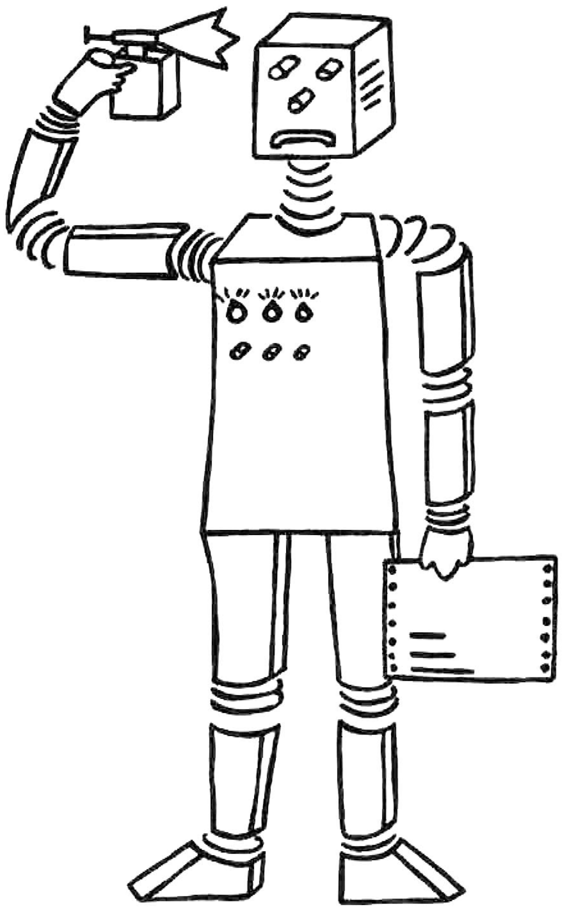
SEKI-REPORT

Artificial
Intelligence
Laboratories

Fachbereich Informatik
Universität Kaiserslautern
Postfach 3049
D-6750 Kaiserslautern 1, W. Germany



Attributive Concept Descriptions with Unions and Complements

Manfred Schmidt-Schauß & Gerd Smolka

SEKI Report SR-88-21

SEKI Report SR-88-21, December 1988
FB Informatik, Universität Kaiserslautern, West Germany

Attributive Concept Descriptions with Unions and Complements

Manfred Schmidt-Schauß[†] *and Gert Smolka*[‡]

† *DFKI, Universität Kaiserslautern, West Germany*

‡ *IWBS, IBM Deutschland, West Germany*

Abstract. This paper investigates the consequences of adding unions and complements to the attributive concept descriptions employed in KL-ONE-like knowledge representation languages. It is shown that deciding consistency and subsumption of such descriptions are PSPACE-complete problems that can be decided with linear space.

Smolka has been funded by the EUREKA Project Protos (EU 56).

Addresses for correspondence: Manfred Schmidt-Schauß, FB Informatik, Universität Kaiserslautern, 6750 Kaiserslautern, West Germany, schauss@uklirb.uucp; Gert Smolka, IWBS, IBM Deutschland, Postfach 800880, 7000 Stuttgart 80, West Germany, smolka@ds0lilog.bitnet.

1 Introduction

The idea of a semantic network has led to the development of a family of logic-based knowledge representation languages known as KL-ONE dialects [Brachman/Schmolze 85, Levesque/Brachman 87, Nebel 88]. All members of the KL-ONE family offer attributive concept descriptions that are interpreted as sets and employ two kinds of symbols, called concepts and roles. Concepts are interpreted as sets and roles are interpreted as binary relations. The distinctive formation rules for concept descriptions are $\forall R\colon C$ and $\exists R\colon C$, where R is a role and C is a concept description. The description $\forall R\colon C$ can be read as "all objects for which all R's are in C", and the description $\exists R\colon C$ can be read as "all objects for which there is an R in C". Given an interpretation $\mathcal{I}$ for the occurring concept and role symbols, these descriptions are interpreted as the sets

$$\mathcal{I}[\![\forall R\colon C]\!] = \{a \in D^{\mathcal{I}} \mid \forall\, (a,b) \in \mathcal{I}[\![R]\!] : \quad b \in \mathcal{I}[\![C]\!]\}$$

and

$$\mathcal{I}[\![\exists R\colon C]\!] = \{a \in D^{\mathcal{I}} \mid \exists\, (a,b) \in \mathcal{I}[\![R]\!] : \quad b \in \mathcal{I}[\![C]\!]\},$$

where $D^{\mathcal{I}}$ is the domain of $\mathcal{I}$. Since concept descriptions denote sets, it's clear how to provide for concept intersection, concept union, and concept complement. We will speak of an attributive concept description language (ACD-language, for short) if at least $\forall R\colon C$, $\exists R\colon \top$ (where $\mathcal{I}[\![\top]\!] = D^{\mathcal{I}}$ in every interpretation $\mathcal{I}$) and concept intersection $C \sqcap D$ are available.

So far, KL-ONE concept descriptions with unions and complements have not been investigated theoretically nor are their present in existing implementations. To get an idea of the rich expressional repertoire opened up by concept unions and complements, let's have a look at a couple of examples.

The class of humans is partioned into the disjoint subclasses of women and men. This can be expressed by the terminological axioms

$$\begin{aligned} \mathit{woman} &\sqsubseteq \mathit{human} \\ \mathit{man} &= \mathit{human} - \mathit{woman}, \end{aligned}$$

where the concept difference *human* $-$ *woman* stands for *human* $\sqcap \neg$*woman*.

Animals that are featherless bipeds are humans. This can be expressed with the axiom

animal $\sqsubseteq$ *featherless_biped* $\rightarrow$ *human*,

where the concept implication *featherless_biped* $\rightarrow$ *human* is an abbreviation for $\neg$*featherless_biped* $\sqcup$ *human*.

The basic reasoning service provided by a KL-ONE system is a subsumption test, which, given two concept descriptions C and D, checks whether every interpretation interprets C as a subset of D. For the minimal ACD-language offering only $\forall R{:}C$, $\exists R{:}\top$ and $C \sqcap D$, which is called $\mathcal{FL}^-$ in [Levesque/Brachman 87], subsumption can be decided in quadratic time. For the more general language $\mathcal{ALE}$ providing $\exists R{:}C$ instead of $\exists R{:}\top$, we could not find a complexity result in the literature. In this paper we will give a linear-space subsumption checking algorithm for $\mathcal{ALE}$.

A common belief of the KL-ONE community says that the subsumption test must be of polynomial complexity to be of use in practical systems. This, of course, means that the expressiveness of concept descriptions must be restricted drastically. Interpreted technically, the quest for polynomial complexity restricts ACD-languages to be less powerful than propositional logic. In particular, this requirement rules out the possibility of expressing unions and complements of concepts. Moreover, based on the results of this paper, we conjecture that already $\mathcal{ALE}$ has a nonpolynomial subsumption problem.

To overcome this obvious lack of expressive power, so-called hybrid knowledge representation systems [Brachman et al. 85, Nebel/Luck 88, Patel-Schneider 84, Vilain 85] have been proposed, which combine an ACD-language with an assertional language that is typically a suitable subset of predicate logic. One way to bring in concept descriptions is to allow for formulas $x{:}C$ that are satisfied if the value of the variable x is in the set denoted by the concept description C. The assertional reasoner then creates conjunctions $x{:}C$ & $x{:}D$, which are passed to the terminological reasoner for consistency checking (often called unification). This is done by simplifying the intersection $C \sqcap D$ to a suitable normal form exhibiting whether $C \sqcap D$ is consistent, that is, is interpreted as a nonempty set by at least one interpretation. Since the terminological reasoner works without knowledge of the assertions, the assertions must yield a conservative extension of the concept description level, a fact that seems to have been ignored in the KL-ONE literature. Höhfeld and Smolka [88] investigate the mathematical foundations of hybrid systems whose assertional language is restricted to definite clauses and show that for this class of hybrid systems the conservative extension property always holds.

For many interesting applications a hybrid representation formalism with a sufficiently expressive assertional language is required. Since the overall reasoning performance of such a system depends crucially on the reasoning performance of the assertional system, which typically is nonpolynomial, the popular insistence on polynomial complexity ACD-languages seems to us rather shortsighted. In the context of hybrid systems, ACD-languages with an NP-complete consistency problem seem to be rather well-behaved.

If the concept description level doesn't provide for unions and complements, they will show up at the assertional level as disjunctions and negations. If the assertional level is undecidable, then lifting complements to negations means to make undecidable negations out of decidable complements. Furthermore, the definite clause sublanguage of predicate logic, which is employed in logic programming and is particularly well-behaved, does not even allow for negations and disjunctions. Thus, if generalized definite clauses are employed as assertions [Höhfeld/Smolka 88], negations and disjunctions can only be expressed at the concept description level as complements and unions.

The conflict between expressive power and computational tractability has led to the use of incomplete algorithms in existing theorem proving and knowledge representation systems. The usefulness of such systems certainly increases if one has a good understanding of the sources of incompleteness, and we believe that the study of complete methods contributes to this understanding and is essential in the development of better, and better described, partial methods.

There is a second family of ACD-languages, which are known as feature descriptions [Kaplan/Bresnan 82, Rounds/Kasper 86, Johnson 87] and have been developed by computational linguists for use in so-called unification grammars.

Closely related is Aït-Kaci's [86] ψ-term calculus that employs a subsort lattice and is geared towards knowledge representation. Recently, Smolka [88] has shown that the subsumption relation employed with feature descriptions can be obtained by a model theoretic semantics given in the same way as outlined above for KL-ONE concept descriptions. The distinct difference between the two families is that feature descriptions are more restrictive in that they only allow functional roles, which are called features hereafter. Furthermore, role inclusions (called role value maps in the KL-ONE world), whose semantics is given by

$$\mathcal{I}[\![r_1 \sqsubseteq r_2]\!] = \{a \in D^{\mathcal{I}} \mid \forall\, (a,b) \in \mathcal{I}[\![r_1]\!].\ \ (a,b) \in \mathcal{I}[\![r_2]\!]\},$$

are essential for linguistic applications, while they are of secondary importance in KL-ONE applications. Restricting roles to features causes a dramatic difference in the hardness of the subsumption problem if role inclusions are available:

- Smolka [88] shows that an ACD-language with role inclusions has a polynomial subsumption problem, provided all roles are interpreted as partial functions; furthermore, he shows that an ACD-language with unions, complements and role inclusions has a Co-NP-complete subsumption problem, again provided all roles are interpreted as partial functions

- Schmidt-Schauß [88] shows that an ACD-language with role inclusions has an undecidable subsumption problem if roles are not restricted to partial functions, even if unions and complements are not available.

In this paper we examine the ACD-language $\mathcal{ALC}$ ($\forall R\colon C$, $\exists R\colon C$, $C \sqcap D$, $C \sqcup D$, and $\neg C$) and prove that its consistency and subsumption problems are PSPACE-complete. We give a linear-space exponential-time algorithm deciding the consistency of $\mathcal{ALC}$-concept descriptions. The consistency checking algorithm also yields a subsumption checking algorithm since C is subsumed by D if and only if $C \sqcap \neg D$ is inconsistent. We also prove more specific complexity results for three sublanguages of $\mathcal{ALC}$ including $\mathcal{ALE}$.

The focus of this paper is on consistency checking, which is the reasoning operation on concept descriptions the assertional reasoner in a hybrid knowledge representation relies on. Our paper complements the existing KL-ONE literature that concentrates on subsumption checking. Although in the case of $\mathcal{ALC}$ consistency and subsumption checking reduce to each other in linear time, consistency checking leads to technically simpler proof methods.

Acknowledgement. We are grateful to Bernhard Nebel for stimulating discussions and his willingness to share our repeated excitement about several consistency checking algorithms for $\mathcal{ALE}$, which one after the other all turned out to be garbage since they were either incomplete or nonpolynomial. The second author would also like to thank his colleagues in the LILOG project of IBM Deutschland for providing the stimulating environment motivating the research reported in this paper.

2 Attributive Concept Descriptions

Let two disjoint alphabets of symbols, called **concepts** and **roles**, respectively, be given. We assume that $\top$ is a concept symbol. The letters A and B will always denote concept symbols and the letter R will always denote a role symbol.

The concept descriptions of the ACD-language $\mathcal{ALC}$ are given by the abstract syntax rule

$$C, D \longrightarrow A \mid \forall R{:}C \mid \exists R{:}C \mid C \sqcap D \mid C \sqcup D \mid \neg C.$$

An **interpretation** $\mathcal{I} = (D^{\mathcal{I}}, \mathcal{I}[\![\cdot]\!])$ consists of a set $D^{\mathcal{I}}$ (the **domain** of $\mathcal{I}$) and a function $\mathcal{I}[\![\cdot]\!]$ (the **interpretation function** of $\mathcal{I}$) that maps every concept description to a subset of $D^{\mathcal{I}}$, every role symbol to a subset of $D^{\mathcal{I}} \times D^{\mathcal{I}}$, and satisfies the following equations:

$$\begin{aligned}
&\mathcal{I}[\![\top]\!] = D^{\mathcal{I}}\\
&\mathcal{I}[\![\forall R{:}C]\!] = \{a \in D^{\mathcal{I}} \mid \forall\, (a,b) \in \mathcal{I}[\![R]\!] :\ \ b \in \mathcal{I}[\![C]\!]\}\\
&\mathcal{I}[\![\exists R{:}C]\!] = \{a \in D^{\mathcal{I}} \mid \exists\, (a,b) \in \mathcal{I}[\![R]\!] :\ \ b \in \mathcal{I}[\![C]\!]\}\\
&\mathcal{I}[\![C \sqcap D]\!] = \mathcal{I}[\![C]\!] \cap \mathcal{I}[\![D]\!]\\
&\mathcal{I}[\![C \sqcup D]\!] = \mathcal{I}[\![C]\!] \cup \mathcal{I}[\![D]\!]\\
&\mathcal{I}[\![\neg C]\!] = D^{\mathcal{I}} - \mathcal{I}[\![C]\!].
\end{aligned}$$

A concept description C is **consistent** if there exists an interpretation $\mathcal{I}$ such that $\mathcal{I}[\![C]\!]$ is nonempty. A concept description C is **subsumed** by a concept description D if $\mathcal{I}[\![C]\!] \subseteq \mathcal{I}[\![D]\!]$ for every interpretation $\mathcal{I}$. A concept description C is **equivalent** to a concept description D if $\mathcal{I}[\![C]\!] = \mathcal{I}[\![D]\!]$ for every interpretation $\mathcal{I}$.

Proposition 2.1. *A concept description C is subsumed by a concept description D if and only if the concept description $C \sqcap \neg D$ is inconsistent.*

Hence a consistency checking algorithm for $\mathcal{ALC}$ can also be used for testing subsumption in $\mathcal{ALC}$ and vice versa.

The syntax of $\mathcal{ALC}$ is redundant. For instance, $\top$ is equivalent to $A \sqcup \neg A$ for every concept symbol A, $\exists R\colon C$ is equivalent to $\neg\forall R\colon \neg C$ and $C \sqcup D$ is equivalent to $\neg(\neg C \sqcap \neg D)$.

The redundant syntax provides for the simplification of complex complements to **simple complements** of the form $\neg A$, where A is a concept symbol. This can be done in linear time by the following rules reducing concept descriptions to equivalent concept descriptions:

$$\begin{aligned}
\neg(\forall R\colon C) &\rightarrow \exists R\colon \neg C\\
\neg(\exists R\colon C) &\rightarrow \forall R\colon \neg C\\
\neg(C \sqcap D) &\rightarrow \neg C \sqcup \neg D\\
\neg(C \sqcup D) &\rightarrow \neg C \sqcap \neg D\\
\neg\neg C &\rightarrow C.
\end{aligned}$$

We call a concept description **simple** if it contains only simple complements.

Proposition 2.2. *For every concept description one can compute in linear time an equivalent simple concept description.*

We define three sublanguages of $\mathcal{ALC}$:

1. $\mathcal{ALE}$ is obtained from $\mathcal{ALC}$ by allowing only simple concept descriptions containing no unions

2. $\mathcal{ALU}$ is obtained from $\mathcal{ALC}$ by allowing only simple concept descriptions and restricting existential role quantifications to the form $\exists R{:}\top$

3. $\mathcal{AL}$ is obtained from $\mathcal{ALC}$ by allowing only simple concept descriptions containing no unions and restricting existential role quantifications to the form $\exists R{:}\top$.

Note that $\mathcal{AL}$ is the intersection of $\mathcal{ALE}$ and $\mathcal{ALU}$. The names of these languages are cooked up as follows: $\mathcal{ALU}$ is obtained from $\mathcal{AL}$ by adding unions, $\mathcal{ALE}$ is obtained from $\mathcal{AL}$ by adding general existential role quantifications, and $\mathcal{ALC}$ is obtained from $\mathcal{AL}$ by adding general complements. We consider $\mathcal{AL}$ to be the minimal sensible ACD-language. Seen from the KL-ONE perspective, simple complements provide the possibility of declaring "primitive concepts" to be disjoint.

Proposition 2.3. *Deciding consistency of $\mathcal{ALU}$-concept descriptions is NP-hard, and deciding subsumption of $\mathcal{ALU}$-concept descriptions is co-NP-hard. This holds already for descriptions not containing role symbols.*

Proof. Since C is inconsistent if and only if C is subsumed by $\neg\top$, it suffices to show that checking consistency is NP-hard.

It is well-known that deciding the satisfiability of propositional formulas in conjunctive normal form (CNF) is an NP-complete problem (see, for instance, [Garey/Johnson 79]). A propositional formula in CNF can be seen as an $\mathcal{ALU}$-concept description by taking propositional variables as concept symbols, conjunctions as intersections, disjunctions as unions, and negations as complements.

Let $F = F_1 \wedge \ldots \wedge F_n$ be a propositional formula in CNF, where every F_i is a disjunction of literals. Obviously, F is satisfiable if and only if one can choose in every F_i a literal L_i such that $L_1, \ldots, L_n$ don't contain a complementary pair.

Suppose F is satisfiable. Then there exist $L_1, \ldots, L_n$ as specified above. Let $\mathcal{I}$ be the interpretation such that $D^{\mathcal{I}} = \{1\}$, $\mathcal{I}[\![A]\!] = \{1\}$ if $A = L_i$ for some i, $\mathcal{I}[\![A]\!] = \emptyset$ otherwise, and $\mathcal{I}[\![R]\!] = \emptyset$ for every role symbol R. Then $\mathcal{I}[\![F]\!] = \mathcal{I}[\![L_i]\!] = \{1\}$ for $i \in 1..n$, which shows that F is a consistent concept description.

Suppose F is a consistent concept description. Then there exists an interpretation $\mathcal{I}$ and an $a \in D^{\mathcal{I}}$ such that $a \in \mathcal{I}[\![F]\!]$. Hence every F_i contains a literal L_i such that $a \in \mathcal{I}[\![L_i]\!]$. Thus $L_1, \dots, L_n$ don't contain a complementary pair, which shows that F is satisfiable. $\square$

In this paper we are going to prove the following results:

- checking consistency and subsumption of $\mathcal{ALC}$-concept descriptions are PSPACE-complete problems that can be decided with linear space
- inconsistency (not consistency) of $\mathcal{ALE}$-concept descriptions can be decided in nondeterministic linear time (this is the best upper bound we could prove; we could not find any upper bound for $\mathcal{ALE}$ in the literature)
- checking consistency of $\mathcal{ALU}$-concept descriptions is an NP-complete problem (we have already shown that checking subsumption of $\mathcal{ALU}$-concept descriptions is an NP-hard problem)
- consistency of $\mathcal{AL}$-concept descriptions can be checked in linear time.

The relationship between our ACD-languages and the ACD-languages $\mathcal{FL}$ and $\mathcal{FL}^-$ in [Levesque/Brachman 87] is as follows:

$$\begin{aligned}
\mathcal{AL} &\equiv \mathcal{FL}^- + \textit{simple complements}\\
\mathcal{ALC} &\equiv \mathcal{FL}^- + \textit{general complements}\\
&\equiv \mathcal{FL} + \bot + \textit{SELF},
\end{aligned}$$

where $\bot$ is a concept symbol and $\textit{SELF}$ is a role symbol such that

$$\begin{aligned}
\mathcal{I}[\![\bot]\!] &= \emptyset\\
\mathcal{I}[\![\textit{SELF}]\!] &= \{(a,b) \in D^{\mathcal{I}} \mid a = b\}
\end{aligned}$$

for every interpretation $\mathcal{I}$. The first equation is obvious; the other two equations follow from the equivalences:

$$\begin{aligned}
\textit{(RESTRICT (RESTRICT R C) D)} &\sim \textit{(RESTRICT R (AND C D))}\\
\textit{(ALL (RESTRICT R C) D)} &\sim \forall R{:}(\neg C \sqcup D)\\
\textit{(SOME (RESTRICT R C))} &\sim \exists R{:}C\\
\textit{(ALL (RESTRICT SELF C)} \perp\textit{)} &\sim \neg C\\
\forall \mathit{SELF}{:}C &\sim C\\
\exists \mathit{SELF}{:}C &\sim C.
\end{aligned}$$

We will show that every $\mathcal{FL}$-concept description is consistent. Hence, $\mathcal{FL}^-$ is a proper sublanguage of $\mathcal{AL}$ and $\mathcal{FL}$ is a proper sublanguage of $\mathcal{ALC}$.

Finally, we discuss the relationship between $\mathcal{ALC}$ and Feature Logic [Smolka 88]. An $\mathcal{ALC}$-interpretation $\mathcal{I}$ is called a **feature interpretation** if it interprets every role symbol as a partial function, that is, every role symbol R satisfies

$$(a,b) \in \mathcal{I}[\![R]\!] \ \wedge \ (a,c) \in \mathcal{I}[\![R]\!] \ \Rightarrow \ b = c$$

for all $a, b, c \in D^{\mathcal{I}}$. Let $\mathcal{FALC}$ be the ACD-language having the concept descriptions of $\mathcal{ALC}$ but admitting only the feature interpretations of $\mathcal{ALC}$. Then $\mathcal{FALC}$ is a sublanguage of Feature Logic, as can be seen from the two equivalences:

$$\begin{aligned}
\forall R{:}C &\sim R{:}C \sqcup \neg R{:}\top\\
\exists R{:}C &\sim R{:}C.
\end{aligned}$$

Feature Logic can be obtained from $\mathcal{FALC}$ by adding agreements (a construct corresponding to KL-ONE's role value maps) and constants (individual concepts). Consistency in Feature Logic is NP-complete [Smolka 88]. Since $\mathcal{FALC}$ is a sublanguage of Feature Logic, we know by Proposition 2.3 that $\mathcal{FALC}$ also has an NP-complete consistency problem. Since Feature Logic and $\mathcal{FALC}$ have general complements, their subsumption problems are thus Co-NP-complete. Adding agreements to $\mathcal{FALC}$ doesn't change $\mathcal{FALC}$'s complexity, while adding role value maps to $\mathcal{ALC}$ causes undecidability [Schmidt-Schauß 88].

A crucial difference between $\mathcal{FALC}$ and $\mathcal{ALC}$ is the fact that $\mathcal{FALC}$ has a disjunctive normal form while $\mathcal{ALC}$ does not. To see this, note that the equivalences

$$\begin{aligned}(C \sqcup D) \sqcap E &\sim (C \sqcap E) \sqcup (D \sqcap E)\\ \exists R{:}(C \sqcup D) &\sim (\exists R{:}\, C) \sqcup (\exists R{:}\, D)\\ \forall R{:}(C \sqcup D) &\sim (\forall R{:}\, C) \sqcup (\forall R{:}\, D)\end{aligned}$$

all hold in $\mathcal{FALC}$ while the last equivalence does not hold in $\mathcal{ALC}$. The other important difference is that the equivalence

$$\exists R{:}(C \sqcap D) \sim (\exists R{:}\, C) \sqcap (\exists R{:}\, D)$$

holds in $\mathcal{FALC}$ but does not hold in $\mathcal{ALC}$.

3 Constraint Systems

The applicative structure of concept descriptions is rather unsuitable for devising consistency checking algorithms. Fortunately, every concept description can be translated in linear time into a constraint system (that is, a finite set of constraints) such that the concept description is consistent if and only if the constraint system is consistent. For constraint systems we will give simple transformation rules keeping consistency and inconsistency invariant and yielding transparent consistency checking algorithms. This technique also has been used successfully for Feature Logic [Smolka 88].

We assume the existence of two further disjoint alphabets of symbols, called **individual** and **concept variables**, respectively. The letters x, y, z will always range over individual variables and the letters X, Y, Z will always range over concept variables.

Let $\mathcal{I}$ be an interpretation. An **$\mathcal{I}$-assignment** is a function α that maps every individual variable to an element of $D^{\mathcal{I}}$ and every concept variable to a subset of $D^{\mathcal{I}}$. We use $\mathrm{ASS}^{\mathcal{I}}$ to denote the set of all $\mathcal{I}$-assignments.

A **constraint** is a piece of abstract syntax having one of the forms

$$X \sqsubseteq C, \quad X(\forall R)Y, \quad X(\exists R)Y, \quad X \sqsubseteq Y \sqcup Z, \quad x{:}\,X, \quad xRy,$$

where the C in the first form is a simple concept description. Given an interpretation $\mathcal{I}$, we extend the interpretation function $\mathcal{I}[\![\cdot]\!]$ to constraints by interpreting them as sets of $\mathcal{I}$-assignments:

$$\mathcal{I}[\![X \sqsubseteq C]\!] = \{\alpha \in \mathrm{ASS}^{\mathcal{I}} \mid \alpha(X) \subseteq \mathcal{I}[\![C]\!]\}$$

$$\mathcal{I}[\![X(\forall R)Y]\!] = \{\alpha \in \mathrm{ASS}^{\mathcal{I}} \mid \forall a \in \alpha(X)\ \forall(a,b) \in \mathcal{I}[\![R]\!]\colon\ b \in \alpha(Y)\}$$

$$\mathcal{I}[\![X(\exists R)Y]\!] = \{\alpha \in \mathrm{ASS}^{\mathcal{I}} \mid \forall a \in \alpha(X)\ \exists(a,b) \in \mathcal{I}[\![R]\!]\colon\ b \in \alpha(Y)\}$$

$$\mathcal{I}[\![X \sqsubseteq Y \sqcup Z]\!] = \{\alpha \in \mathrm{ASS}^{\mathcal{I}} \mid \alpha(X) \subseteq \alpha(Y) \cup \alpha(Z)\}$$

$$\mathcal{I}[\![x\colon X]\!] = \{\alpha \in \mathrm{ASS}^{\mathcal{I}} \mid \alpha(x) \in \alpha(X)\}$$

$$\mathcal{I}[\![xRy]\!] = \{\alpha \in \mathrm{ASS}^{\mathcal{I}} \mid (\alpha(x), \alpha(y)) \in \mathcal{I}[\![R]\!]\}.$$

A **constraint system** is a finite, nonempty set of constraints. An interpretation $\mathcal{I}$ interprets a constraint system S as follows:

$$\mathcal{I}[\![S]\!] = \bigcap_{c \in S} \mathcal{I}[\![c]\!].$$

A constraint system S is **consistent** if there exists an interpretation $\mathcal{I}$ such that $\mathcal{I}[\![S]\!]$ is nonempty. The next proposition gives a translation of simple $\mathcal{ALC}$-concept descriptions into consistency equivalent constraint systems:

Proposition 3.1. *Let x be an individual variable and X be a concept variable. Then a simple concept description C is consistent if and only if the constraint system $\{x \in X, X \sqsubseteq C\}$ is consistent.*

A constraint system S is **simple** if for every constraint $X \sqsubseteq C$ in S the concept description C is either a concept symbol different from $\top$ or a complemented concept symbol.

The following **unfolding rules** can be used to simplify general constraint systems to simple constraint systems:

$X \sqsubseteq \forall R\colon C \ \rightarrow\ X(\forall R)Y,\ Y \sqsubseteq C,$ where Y is a new variable

$X \sqsubseteq \exists R\colon C \ \rightarrow\ X(\exists R)Y,\ Y \sqsubseteq C,$ where Y is a new variable

$X \sqsubseteq C \sqcap D \ \rightarrow\ X \sqsubseteq C,\ X \sqsubseteq D$

$X \sqsubseteq C \sqcup D \ \rightarrow\ X \sqsubseteq Y \sqcup Z,\ Y \sqsubseteq C,\ Z \sqsubseteq D,$ where Y, Z are new variables

$X \sqsubseteq \top \ \rightarrow$ *nothing*.

Proposition 3.2. *Let a constraint system S' be obtained from a constraint system S by the application of an unfolding rule. Then S is consistent if and only if S' is consistent.*

Proposition 3.3. *For every constraint system S one can compute in linear time a simple constraint system S' such that S is consistent if and only if S' is consistent.*

A simple constraint system defines a directed graph, called its **skeleton**, as follows: every concept variable occurring in the constraint system is taken as a node, the constraints $X(\exists R)Y$ and $X(\forall R)Y$ define existential and universal edges from X to Y, respectively, and a constraint $X \sqsubseteq Y \sqcup Z$ defines an or-connected pair of edges from X to Y and Z, respectively. Furthermore, the constraints $X \sqsubseteq A$ and $X \sqsubseteq \neg A$ define A and $\neg A$, respectively, as labels of the node X. Thus every node has a finite, possibly empty set of labels, where every label is either a concept symbol different from $\top$ or a complemented concept symbol. The individual constraints $x\!:\!X$ and xRy don't contribute to the skeleton.

A **constraint tree** [**constraint forest**] is a simple constraint system whose skeleton is a tree [forest]. A constraint tree T is **fresh** if it can be obtained by unfolding a simple $\mathcal{ALC}$-concept description. Note that a fresh constraint tree has only one constraint containing an individual variable, which has the form $x\!:\!X$, where X is the root of the tree. Now we can formulate the main result of this section:

Theorem 3.4. *For every concept description C on can compute in linear time a fresh constraint tree T such that C is consistent if and only if T is consistent.*

Proof. First C is transformed into a simple concept description using the simplification rules given in the previous section, then the corresponding constraint system is created, which is then simplified to a fresh constraint tree using the unfolding rules. All three steps require at most linear time and preserve consistency and inconsistency. □

4 Consistency Checking as Completion

We now define so-called complete constraint systems whose consistency can be checked in linear time. We will show that every fresh constraint tree can be "completed" to a consistency equivalent complete constraint system by adding individual constraints of the form $x\colon X$ and xRy. This provides a framework in which consistency checking algorithms are obtained as completion algorithms for fresh constraint trees.

A **clash** is a constraint system having either the form $\{x\colon X,\ X \sqsubseteq \neg\top\}$ or the form $\{x\colon X,\ X \sqsubseteq A,\ x\colon Y,\ Y \sqsubseteq \neg A\}$.

Proposition 4.1. *Every constraint system containing a clash is inconsistent. Furthermore, one can check in linear time whether a constraint system contains a clash.*

A **complete** constraint system is a simple constraint system S satisfying the following conditions:

1. if $x\colon X$ and $X(\exists R)Y$ are in S, then there exists a variable y such that $y\colon Y$ and xRy are in S.
2. if $x\colon X$, $X(\forall R)Y$ and xRy are in S, then $y\colon Y$ is in S
3. if $x\colon X$ and $X \sqsubseteq Y \sqcup Z$ are in S, then $x\colon Y$ or $x\colon Z$ is in S.

Proposition 4.2. *A complete constraint system is consistent if and only if it contains no clash.*

Proof. One direction is obvious. To see the other direction, let S be a complete constraint system containing no clash. We define an interpretation $\mathcal{I}$ by taking for $D^{\mathcal{I}}$ all individual variables occurring in S, for $\mathcal{I}[\![A]\!]$ all x such that $x\colon X$ and $X \sqsubseteq A$ are in S for some X, and by taking for $\mathcal{I}[\![R]\!]$ all pairs (x, y) such that xRy is in S. Furthermore, we obtain an assignment $\alpha \in \mathcal{I}[\![S]\!]$ by mapping individual variables to themselves and taking for $\alpha(X)$ all x such that $x\colon X$ is in S. $\square$

Our basic completion algorithm relies on three completion rules whose logical properties are stated in the following proposition:

1. $S \rightarrow_{\exists} \{y\colon Y, xRy\} \cup S$

   if $x\colon X$ and $X(\exists R)Y$ are in S,

   there exists no variable z such that xRz and $z\colon Y$ are in S, and

   y is a variable not occurring in S

2. $S \rightarrow_{\forall} \{y\colon Y\} \cup S$

   if $x\colon X$, xRy and $X(\forall R)Y$ are in S and $y\colon Y$ is not in S

3. $S \rightarrow_{\sqcup} \{x\colon Z\} \cup S$

   if $x\colon X$ and $X \sqsubseteq Y_1 \sqcup Y_2$ are in S,

   neither $x\colon Y_1$ nor $x\colon Y_2$ is in S, and Z is either Y_1 or Y_2

Figure 4.1. The basic completion rules for constraint trees.

Proposition 4.3. *Let S be a constraint system. Then:*

1. *if $x\colon X$ and $X(\exists R)Y$ are in S and y is an individual variable not occurring in S, then S is consistent if and only if $S \cup \{xRy, y\colon Y\}$ is consistent*

2. *if $x\colon X$, xRy and $X(\forall R)Y$ are in S, then S is consistent if and only if $S \cup \{y\colon Y\}$ is consistent*

3. *if $x\colon X$ and $X \sqsubseteq Y \sqcup Z$ are in S, then S is consistent if and only if $S \cup \{x\colon Y\}$ or $S \cup \{x\colon Z\}$ is consistent.*

To obtain an algorithm, we need to impose some control that ensures that after finitely many completion steps no further completion step is applicable. This leads to the completion rules given in Figure 4.1.

Proposition 4.4. *The basic completion rules in Figure 4.1 have the following properties:*

1. *there is no infinite chain of completion steps issuing from a fresh constraint tree*

2. *a simple constraint system is complete if and only if none of the completion rules applies to it*

3. *if T' is obtained from a constraint tree T by one of the completion rules, then T' is a constraint tree and T is consistent if T' is consistent.*

A **completion** of a constraint tree T is a complete constraint tree that can be obtained from T by finitely many applications of the basic completion rules in Figure 4.1.

Theorem 4.5. *Every fresh constraint tree has a completion. Furthermore, a fresh constraint tree is consistent if and only if it has a clash-free completion.*

Proof. Follows from the preceding propositions. □

Thus we have an algorithm for deciding consistency and subsumption of $\mathcal{ALC}$-concept descriptions. The example in Figure 4.2 shows that the completions of a fresh constraint tree can all be exponentially larger than the initial tree. However, there is no need to keep the entire completion in memory. In the next section we will give a smarter control for the completion rules yielding a consistency checking algorithm that requires only linear space.

Theorem 4.6. *Every $\mathcal{FL}$-concept description is consistent.*

Proof. Let F be an $\mathcal{FL}$-concept description and T be a simple constraint tree obtained by unfolding F's translation into an $\mathcal{ALC}$-concept description. Although F contains neither unions nor complements, T does since they are introduced by the translation rule

$$(ALL\ (RESTRICT\ R\ C)\ D) \rightarrow \forall R{:}(\neg C \sqcup D).$$

However, if we complete T such that individual variables are always propagated to the right-hand sides of unions, we obtain a clash-free completion since then a pair $x{:}X$ and $X{:}\neg A$ cannot occur. □

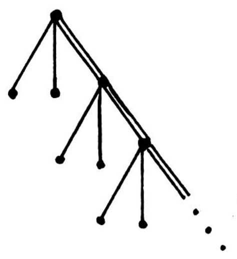


Figure 4.2. A family of skeletons for which the number of individual variables in every completion is exponential in the size of the skeleton. These skeletons can be obtained from the concept descriptions $(\exists R{:}\top)\sqcap(\exists R{:}\top)\sqcap(\forall R{:}\ \cdots)$. A "double line edge" represents a universal edge and a "singe line edge" represents an existential edge. Note that only one role symbol is used.

5 Upper Complexity Bounds

In this section we will prove upper complexity bounds for the ACD-languages $\mathcal{AL}$, $\mathcal{ALE}$, $\mathcal{ALU}$ and $\mathcal{ALC}$. In particular, we will show that the consistency of $\mathcal{ALC}$-concept descriptions can be decided with linear space. The basic idea behind our linear-space algorithm is that a completion can be sliced up into so-called traces such that the completion contains a clash if and only if one of its traces contains a clash. While the size of completions can be exponential, the size of traces is linear in the size of the initial concept description. The algorithm systematically enumerates traces until it has found a clash-free completion.

A **partial completion** of a fresh constraint tree T is a constraint tree that can be obtained from T by finitely many applications of the completion rules; in particular, a fresh constraint tree is a partial completion of itself. Given a constraint tree T and individual variables x and y occurring in T, y is called a **successor** of x and x is called a **predecessor** of y if T contains a constraint xRy. An individual variable x occurring in a constraint tree T is called an **individual root** if T contains a constraint $x{:}X$ such that X is the root of the skeleton of T.

Proposition 5.1. *Let T be a partial completion of a fresh constraint tree. Then T has a unique individual root, the individual root has no predecessor, and every other individual variable occurring in T has a unique predecessor.*

A partial completion U of a fresh constraint tree T is called a **trace** of T if

1. every individual variable occurring in U has at most one successor
2. the completion rules $\rightarrow_{\forall}$ and $\rightarrow_{\sqcup}$ don't apply to U
3. every application of the completion rule $\rightarrow_{\exists}$ to U yields a constraint tree containing an individual variable having two successors.

Traces can be computed using the following restriction of the existential completion rule $\rightarrow_{\exists}$:

$S \rightarrow_{T\exists} \{y\colon Y, xRy\} \cup S$
  if $x\colon X$ and $X(\exists R)Y$ are in S,
    there is no constraint $xR'y'$ in S, and
    y is a variable not occurring in S.

We define the binary relation $\rightarrow_T$ on simple constraint systems by

$$\rightarrow_T \;=\; \rightarrow_{T\exists} \cup \rightarrow_{\forall} \cup \rightarrow_{\sqcup} ,$$

where $\rightarrow_{T\exists}$, $\rightarrow_{\forall}$ and $\rightarrow_{\sqcup}$ are the relations on simple constraint systems given by the corresponding completion rules. Note that the traces of a fresh constraint tree T are the $\rightarrow_T$ -normal forms of T (that is, constraint trees U such that $T \rightarrow_T^* U$ and $U \rightarrow_T V$ for no constraint tree V).

Proposition 5.2. *Let T be a fresh constraint tree. Then:*

1. *if $T \rightarrow_T^* T'$ and T' contains the constraints $x\colon X$ and $y\colon X$, then $x = y$ (that is, no concept variable in T' has more than one individual variable associated with it)*
2. *the length of $\rightarrow_T$ -derivations issuing from T is bound linearly in the size T*
3. *every trace of T is contained in a completion of T*

```
eval: nat x constraint tree → bool

eval($x, S$) =
    if $\{x{:}X,\ X \sqsubseteq A,\ x{:}Y,\ Y \sqsubseteq \neg A\} \subseteq S \ \vee\ \{x{:}X,\ X \sqsubseteq \neg\top\} \subseteq S$
    then false
    elsif $\{x{:}X,\ X \sqsubseteq Y \sqcup Z\} \subseteq S \ \wedge\ x{:}Y \notin S \ \wedge\ x{:}Z \notin S$
    then eval($x$, $\{x{:}Y\} \cup S$) $\vee$ eval($x$, $\{x{:}Z\} \cup S$)
    else let $y = x + 1$ in
        $\forall\ \{x{:}X,\ X(\exists R)Y\} \subseteq S$:
            eval($y$, $\{y{:}Y\} \cup \{y{:}Z' \mid \exists\ \{x{:}Z,\ Z(\forall R)Z'\} \subseteq S\} \cup S$)
```

Figure 5.1. A functional linear-space consistency checking algorithm for fresh constraint trees. If T is a fresh constraint tree and x is the individual root of T, then *eval*(x, T) = *true* if and only if T has a clash-free completion, that is, is consistent. For convenience, individual variables are assumed to be natural numbers.

4. *every completion $\tilde{T}$ of T can be obtained as the union of the finitely many traces contained in $\tilde{T}$.*

The recursive function *eval* in Figure 5.1 yields *true* if the fresh constraint tree given as argument has a clash-free completion and *false* otherwise. The maximal recursion depth is the height of the given tree. The function *eval* can be executed such that, besides some control information, at most a trace of the given tree needs to be kept in memory. Hence *eval* needs at most space linear in the size of the input tree. The total correctness of *eval* follows from the propositions we have stated so far. Thus we have:

Theorem 5.3. *Checking consistency and subsumption of $\mathcal{ALC}$-concept descriptions can be done with linear space.*

Proposition 5.4. *Let T be a fresh constraint tree not containing union con-*

straints. Then:

1. *all completions of T are equal up to consistent renaming of individual variables*
2. *T is consistent $\iff$ T has a clash-free completion*
   $\iff$ every completion of T is clash-free
   $\iff$ no trace of T contains a clash.

Theorem 5.5. *Inconsistency of $\mathcal{ALE}$-concept descriptions can be decided in nondeterministic linear time. Thus the consistency problem for $\mathcal{ALE}$-concept descriptions is in co-NP.*

Proof. We have to show that inconsistency of fresh constraint trees containing no union constraints can be decided in nondeterministic linear time. Since such constraint trees are inconsistent if and only if they have a trace containing a clash, this follows from the fact that every trace can be obtained by a $\rightarrow_{T}$-derivation whose length is bound linearly by the constraint tree it is issuing from. □

To establish our upper complexity bounds for $\mathcal{ALU}$ and $\mathcal{AL}$ we need yet another restriction of the completion rule $\rightarrow_{\exists}$:

$S \rightarrow_{\mathsf{T}\exists} \{y\colon Y, xRy\} \cup S$
  if $x\colon X$ and $X(\exists R)Y$ are in S,
    there is no constraint xRy' in S, and
    y is a variable not occurring in S.

Note that $\rightarrow_{T\exists} \subseteq \rightarrow_{\mathsf{T}\exists}$ since $\rightarrow_{T\exists}$ can be applied to at most one existential edge at every level while $\rightarrow_{\mathsf{T}\exists}$ can be applied to several existential edges if they are labeled with different relation symbols. We define the binary relation $\rightarrow_{\mathsf{T}}$ on simple constraint systems by

$$\rightarrow_{\mathsf{T}} \;=\; \rightarrow_{\mathsf{T}\exists} \cup \rightarrow_{\forall} \cup \rightarrow_{\sqcup} .$$

Note that $\rightarrow_{T} \subseteq \rightarrow_{\mathsf{T}}$ since $\rightarrow_{T\exists} \subseteq \rightarrow_{\mathsf{T}\exists}$.

Proposition 5.6. *The length of $\rightarrow_{\mathsf{T}}$-derivations issuing from fresh constraint trees is bound linearly in the size of the initial tree.*

Proof. Let T be a fresh constraint tree and U be a constraint tree such that $T \rightarrow_{\top}^{*} U$. Then for every concept variable X in U there exists at most one individual variable x such that the constraint $x{:}X$ is in U. From this invariant the claim follows easily. □

The $\top$-**completions** of a fresh constraint tree T are the $\rightarrow_{\top}$-normal forms of T.

Proposition 5.7. *Let T be a $\top$-completion of a fresh constraint tree obtainable by unfolding an $\mathcal{ALU}$-concept description. If T contains the constraints $x{:}X$, $X(\exists R)Y$ and xRy, then T is consistent if and only if $T \cup \{y{:}Y\}$ is consistent.*

Proof. Follows from the fact that every concept variable reachable through an existential edge is an unlabeled leaf. □

Proposition 5.8. *Let T be a fresh constraint tree obtainable by unfolding an $\mathcal{ALU}$-concept description. Then T is consistent if and only if T has a clash-free $\top$-completion.*

Proof. If T is consistent, then every $\top$-completion of T is clash-free. To show the other direction, suppose T has a clash-free $\top$-completion U. Using the preceding proposition, we can extend U to a complete constraint tree V since concept variables reachable through existential edges are always leaves. Since U is clash-free and concept variables reachable through existential edges are not labeled, V is clash-free. Hence V is consistent. Since $T \subseteq U \subseteq V$, we thus know that T is consistent. □

Theorem 5.9. *Checking the consistency of $\mathcal{ALU}$-concept descriptions is an NP-complete problem.*

Proof. The NP-hardness was already stated in Proposition 2.3. That consistency checking is in NP follows from the preceding theorem since unfolding can be done in linear time, every $\top$-completion can be computed in nondeterministic linear time, and clash-freeness can be checked in linear time. □

Theorem 5.10. *The consistency of $\mathcal{AL}$-concept descriptions can be checked in linear time.*

Proof. Let T be a fresh constraint tree obtainable by unfolding an $\mathcal{AL}$-concept description. Then all $\top$-completions of T are equal up to consistent renaming of individual variables. Thus it suffices to compute any $\top$-completion and to check it for clashes. □

6 PSPACE-Completeness

We now show that deciding consistency and subsumption of $\mathcal{ALC}$-concept descriptions are problems that are as hard as any problem that can be decided with polynomial space. Since we have proved in the last section that consistency and subsumption of $\mathcal{ALC}$-concept descriptions can be decided with linear space, we will be able to conclude that these problems are PSPACE-complete. We will prove the PSPACE-hardness of the consistency problem for $\mathcal{ALC}$-concept descriptions by reducing the validity problem for quantified boolean formulas to it.

6.1 Quantified Boolean Formulas

We now review quantified boolean formulas whose validity problem (called QBF, for short) is known to be PSPACE-complete (see, for instance, [Garey/Johnson 79]). We use a notation providing for a smooth reduction of QBF to the consistency problem for $\mathcal{ALC}$.

A **literal** is a nonzero integer. A **clause** is a nonempty sequence $l_1 \cdots l_n$ of literals such that $|l_1| \leq |l_2| \leq \cdots \leq |l_n|$. A **prefix** from m to n, where m and n are positive integers such that $m \leq n$, 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 (M is called the **matrix** of the formula).

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

$$\{m, m+1, \ldots, n\} \rightarrow \{0,1\}.$$

An assignment α **satisfies** a literal l if $\alpha(l) = 1$ if l is positive and $\alpha(-l) = 0$ 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 A of P-assignments is **canonical** for P if it satisfies the following conditions:

1. A is nonempty
2. if $P = (\exists m)P'$, then all assignments of A agree on m and, if P' is nonempty, $\{\alpha|_{\{m+1,\ldots,n\}} \mid \alpha \in \mathrm{A}\}$ is canonical for P'
3. if $P = (\forall m)P'$, then

   3.1 A contains an assignment that satisfies m and, if P' is nonempty, $\{\alpha|_{\{m+1,\ldots,n\}} \mid \alpha \in \mathrm{A} \wedge \alpha(m) = 1\}$ is canonical for P'

   3.2 A contains an assignment that satisfies $-m$ and, if P' is nonempty, $\{\alpha|_{\{m+1,\ldots,n\}} \mid \alpha \in \mathrm{A} \wedge \alpha(m) = 0\}$ is canonical for P'.

A quantified boolean formula $P.M$ is **valid** if there exists a set A of P-assignments canonical for P such that every assignment in A satisfies every clause of M. An example of a valid quantified boolean formula written in a readable syntax is $\forall x \exists y.(\neg x \vee y) \wedge (x \vee \neg y)$. The following theorem is taken from [Garey/Johnson 79]:

Theorem 6.1. *Deciding the validity of quantified boolean formulas is a PSPACE-complete problem.*

6.2 The Reduction

In the following we assume R to be a fixed role symbol and A to be a fixed concept symbol. Quantified boolean formulas are translated into $\mathcal{ALC}$-concept descriptions using the equation

$$[P.\{C_1, \ldots, C_n\}] = [P] \sqcap [C_1]^0 \sqcap \cdots \sqcap [C_n]^0,$$

where prefixes are translated using the equations

$$\begin{aligned}
[(\exists m)P] &= ((\exists R\colon A) \sqcup (\exists R\colon \neg A)) \sqcap (\forall R\colon [P]) \\
[(\forall m)P] &= (\exists R\colon A) \sqcap (\exists R\colon \neg A) \sqcap (\forall R\colon [P]) \\
[(\exists m)] &= (\exists R\colon A) \sqcup (\exists R\colon \neg A) \\
[(\forall m)] &= (\exists R\colon A) \sqcap (\exists R\colon \neg A)
\end{aligned}$$

and clauses are translated using the equations

$$\begin{aligned}
[lC]^m &= \forall R\colon [lC]^{m+1} \quad \text{if } |l| > m \\
[mC]^m &= A \sqcup [C]^m \\
[-mC]^m &= \neg A \sqcup [C]^m \\
[l]^m &= \forall R\colon [l]^{m+1} \quad \text{if } |l| > m \\
[m]^m &= A \\
[-m]^m &= \neg A.
\end{aligned}$$

The number argument m of the translation function $[C]^m$ for clauses is needed to ensure that only unions of the form $A \sqcup C$ or $\neg A \sqcup C$ are introduced, which is essential. Figure 6.1 gives an example of a translation.

To show that a quantified boolean formula is valid if and only if its translation into an $\mathcal{ALC}$-concept description is consistent, we assign levels to the variables occurring in constraint trees as follows:

1. the concept variable that is the root of the constraint tree has the level 0
2. if the constraint tree contains a constraint $X(\exists R)Y$ or $X(\forall R)Y$ and X has the level n, then Y has the level $n+1$
3. if the constraint tree contains a constraint $X \sqsubseteq Y \sqcup Z$, then X, Y and Z all have the same level
4. if the constraint tree contains a constraint $x\colon X$, then x and X have the same level

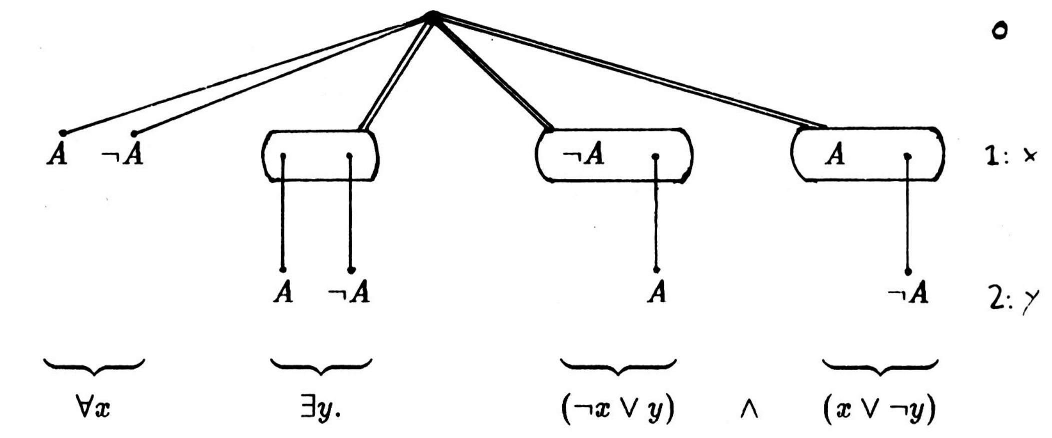


Figure 6.1. A valid quantified boolean formula and its translation into a constraint tree.

5. if the constraint tree contains a constraint xRy and x has the level n, then y has the level $n + 1$.

This defines a unique level assignment for constraint trees that are obtained from fresh constraint trees by finitely many applications of the completion rules.

Lemma 6.2. *A quantified boolean formula $P.M$ is valid if and only if its translation $[P.M]$ is a consistent $\mathcal{ALC}$-concept description.*

Proof. Let $P.\{C_1, \ldots, C_n\}$ be a quantified boolean formula such that P is a prefix from 1 to m. Furthermore, let $\bar{P} \cup \bar{C}_1 \cup \cdots \cup \bar{C}_n$ be a fresh constraint tree obtainable by unfolding the translation $[P.\{C_1, \ldots, C_n\}]$, where $\bar{P}$ is a fresh constraint tree obtainable by unfolding $[P]$ and, for $i \in 1..n$, $\bar{C}_i$ is a fresh constraint tree obtainable by unfolding $[C_i]$.

Let $\hat{P}$ be a trace of $\bar{P}$. Then $\hat{P}$ contains exactly one individual variable x_i for every $i \in 0..m$. Furthermore, $\hat{P}$ contains the chain $x_0 R x_1, \ldots, x_{m-1} R x_m$. We say that i is *true* in $\hat{P}$ if $\hat{P}$ contains two constraints $x_i\colon X$ and $X \sqsubseteq A$, and that i is *false* in $\hat{P}$ if $\hat{P}$ contains two constraints $x_i\colon X$ and $X \sqsubseteq \neg A$. Every $i \in 1..m$ is either true or false in $\hat{P}$, but not both. Thus $\hat{P}$ defines a P-assignment α as follows: $\alpha(i) = 1$ if i is true in $\hat{P}$ and $\alpha(i) = 0$ if i is false in $\hat{P}$.

Since no trace of $\bar{P}$ contains a clash, every completion of $\bar{P}$ is clash-free. Furthermore, the set of the P-assignments defined by the traces contained in a completion of $\bar{P}$ is canonical for P. Vice versa, every set of P-assignments that is canonical for P can be obtained from the same completion of $\bar{P}$. This correspondence between canonical sets of assignments and completions is crucial for our proof.

Let C_i be one of the clauses. Then $\bar{C}_i$ contains no existential edges. The leaves of $\bar{C}_i$ correspond exactly to the literals of C_i. If l is a positive literal in C, then the corresponding leaf X of $\bar{C}_i$ has the level l and $\bar{C}_i$ contains the constraint $X \sqsubseteq A$. If l is a negative literal in C, then the corresponding leaf X of $\bar{C}_i$ has the level $-l$ and $\bar{C}_i$ contains the constraint $X \sqsubseteq \neg A$.

1. "[P.M] consistent $\Rightarrow$ P.M valid". Suppose $[P.\{C_1, \ldots, C_n\}]$ is a consistent $\mathcal{ALC}$-concept description. Then $\bar{P} \cup \bar{C}_1 \cup \cdots \cup \bar{C}_n$ has a clash-free completion $\tilde{P} \cup \tilde{C}_1 \cup \cdots \cup \tilde{C}_n$ such that $\tilde{P}$ is a completion of $\bar{P}$.

Let $\hat{P}$ be a trace of $\bar{P}$ such that $\hat{P} \subseteq \tilde{P}$ and let α be the P-assignment defined by $\hat{P}$. Furthermore, let C_i be one of the clauses. It suffices to show that C_i contains a literal that is satisfied by α.

Let $\hat{P} \cup \hat{C}_i$ be the clash-free trace of $\bar{P} \cup \tilde{C}_i$ such that $\hat{C}_i \subseteq \tilde{C}_i$. Then $\hat{C}_i$ contains exactly one constraint $x{:}X$ such that X is a leaf. Since $\hat{C}_i$ is clash-free, α satisfies the literal in C_i that corresponds to X.

2. "P.M valid $\Rightarrow$ [P.M] consistent". Suppose $P.\{C_1, \ldots, C_n\}$ is valid. Then there exists a set A of P-assignments that is canonical for P such that every $\alpha \in$ A satisfies every clause C_i. Let $\tilde{P}$ be a completion of $\bar{P}$ that yields A. It suffices to show that there exists a clash-free completion $\tilde{P} \cup \tilde{C}_i$ of $\bar{P} \cup \bar{C}_i$ for every clause C_i since the union of these completions is a clash-free completion of $\bar{P} \cup \bar{C}_1 \cup \cdots \cup \bar{C}_n$ (because no $\bar{C}_i$ contains an existential edge and every individual variable having a level between 1 and m is either true or false in $\tilde{P}$).

Let C_i be one of the clauses and let $\rightarrow_Q$ be the following restriction of the completion rule $\rightarrow_\sqcup$:
By applying the completion rules $\rightarrow_\forall$ and $\rightarrow_Q$ to $\tilde{P} \cup \bar{C}_i$ we obtain a completion $\tilde{P} \cup \tilde{C}_i$ of $\bar{P} \cup \bar{C}_i$. It remains to show that $\tilde{P} \cup \tilde{C}_i$ is clash-free.

if $X \sqsubseteq Y_1 \sqcup Y_2$, $x: X$, $x: Z$, $Z \sqsubseteq C$ and $Y_1 \sqsubseteq D$ are in S, then add $x: Y_1$ if $C{=}D$ and $x: Y_2$ if $C \neq D$.

Suppose $\tilde{P} \cup \tilde{C}_i$ contains a clash. Let $x_0 R x_1, \ldots, x_{k-1} R x_k$ be constraints in $\tilde{P}$ such that x_0 has the level 0 and x_k is involved in a clash in $\tilde{P} \cup \tilde{C}_i$. Because $\tilde{P} \cup \tilde{C}_i$ was obtained using the completion rule $\rightarrow_Q$, the greatest level in $\tilde{C}_i$ must be k. Let $\hat{P} \subseteq \tilde{P}$ be a trace of $\tilde{P}$ containing the constraints $x_0 R x_1, \ldots, x_{k-1} R x_k$ and let $\alpha \in \mathrm{A}$ be the P-assignment defined by $\hat{P}$. Now it is easy to verify that α satisfies no literal in C_i. This contradicts our assumption that every assignment of A satisfies every clause. □

Theorem 6.3. *Deciding consistency and subsumption of $\mathcal{ALC}$-concept descriptions are PSPACE-complete problems.*

Proof. In Section 2 we have shown that the subsumption and consistency problems can be reduced to each other in linear time. In Section 5 we have shown that consistency can be decided with linear space. Since QBF is PSPACE-complete and the given reduction to the consistency problem is quadratic-time, we have the claim by the preceding lemma stating the correctness of the reduction. □

7 Conclusions

This paper is a further step in an effort aimed at the integration of feature descriptions used in computational linguistics with concept descriptions employed in KL-ONE-like knowledge representation languages. The first step in this effort was the development of Feature Logic [Smolka 88], which integrates existing feature description formalisms and provides a model-theoretic semantics revealing the close relationship with KL-ONE concept descriptions. Technically, the difference between feature descriptions and KL-ONE is that in feature descriptions roles must be partial functions (called features) while in KL-ONE roles can be arbitrary binary relations.

The minor semantical difference between features and general roles causes major computational differences. If role inclusion or agreement is used with roles, it causes undecidability [Schmidt-Schauß 88], while its use in feature descriptions neither destroys decidability nor causes a complexity jump. If role inclusion is not present, the consistency problem for descriptions using general roles is PSPACE-complete [this paper], while the corresponding problem for feature descriptions is NP-complete [Smolka 88].

Not surprisingly, feature descriptions have been developed for applications where feature agreement is essential, while KL-ONE applications exploit general roles and avoid role inclusions. Nevertheless, there are important applications in computational linguistics (for instance, coordination) that could be given elegant solutions using general roles (in this context known as set-valued attributes). The undecidability result of Schmidt-Schauß [88] and the results of this paper indicate that the integration of set-valued attributes is computationally nontrivial and requires great care if decidability is to be preserved.

More work will be necessary to arrive at a computationally useful integration of Feature Logic and KL-ONE. One difficulty is that the consistency checking algorithms for feature descriptions are very different from the algorithms for $\mathcal{ALC}$ developed in this paper. In addition, the consistency checking algorithms for $\mathcal{ALC}$ are not incremental, that is, they don't produce a simplified version of the checked description. In contrast, the consistency checking algorithms for feature descriptions are incremental and are thus much better suited for practical applications. In fact, what is called unification in the context of feature descriptions is just incremental consistency checking.

References

H. Aït-Kaci, An Algebraic Semantics Approach to the Effective Resolution of Type Equations. Theoretical Computer Science 45, 1986, 293–351.

R.J. Brachman, V.P. Gilbert, and H.J. Levesque, An Essential Hybrid Reasoning System: Knowledge and Symbol Level Accounts in KRYPTON. Proc. 9th IJCAI, Los Angeles, Cal., 1985, 532–539.

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

M.R. Garey and D.S. Johnson, Computers and Intractability—A Guide to the Theory of NP-Completeness. Freeman, 1979.

M. Höhfeld and G. Smolka, Definite Relations over Constraint Languages. LILOG Report 53, IBM Deutschland, West Germany, October 1988.

M.E. Johnson, Attribute-Value Logic and the Theory of Grammar. PhD Dissertation, Stanford University, 1987. To appear as CSLI Lecture Notes.

R. Kaplan and J. Bresnan, Lexical-Functional Grammar, a Formal System for Grammatical Representation. In J. Bresnan (Ed.), The Mental Representation of Grammatical Relations, The MIT Press, 1982, 173–381.

H.J. Levesque and R.J. Brachman, Expressiveness and Tractability in Knowledge Representation and Reasoning. Computational Intelligence 3, 1987, 78–93.

B. Nebel, Computational Complexity of Terminological Reasoning in BACK. Artificial Intelligence 34, 1988, 371–383.

B. Nebel and K. von Luck, Hybrid Reasoning in BACK. In Z. W. Ras and L. Saitta (eds.), Methodologies for Intelligent Systems 3, North-Holland, 1988, 260–269.

P.F. Patel-Schneider, Small can be Beautiful in Knowledge Representation. Proc. IEEE Workshop on Principles of Knowledge-Based Systems, Denver, Colorado, 1984, 11–16.

W.C. Rounds and R.T. Kasper, A Complete Logical Calculus for Record Structures Representing Linguistic Information. Proc. of the First IEEE Symposium on Logic in Computer Science, Boston, 1986, 38–43.

Manfred Schmidt-Schauß, Subsumption in KL-ONE is Undecidable. SEKI Report SR-88-14, FB Informatik, Universität Kaiserslautern, West Germany.

G. Smolka, A Feature Logic with Subsorts. LILOG Report 33, IBM Deutschland, West Germany, May 1988. To appear in the proceedings of the Workshop on Unification Formalisms—Syntax, Semantics and Implementation, Titisee, The MIT Press.

M.B. Vilain, The Restricted Language Architecture of a Hybrid Representation System. Proc. of the 9th IJCAI, Los Angeles, Cal., August 1985, 547–551.