained from S' by a quasi-completion step. It is easy to see that a $\rightarrow_\sqcap$- or a $\rightarrow_\sqcup$-step does not change this property. If the step is a $\rightarrow_{\geqslant 1}$-step then the property remains unchanged, too, because no constraint of the form $z : F$ is added. It also remains unchanged by an $\rightarrow_\leqslant$-step, because such a step identifies only outside-$\forall$ variables. Given the particular form of $C \sqcap D'$, a $\rightarrow_\exists$-step introduces only constraints for outside-$\forall$ variables. As seen above, since inside-$\forall$ variables are created by the $\rightarrow_{\geqslant 1}$-rule, a variable has for a given role P at most one inside-$\forall$ P-successor. Hence, the property remains invariant under the $\rightarrow_\forall$-rule.

Now, let y be an outside-$\forall$ variable in S. We first consider the number of all-inside-$\forall$ descendants z such that for some subconcept F of $C \sqcap D'$ the constraint $z : F$ is in S. This number is obviously bounded by the number of subconcepts of $C \sqcap D'$, which is at most n. Next we consider the number of all-inside-$\forall$ descendants z' such that S contains no constraint of the form $z' : F$. Such a variable must have been created by the $\rightarrow_{\geqslant 1}$-rule. Thus, if z' is such a variable and z is the predecessor of z', then S contains a constraint $z : (\geqslant m P)$. Hence, by the preceding argument, the number of such variables is bounded by n, too. This means that y has no more than $2n$ all-inside-$\forall$ descendants.

2. By Lemma 6.3, S contains at most n outside-$\forall$ variables. Since every inside-$\forall$ variable is an all-inside-$\forall$ descendant of a unique outside-$\forall$ variable, (1) implies that S contains at most $n(2n)$ inside-$\forall$ variables. ∎

LEMMA 6.5. *If S has been derived from $\{x : C \sqcap D'\}$ by means of the colored quasi-completion calculus and $n = \sharp(C \sqcap D')$, then*

1. *S contains at most $2n^2 + n$ variables and $(2n^2 + n)(n + 1)$ constraints;*
2. *any derivation leading to S comprises at most $O(n^3)$ steps.*

Proof. 1. By Lemma 6.3, S contains no more than n outside-$\forall$ variables and by Lemma 6.4, S contains at most $2n^2$ inside-$\forall$ variables. Summing up this yields at most $2n^2 + n$ variables.

Let N be the number of variables occurring in S. Since $C \sqcap D'$ has no more than n subconcepts, S contains at most Nn constraints of the form $y : E$. If z is a successor of y in S then S contains exactly one constraint of the form yPz, because in $\mathcal{ALUN}$ all roles are primitive. Since the "successor" relation on the variables in S is a tree, and a tree with N nodes has $N - 1$ edges, S contains $N - 1$ constraints of the form yPz. Thus, S contains no more than $Nn + N = N(n + 1) = (2n^2 + n)(n + 1)$ constraints.

2. Let us first consider the effect of $\rightarrow_{\leqslant}$-steps. As already seen, only outside-$\forall$ variables are identified during such a step. Suppose S' is a constraint system occurring during the derivation of S. If y, y' are distinct outside-$\forall$ variables and S' contains constraints $y : E$, $y' : E'$, where E, E' are outside-$\forall$ subconcepts, then Lemma 6.3 implies that E and E' are distinct.[3] Observe that E and E' need not be distinct if we do not take the coloring into account. As a consequence, these constraints remain distinct if one of the variables y, y' is replaced by the other. Hence, a $\rightarrow_{\leqslant}$-step does not reduce the number of constraints in S having the form $y : E$, where $E \neq \bot$ is an outside-$\forall$ subconcept. Whenever an outside-$\forall$ variable y is generated, a corresponding constraint $y : E$ is generated. Since the number of such constraints in S is at most n, we conclude that during the whole derivation of S at most n outside-$\forall$ variables have been created. Hence, during the derivation the $\rightarrow_{\leqslant}$-rule has been applied at most n times. All other rules have the property that applying them leads to at least one additional constraint.

The number of constraints created during the derivation of S equals the number of constraints contained in S plus the number of constraints eliminated by $\rightarrow_{\leqslant}$-steps. Since one variable occurs in no more than n constraints, S contains no

[3] Observe that E and E' need not be distinct if we do not take the coloring into account.

more than $(2n^2+n)n$ constraints. As seen above, a $\rightarrow_{\leqslant}$-step does not reduce the number of membership constraints. Since roles are primitive, a $\rightarrow_{\leqslant}$-step reduces the number of relationship constraints by one. Hence, no more than $(2n^2+n)n+n$ constraints have been created during the derivation of S.

This is an upper bound of the number of times rules other than the $\rightarrow_{\leqslant}$-rule have been used during the derivation of S. If we take into account that there are at most n applications of the $\rightarrow_{\leqslant}$-rule, we obtain that S has been derived with no more than $(2n^2+n)n+n+n$ quasi-completion steps. ▮

COROLLARY 6.6. *Any derivation from* $\{x : C \sqcap D'\}$ *by means of the ordinary quasi-completion calculus has length* $O(n^3)$ *where* $n = \sharp(C \sqcap D')$.

Let us summarize what we have achieved so far. Suppose C and D are $\mathcal{ALUN}$-concepts. Then C is not subsumed by D if and only if $C \sqcap D'$ is satisfiable, where D' is the negation normal form of $\neg D$. By Lemma 6.1 and the invariance property of the quasi-completion calculus, this is the case if and only if one can derive from $\{x : C \sqcap D'\}$ a quasi-complete constraint system that does not contain a simplified clash. By Corollary 6.6, the length of such a derivation is polynomial with respect to $\sharp(C \sqcap D')$, which obviously means that it is as well polynomial with respect to $\sharp C + \sharp D$.

PROPOSITION 6.7. *Nonsubsumption in* $\mathcal{ALUN}$ *can be decided in nondeterministic polynomial time.*

Proof. A nondeterministic algorithm can be devised as follows. It takes $\mathcal{ALUN}$-concepts C and D as input, computes the negation normal form D' of $\neg D$, derives a quasi-completion S of $\{x : C \sqcap D'\}$, and checks whether it is free of simplified clashes. As said before, transforming D into D' can be done within polynomial time. Computing S takes polynomial time, since one has to make polynomially many quasi-completion steps, each of which can be computed in polynomial time. The check for simplified clashes can be performed in polynomial time, too, since S has polynomial cardinality. ▮

6.3. Co-NP-*Hard Languages*

In Section 5 we have seen that deciding the unsatisfiability of $\mathcal{ALEN}$-concepts is NP-hard. Actually, Nebel's proof (Nebel, 1988) of co-NP-hardness of subsumption in $\mathcal{ALNR}$ yields a second lower bound of the complexity of unsatisfiability in $\mathcal{ALEN}$. Analyzing his reduction, one finds, first, that the subsumption problem to which he reduces the hitting set problem can be converted into an unsatisfiability problem and, second, that he uses role intersection only to mimic full existential quantification. Thus, his reduction can easily be modified to yield a proof of the following theorem.

THEOREM 6.8. *Unsatisfiability in* $\mathcal{ALEN}$ *is* co-NP-*hard.*

Moreover, in (Schmidt-Schauß and Smolka, 1991) it is noted that unsatisfiability in $\mathcal{ALU}$ easily simulates propositional unsatisfiability, hence unsatisfiability in $\mathcal{ALU}$ is co-NP-hard.

6.4. *Summary on* Co-NP-*Complete Languages*

Combining the fact that unsatisfiability in $\mathcal{AL}\mathcal{U}$ is co-NP-hard and the fact that subsumption in $\mathcal{AL}\mathcal{U}\mathcal{N}$ is in co-NP we obtain the following result.

THEOREM 6.9 (Co-NP-Completeness). *Subsumption and unsatisfiability are* co-NP-*complete problems for* $\mathcal{AL}\mathcal{U}$ *and* $\mathcal{AL}\mathcal{U}\mathcal{N}$, *independent of whether numbers are represented in q-ary notation for* $q=1$ *or* $q>1$.

7. POLYNOMIAL LANGUAGES

In Sections 5 and 6 we have shown that two different sources of complexity are inherent in inferences in concept languages. On the one hand, the interplay between universal and full existential quantifiers—where the latter might be present explicitly or realized implicitly through restricted existential quantification over subroles, as in $\mathcal{AL}\mathcal{R}$—is responsible for constraint systems of exponential size. On the other hand, the interaction of intersection and constructs that embody logical disjunction—like union, or "at most"-restriction in combination with full existential quantification—makes it necessary to generate an exponential number of constraint systems.

Consequently, one expects languages to be computationally tractable if they contain neither of the two sources of complexity. The largest such language is $\mathcal{AL}\mathcal{N}$. We will prove that in order to decide subsumption between two $\mathcal{AL}\mathcal{N}$-concepts it suffices to check a linear number of constraint systems each of which can be derived in polynomial time with the quasi-completion calculus using only deterministic rules. Building on this result, one can devise a polynomial-time algorithm for deciding subsumption in $\mathcal{AL}\mathcal{N}$. As in the previous section, the result will not depend on a particular notation of numbers. Therefore, we assume that the numbers occurring in number restrictions are represented in the standard binary notation.

Throughout this section, we assume that everywhere subconcepts of the form $\exists P.\top$ have been replaced with the equivalent concept $(\geqslant 1P)$.

Every $\mathcal{AL}\mathcal{N}$-concept can be rewritten in quadratic time to an equivalent concept of the form

$$C_1 \sqcap \cdots \sqcap C_k,$$

where none of the concepts C_i contains intersections, by using the rule

$$\forall P.(C \sqcap D) \quad \rightarrow \quad (\forall P.C) \sqcap (\forall P.D).$$

Suppose that C, D are $\mathcal{AL}\mathcal{N}$-concepts and that D is of the above form, that is, $D = D_1 \sqcap \cdots \sqcap D_k$ and none of the D_i's contains intersections. Deciding whether C is subsumed by D is equivalent to checking whether $C \sqcap \neg D = C \sqcap \neg(D_1 \sqcap \cdots \sqcap D_k)$ is unsatisfiable. Now, $C \sqcap \neg D$ is equivalent to $(C \sqcap \neg D_1) \sqcup \cdots \sqcup (C \sqcap \neg D_k)$. The latter is unsatisfiable if and only if each concept $C \sqcap \neg D_i$, $i \in 1..k$, is unsatisfiable, or equivalently, if each $C \sqcap D_i'$ is

unsatisfiable, where D_i' is the negation normal form of $\neg D_i$. Note that each concept D_i' has the form

$$\exists P_1.\exists P_2.\cdots.\exists P_m.E,$$

where E is of the form $\top$, $\bot$, A, $\neg A$, $\forall P.\bot$, $(\geqslant nP)$, or $(\leqslant nP)$. We call such a concept a *thread.*

LEMMA 7.1. *Let C be an $\mathcal{ALN}$-concept, D' be a thread, and $n = \sharp(C \sqcap D')$. Suppose that $C \sqcap D'$ is colored and that S has been derived from $\{x : C \sqcap D'\}$ with the colored quasi-completion calculus. Then*:

1. *every variable in S has at most one successor per role*;
2. *only deterministic rules are applicable to S*;
3. *S contains for every subconcept E of $C \sqcap D'$ at most one constraint of the form $x : E$*;
4. *S contains at most $2n$ constraints.*

Proof. 1. Successors of variables are created either by the $\rightarrow_{\geqslant 1}$-rule or by the $\rightarrow_{\exists}$-rule. As argued before, if a P-successor of a variable y has been introduced by the $\rightarrow_{\geqslant 1}$-rule, then y does not have any other P-successor and there never will be one in all systems derived from S. Moreover, only D' has existentially quantified subconcepts. Since D' is a thread, for any variable y in S there is only one constraint of the form $y : \exists P.E$. Thus, the $\rightarrow_{\exists}$-rule is applied at most once for any variable y.

2. Since no disjunction occurs in $C \sqcap D'$, the $\rightarrow_{\sqcup}$-rule is not applicable to S. Since every variable in S has at most one successor per role, the $\rightarrow_{\leqslant}$-rule is not either.

3. The proof is by induction over the length of the derivation leading to S. Obviously, $\{x : C \sqcap D'\}$ has the required property. Now, suppose S' has this property and S has been obtained from S' by a quasi-completion step. The property is not changed if $S' \rightarrow_{\geqslant 1} S$ because no constraint of the form $y : E$ is introduced into S. If $S' \rightarrow_{\sqcap} S$ then there is a constraint $y : E \sqcap E'$ in S such that neither $y : E$ nor $y : E'$ are in S' and $S = S' \cup \{y : E, y : E'\}$. (Observe that we can conclude the "neither-nor" because we have assumed that $C \sqcap D'$ is colored.) Assume that S contains in addition a constraint of the form $y' : E$ or $y' : E'$. Then this constraint is already contained in S'. Now, the coloring of concepts implies that S' contains also the constraint $y' : E \sqcap E'$, which contradicts the fact that S' has the required property. A similar argument applies to the case $S' \rightarrow_{\exists} S$. Finally, suppose that $S' \rightarrow_{\forall} S$. Then S' contains constraints $y : \forall P.E$ and yPz such that $S = S' \cup \{z : E\}$, but S' does not contain a constraint $y' : \forall P.E$ for some y' distinct from y. By (1) we know that z is the only P-successor of y. Hence, S does not contain another constraint of the form $z' : E$. Since by (2) we only have to consider deterministic rules, this yields the claim.

4. By (3) we know that S contains at most n membership constraints. Since every relationship constraint has been generated from a membership of the form

$y : (\geqslant mP)$ or $y : \exists P.E$, we conclude that S contains at most n relationship constraints. Hence, S contains at most $2n$ constraints. ▮

LEMMA 7.2. *Let C be an $\mathcal{AL}\mathcal{N}$-concept, D' be a thread, and let $n = \sharp(C \sqcap D')$. Then every derivation with the quasi-completion rules issuing from $\{x : C \sqcap D'\}$ comprises at most $2n$ steps.*

Proof. Since by 7.11 the $\rightarrow_{\leqslant}$-rule is not applicable to any constraint system derived from $\{x : C \sqcap D'\}$ with the colored quasi-completion rules, since each of the other rules adds at least one constraint, and since by 7.1.4 no such system contains more than $2n$ elements, it follows that no more than $2n$ rules have been applied in a colored derivation. By Lemma 6.2 no derivation with the original calculus comprises more than $2n$ steps.

THEOREM 7.3. *Subsumption and satisfiability in $\mathcal{AL}$ and $\mathcal{AL}\mathcal{N}$ can be decided in polynomial time, independently of the notation for numbers.*

Proof. It suffices to prove the claim for subsumption in $\mathcal{AL}\mathcal{N}$.

Let C, D be $\mathcal{AL}\mathcal{N}$-concepts, and let $n = \sharp C + \sharp D$. In order to decide whether C is subsumed by D, we check whether the concept $C \sqcap \neg D$ is unsatisfiable, transforming it in the following way: We first rewrite D into the concept $D_1 \sqcap \cdots \sqcap D_k$, where each D_i contains no intersection. Then we rewrite $C \sqcap \neg D$ as $(C \sqcap \neg D_1) \sqcup \cdots \sqcup (C \sqcap \neg D_k)$. Finally, we rewrite each concept $\neg D_i$ into its negation normal form D_i', which is a thread. This rewriting takes time at most quadratic in n.

Now for each of the threads D_i' we have to decide whether $C \sqcap D_i'$ is unsatisfiable. By Lemma 7.1 Part 2 this can be done by deriving a single quasi-completion of $\{x : C \sqcap D_i'\}$. By Lemma 7.2 the number of steps in such a derivation is linear in n. Since deciding the applicability of a rule and applying it takes time polynomial in n, computing the whole derivation takes polynomial time. Checking whether the resulting constraint system contains a clash takes again polynomial time. Therefore, deciding whether $C \sqcap D_i'$ is unsatisfiable can be done in polynomial time.

Since the number of concepts D_i' is linear in n, the whole check whether $C \sqcap \neg D$ is unsatisfiable takes time polynomial in n. ▮

8. SUMMARY OF COMPLEXITY RESULTS

We have studied the complexity of reasoning about concepts from $\mathcal{AL}$-languages, a family that covers most of the concept-forming constructs proposed in the literature. The basis for our investigation was a calculus for satisfiability checking, which can be understood as a tableaux calculus with a special control on the application of rules. For some $\mathcal{AL}$-languages that do not offer the full range of constructs we have optimized the general calculus. Employing these calculi for checking unsatisfiability and subsumption we have derived upper bounds for the complexity of the two inferences.

Studying the computational properties of the calculi, we have come up with a series of hardness results, that yield lower bounds on complexity. Combining upper and lower bounds, we obtain an (almost) complete picture of the complexity of

reasoning with concepts: for 15 out of the 16 $\mathcal{AL}$-languages we precisely characterized the satisfiability and the subsumption problem as either being polynomial or being complete for one of the classes NP, co-NP, or PSPACE. For each of these languages the complexity of unsatisfiability and subsumption are the same. Consequently, the equivalence and the disjointness problem have the same complexity (see Subsection 2.3).

We summarize the results for $\mathcal{AL}$-languages in Table 1. The table is to be read as follows: each row corresponds to one of the concept forming constructs union, full existential quantification over roles, number restrictions on roles, and intersection of roles, and each column corresponds to a language that extends the basic language $\mathcal{AL}$ by the constructs whose row is marked with "×". Languages are grouped according to the complexity of unsatisfiability and subsumption.

Previous results, noted in the last row, are the following. Complete results for both unsatisfiability and subsumption have been given (2) for $\mathcal{ALE}$ in (Donini *et al.*, 1992), and (4) for $\mathcal{ALC}$ in (Schmidt-Schauß and Smolka, 1991). Starred notes refer to the following partial results: In (Schmidt-Schauß and Smolka, 1991), it was shown that (1*) unsatisfiability in $\mathcal{AL}$ is polynomial, and that (3*) unsatisfiability in $\mathcal{ALU}$ is co-NP-complete. In (Nebel, 1988), it was shown that (5*) subsumption in $\mathcal{ALNR}$ is co-NP-hard. All other results represented by the table have been proved in this paper.

The only language that we could not completely characterize is $\mathcal{ALEN}$, which extends $\mathcal{AL}$ by full existential quantification and number restrictions. However, we have provided lower and upper complexity bounds. It follows from results reported in (Nebel, 1988) that unsatisfiability for $\mathcal{ALEN}$ is co-NP-hard (see Subsection 6.4), whereas in this paper we proved that unsatisfiability is also NP-hard (see Subsection 5.3). Since $\mathcal{ALEN}$ is a sublanguage of $\mathcal{ALCNR}$, the top element in our lattice, we also know that subsumption can be decided with polynomial space. For the concept language $\mathcal{FL}$, which does not fit into the lattice of $\mathcal{AL}$-languages, we have shown that subsumption is PSPACE-complete, thus improving upon an earlier result by Brachman and Levesque (Brachman and Levesque, 1984), who showed that it is co-NP-hard.

In Section 3.4, we have singled out two possible sources of complexity. First, complexity comes in through the interplay between conjunctive constructs, like

TABLE 1

Complexity of Unsatisfiability and Subsumption in $\mathcal{AL}$-Languages

$\mathcal{AL}$ +	poly-nomial	NP-complete		Co-NP-complete		PSPACE-complete							
$C \sqcup D$				×	×	×	×		×	×	×		×
$\exists R.C$		×	×			×			×	×		×	×
$(\geqslant nR), (\leqslant nR)$	×				×			×	×		×	×	×
$R \sqcap R'$			×	×			×	×		×	×	×	×
Previous results	(1*)	(2)		(3*)		(4)		(5*)					

Note. Previous results with an * refer to partial results.

intersection, and disjunctive constructs, like union or "at most" restrictions that are activated by full existential quantification. This source is reflected by co-NP-hardness of unsatisfiability and subsumption. Speaking in terms of our calculus, it may give rise to an exponential number of constraint systems that have to be explored for clashes. Less obviously, the interplay between universal and existential quantifiers is a source of complexity too. This is reflected by the NP-hardness results. On the level of the completion calculus, this source may lead to constraint systems that contain exponentially many variables. For all $\mathcal{AL}$-languages where both sources of complexity are present—except $\mathcal{ALEN}$—it has been shown that the unsatisfiability and the subsumption problem are PSPACE-hard.

We remark that in Sections 3.4 and 4, we have assumed unary encoding of numbers occurring in number restrictions. We feel that this assumption is justified because it allows one to translate concept expression into first-order formulas in polynomial time, so that our complexity results refer also to the decision problem for the fragments of first-order logic that correspond to concept languages. Without this assumption, however, we would not have been able to prove that satisfiability in $\mathcal{ALCNR}$ is in PSPACE. It is open whether this upper bound still holds if we allow for binary encoding of numbers.

Since the complexity results—with an exception for $\mathcal{ALEN}$—are all completeness results, they show that our calculi are optimal for the problems they solve in that they do not use more time or space resources than allowed by the complexity class for which the problem is hard.

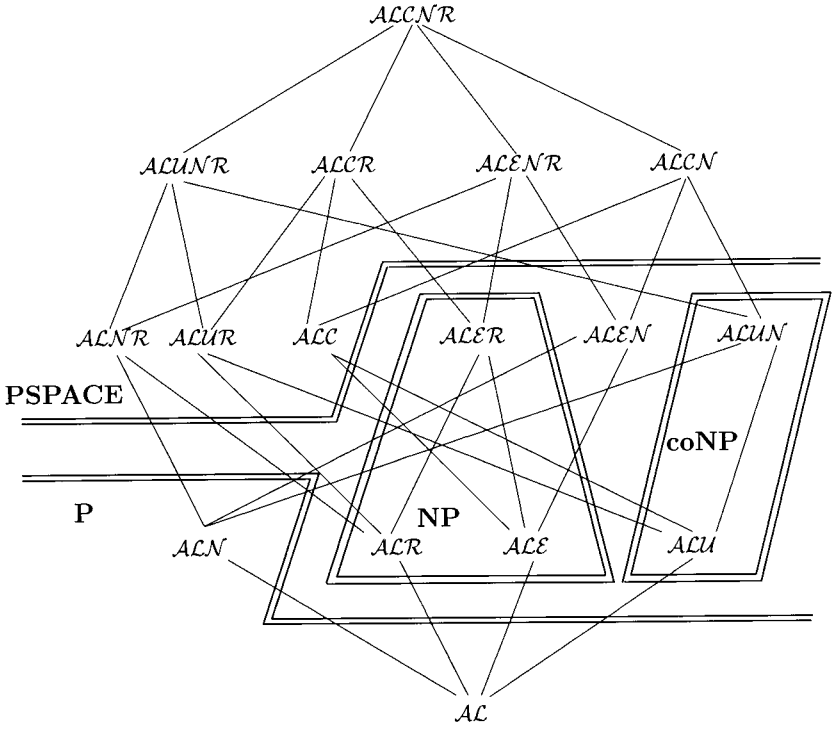

FIG. 4. Complexity of unsatisfiability and subsumption in $\mathcal{AL}$-languages.

In Fig. 4 we represent the overall picture in a lattice whose bottom language is $\mathcal{AL}$ and top language is $\mathcal{ALCNR}$. Double thick lines indicate the borders of complexity classes, while thin lines represent the relationships between $\mathcal{AL}$-languages.

9. CONCLUSION

In this paper, we have shown that a general approach to inferences in concept languages based on a modified tableaux calculus for first order logic leads to decision procedures that—in terms of worst case complexity—are very often optimal for the problem they solve.

Originally, the complexity analysis of terminological reasoning started with the goal to identify languages for which subsumption can be decided in polynomial time (Brachman and Levesque, 1984). After a deep analysis of the tractabilitity threshold (Donini *et al.*, 1991), it turned out that many interesting constructs lead to concept languages whose reasoning problems are intractable.

One might conclude from the results in this paper that terminological reasoning in all its variants is infeasible. Such a conclusion would implicitly assume that the complexity analysis is intended to restrict the practical use of concept languages to those where satisfiability and subsumption can be computed in polynomial time. However, it is our opinion that the study of the complexity of concept languages goes far beyond a mere classification of tractable and intractable languages.

First of all, the results developed so far refer to the computational complexity in the worst case, which represents only one aspect to be taken into account when considering the practical use of concept languages. Notice that, as pointed out in (Nebel, 1990), another aspect that deserves further investigation is the characterization of the average cases occurring in practice. Second, the techniques used for the complexity analysis have provided the formal basis for the design of effective algorithms for computing subsumption and unsatisfiability in a large class of concept languages. Finally, in the design of deduction procedures for knowledge representation systems based on concept languages, one can take advantage of the knowledge about the complexity of subsumption, by isolating difficult cases and using specialized efficient algorithms whenever possible.

Received October 4, 1995; final manuscript received January 14, 1997

REFERENCES

Baader, F., Franconi, E., Hollunder, B., Nebel, B., and Profitlich, H.-J. (1994), An empirical analysis of optimization techniques for terminological representation systems or: Making KRIS get a move on, *Appl. Intell.* **4**, 109–132. [Special Issue on Knowledge Base Management]

Baader, F., and Hollunder, B. (1991), $\mathcal{KRIS}$: $\mathcal{K}$nowledge $\mathcal{R}$epresentation and $\mathcal{I}$nference $\mathcal{S}$ystem, *SIGART Bull.* **2**(3), 8–14.

Bell, J. L., and Machover, M. (1977), "A Course in Mathematical Logic," North-Holland, Amsterdam.

Brachman, R. J. (1979), On the epistemological status of semantic networks, *in* "Associative Networks" (N. V. Findler, Ed.), pp. 3–50, Academic Press, San Diego.

Brachman, R. J. (1985), I lied about the trees, *AI Magazine* **6**(3), 80–93.

Brachman, R. J. (1992), "Reducing" CLASSIC to practice: Knowledge representation meets reality, *in* "Proceedings of the Third International Conference on the Principles of Knowledge Representation and Reasoning (KR-92)," pp. 247–258, Morgan Kaufmann, Los Altos, CA.

Brachman, R. J., Fikes, R. E., and Levesque, H. J. (1983), KRYPTON: A functional approach to knowledge representation, *IEEE Comput.* **16**(10), 67–73.

Brachman, R. J., and Levesque, H. J. (1984), The tractability of subsumption in frame-based description languages, *in* "Proceedings of the Fourth National Conference on Artificial Intelligence (AAAI-84)," pp. 34–37.

Brachman, R. J., and Levesque, H. J. (1985), "Readings in Knowledge Representation," Morgan Kaufmann, Los Altos, CA.

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

de Rijke, M., and van der Hoek, W. (1995), Counting objects in generalized quantifier theory, modal logic, and knowledge representation, *J. Logic Comput.* **5**(3), 325–346.

Donini, F. M., Hollunder, B., Lenzerini, M., Spaccamela, A. M., Nardi, D., and Nutt, W. (1992), The complexity of existential quantification in concept languages, *Artificial Intell.* **2–3**, 309–327.

Donini, F. M., Lenzerini, M., Nardi, D., and Nutt, W. (1991), Tractable concept languages, *in* "Proceedings of the Twelfth International Joint Conference on Aritificial Intelligence (IJCAI-91)," Sydney, pp. 458–463.

Fitting, M. (1990), "First-Order Logic and Automated Theorem Proving," Springer-Verlag, Berlin/New York.

Garey, M. R., and Johnson, D. S. (1979), "Computers and Intractability—A Guide to NP-Completeness," Freeman, San Francisco.

Halpern, J. Y., and Moses, Y. (1992), A guide to completeness and complexity for modal logics of knowledge and belief, *Artificial Intell.* **54**(3), 319–380.

Heinsohn, J., Kudenko, D., Nebel, B., and Profitlich, H.-J. (1994), An empirical analysis of terminological representation systems, *Artificial Intell.* **68**(2), 367–396.

Hollunder, B., and Nutt, W. (1990), "Subsumption Algorithms for Concept Languages," Technical Report RR-90-04, Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI), Kaiserslautern, Germany.

Hollunder, B., Nutt, W., and Schmidt-Schauß, M. (1990), Subsumption algorithms for concept description languages, *in* "Proceedings of the Ninth European Conference on Artificial Intelligence (ECAI-90), London," pp. 248–353, Pitman.

Johnson, D. S. (1990), A catalog of complexity classes, *in* "Handbook of Theoretical Computer Science" (J. van Leeuwen, Ed.), Vol. A, Chap. 2, Elsevier (North Holland), Amsterdam.

Kanellakis, P. C. (1990), Elements of relational database theory, *in* "Handbook of Theoretical Computer Science" (J. van Leeuwen, Ed.), Vol. A, Chap. 17, Elsevier (North Holland), Amsterdam.

Ladner, R. E. (1977), The computational complexity of provability in systems of modal propositional logic, *SIAM J. Comput.* **6**(3), 467–480.

Levesque, H. J. (1984), Foundations of a functional approach to knowledge representation, *Artificial Intell.* **23**, 155–212.

Levesque, H. J., and Brachman, R. J. (1987), Expressiveness and tractability in knowledge representation and reasoning, *Computat. Intell.* **3**, 78–93.

Lipkis, T. (1982), A KL-ONE classifier, *in* "Proceedings, 1981 KL-ONE Workshop, Cambridge, MA" (J. Schmolze and R. Brachman, Eds.), pp. 128–145. [The proceedings have been published as BBN Report No. 4842 and as AI Technical Report 4, Schlumberger Palo Alto Research]

Nebel, B. (1988), Computational complexity of terminological reasoning in BACK, *Artificial Intell.* **34**(3), 371–383.

Nebel, B. (1990), Terminological reasoning is inherently intractable, *Artificial Intell.* **43**, 235–249.

Nutt, W. (1993), "Algorithms for Constraints in Deduction and Knowledge Representation," Ph.D. thesis, Technische Fakultät der Universität des Saarlands, Saarbrücken, Germany.

Patel-Schneider, P. F. (1989), Undecidability of subsumption in NIKL, *Artificial Intell.* **39**, 263–272.

Patel-Schneider, P. F., and Swartout, B. (1993), Working version (draft): Description logic specification from the KRSS effort, unpublished manuscript.

Schild, K. (1988), "Undecidability of Subsumption in $\mathcal{U}$," Technical Report KIT-Report 67, Fachbereich Informatik, Technische Universität Berlin, Berlin, Germany.

Schild, K. (1991), A correspondence theory for terminological logics: Preliminary report, *in* "Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91), Sydney," pp. 466–471.

Schmidt-Schauß, M. (1989), Subsumption in KL-ONE is undecidable, *in* "Proceedings of the First International Conference on the Principles of Knowledge Representation and Reasoning (KR-89)" (R. J. Brachman, H. J. Levesque, and R. Reiter, Eds.), pp. 421–431, Morgan Kaufmann, Los Altos, CA.

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

Schmolze, J. G., and Brachman, R. J., Eds. (1982), "Proceedings, 1981 KL-ONE Workshop," BBN Report 4842, Bolt, Beranek, & Newman, Cambridge, MA. (Also available as AI Technical Report 4, Schlumberger Palo Alto Research, May 1982)

SIGART (1991), *SIGART Bull.*, Special issue on implemented knowledge representation and reasoning systems.

Smolka, G. (1988), "A Feature Logic with Subsorts," Technical Report 33, IWBS, IBM Deutschland, P.O. Box 80 08 80, D-7000 Stuttgart 80, Germany.

Smullyan, R. M. (1968), "First Order Logic," Springer-Verlag, Berlin.

Woods, W. A. (1975), What's in a link: Foundations for semantic networks, *in* "Representation and Understanding: Studies in Cognitive Science" (D. G. Bobrow and A. M. Collins, Eds.), pp. 35–82, Academic Press, San Diego. (Republished in Brachman and Levesque, 1985)

Woods, W. A., and Schmolze, J. G. (1992), The KL-ONE family, *in* "Semantic Networks in Artificial Intelligence" (F. W. Lehmann, Ed.), pp. 133–178, Pergamon, Elmsford, NY. (Published as a special issue of *Computers & Mathematics with Applications*, Vol. 23, No. 2–9)