Tractable Concept Languages

Francesco M. Donini
Maurizio Lenzerini
Daniele Nardi
Dipartimento di Informatica e Sistemistica,
Universita di Roma "La Sapienza"
Via Salaria 113, 1-00198, Roma, Italy

Werner Nutt
Deutsches Forschungszentrum fur
Kiinstliche Intelligenz
Postfach 2060, D-6750 Kaiseislautern
Germany

Abstract

We present two concept languages,, called PL_1 and PL_2 which are extensions of *TC*$^-$. We prove that the subsumption problem in these languages can be solved in polynomial time. Both languages include a construct for expressing inverse roles, which has not been considered up to now in tractable languages. In addition, PL_1 includes number restrictions and negation of primitive concepts, while Pl_2 includes role conjunction and role chaining.

By exploiting recent complexity results, we show that none of the constructs usually considered in concept languages can be added to PL_1 and PL_2 without losing tractabtlity. Therefore, on the assumption that Languages are characterized by the set of constructs they provide, the two languages presented in this paper provide a solution to the problem of singling out an optimal trade-off between expressive power and computational complexity.

1 Introduction

Concept languages provide a means for expressing knowledge about hierarchies of concepts, i.e. classes of objects with common properties. They have been investigated following the ideas initially embedded in many frame-based and semantic-network-based languages, especially the KL-ONE language [Brachman and Schmolze, 1985]. In contrast to earlier formalisms, concept Languages are given a Tarski style declarative semantics that allows them to be conceived as sublanguages of predicate logic [Brachman and Levesque, 1984].

The basic reasoning tasks on concepts are unsatisfiability and subsumption checking. A concept is unsatisfiable if it always denotes an empty set. A concept *C* is subsumed by a concept *D* if C always denotes a subset of *D.*

Since the performance of any application developed using concept languages will heavily rely on the above reasoning tasks, it is important both to characterize their computational complexity and to devise algorithms as much efficient as possible-

Recent results allow us to draw a fairly complete picture of the complexity of a wide class of concept languages [Schmidt-Schaufi and Smolka, 1988,Donini et al., 1990]. Such results have been obtained by exploiting a general technique for satisfiability checking in concept languages. The technique relies on a form of tableaux calculus, and has been proved extremely useful for studying both the correctness and the complexity of the algorithms.

The outcomes of this body of research go far beyond a mere complexity analysis. In particular, we think that they shed light on three basic aspects related to the use of concept languages in knowledge representation.

First of all, since the complexity of both satisfiability and subsumption depends upon the constructs allowed in the language, we have now an appropriate framework for the study of the trade-off between the expressive power of the languages and their inherent complexity, which was the initial motivation of the seminal work by Brachman and Levesque (see [Levesque and Brachman, 1987]).

Secondly, the design of concept languages and the associated reasoning procedures can now be realized through the application of the above mentioned technique, which provides an algorithmic framework that is parametric with respect to the language constructs.

Thirdly, the study of the computational behaviour of concept languages has led to a clear understanding of the properties of the language constructs and their interaction. The knowledge about the structure of concept languages can thus be used in the design of intelligent reasoning procedures, that—by looking at the form of concepts—can reason about the deductive service, for example estimating the difficulty of performing the required deduction, attempting to provide quick answers to subproblems, or trying possible simplification of the problem.

The work reported in this paper is concerned with the first of the above three points. Our goal was the design of languages including the most powerful set of constructs, while retaining the tractability of subsumption, in particular extending the basic polynomial language FL^- [Brachman and Levesque, 1984]. Fl^- includes conjunction of concepts (written CI D), universal role quantification (VJ?.C), and unqualified existential role quantification (3#). Various extensions of Fl^- with a polynomial subsumption problem have already been considered:

This work was partly funded by the ESPRIT BRA 3012 (Compnlog), the Italian CNR under Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo, contract 90.00681.PF69, and the German BMFT under grant ITW 8903 0.

- $\mathcal{FL}^-$ + role concatenation (also called role chaining) [Brachman and Levesque, 1984];
- $\mathcal{FL}^-$ + number restrictions [Nebel, 1988];
- $\mathcal{FL}^-$ + role conjunction [Nebel, 1988];
- $\mathcal{FL}^-$ + negation of primitive concepts [Schmidt-Schauß and Smolka, 1988].

The result of our work is the definition of two new extensions of $\mathcal{FL}^-$, called $\mathcal{PL}_1$ and $\mathcal{PL}_2$. We show that subsumption in both languages can be solved in polynomial time. Moreover, they are maximally expressive, in the sense that no construct can be added to them without losing tractability.

In particular, $\mathcal{PL}_1$ extends $\mathcal{FL}^-$ with number restrictions, negation of primitive concepts, and inverse roles, and is therefore maximally expressive relative to the costructs available for concepts. On the other hand, $\mathcal{PL}_2$ extends $\mathcal{FL}^-$ with role conjunction, role concatenation, and inverse roles, and is therefore maximally expressive relative to the costructs available for roles. Notice that the construct for inverse roles has not been considered up to now in tractable languages.

The paper is organized as follows. In Section 2 we summarize the main notions used in the formalization of concept languages. In Section 3 we describe the technique used for subsumption checking. In Sections 4 and 5 we present the languages $\mathcal{PL}_1$ and $\mathcal{PL}_2$, together with the corresponding subsumption algorithms. Finally, conclusions are drawn in Section 6. For the sake of brevity, the proofs of the theorems are omitted; they are fully reported in [Donini et al., 1991b].

2 Basic Notions on Concept Languages

In this section we provide the essential notions about the concept languages considered in the paper. For a general presentation, see [Nebel and Smolka, 1990].

We start by considering the language $\mathcal{FL}^-$, where *concepts* (denoted by the letters C and D) are built out of *primitive concepts* (denoted by the letter A) and *primitive roles* (denoted here by the letter R) according to the syntax rule

$$C, D \longrightarrow A \mid C \sqcap D \mid \forall R.C \mid \exists R$$

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

$$\begin{aligned}(C \sqcap D)^{\mathcal{I}} &= C^{\mathcal{I}} \cap D^{\mathcal{I}}\\ (\forall R.C)^{\mathcal{I}} &= \{a \in \Delta^{\mathcal{I}} \mid \forall b.(a,b) \in R^{\mathcal{I}} \rightarrow b \in C^{\mathcal{I}}\}\\ (\exists R)^{\mathcal{I}} &= \{a \in \Delta^{\mathcal{I}} \mid \exists b.(a,b) \in R^{\mathcal{I}}\}.\end{aligned}$$

An interpretation $\mathcal{I}$ is a *model* for a concept C if $C^{\mathcal{I}}$ is nonempty; C is *subsumed* by D if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for every interpretation $\mathcal{I}$. A concept is *satisfiable* if it has a model and *unsatisfiable* otherwise. Notice that since $\mathcal{FL}^-$ does not include any form of negation, every $\mathcal{FL}^-$-concept is satisfiable.

Other constructs have been considered in the literature to define more general languages. They are:

- the empty concept and the universal concept, defined by $\bot^{\mathcal{I}} = \emptyset$ and $\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$, respectively;
- disjunction of concepts: $(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$;
- negation of concepts (also called complement): $(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$; sometimes negation can only be applied to primitive concepts (in this case we use the notation $\neg A$);
- qualified existential role quantification: $(\exists R.C)^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} \mid \exists (a,b) \in R^{\mathcal{I}}.\, b \in C^{\mathcal{I}}\}$;
- number restrictions: $(\geq n\, R)^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} \mid |\{b \in \Delta^{\mathcal{I}} \mid (a,b) \in R^{\mathcal{I}}\}| \geq n\}$, and $(\leq n\, R)^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} \mid |\{b \in \Delta^{\mathcal{I}} \mid (a,b) \in R^{\mathcal{I}}\}| \leq n\}$.
- role conjunction: $(Q \sqcap R)^{\mathcal{I}} = Q^{\mathcal{I}} \cap R^{\mathcal{I}}$;
- inverse of roles: $(R^{-1})^{\mathcal{I}} = \{(a,b) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid (b,a) \in R^{\mathcal{I}}\}$;
- role chaining: $(R_1 \circ R_2)^{\mathcal{I}} = \{(a,b) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid \exists c.(a,c) \in R_1^{\mathcal{I}}, (c,b) \in R_2^{\mathcal{I}}\}$.

In the following, we consider concept languages obtained as combinations of the above constructs, Several such combinations have been taken into account in the design of concept languages. In general, the constructs for role formation have been introduced only in the most sophisticated and powerful languages. In particular, the construct for the description of inverse roles has been considered very rarely, and never in tractable languages. This is somewhat surprising, especially if one consider its importance in the description of complex concepts.

For example, let child be a primitive role, and let young be a primitive concept. It is easy to see that the concept "someone that has at least one child and all of whose parents are young" cannot be defined in a language without inverse roles. Indeed, the introduction of a new primitive role parent would not solve the problem, because child and parent would be completely unrelated. On the other hand, the notion of parent is obviously captured by the inverse of the role child. Therefore, in a language with inverse roles, the above concept can be defined as: child child^{-1},young.

One of the goals of this paper is to study the impact of inverse roles in the tractability of concept languages. In fact, we show in Sections 4 and 5 that the construct for inverse roles can be added to *TC~* without increasing the complexity of subsumption.

3 Checking Subsumption Using Constraint Systems

In this section we present the basic features of a technique for checking concept satisfiability. The technique is a refinement of the one used in [Schmidt-SchauB and Smolka, 1988], and is fully described in [Donini et al., 1990].

We assume that there exists an alphabet of variable symbols, which will be denoted by the letters x, *y*, and z. The calculus operates on constraints consisting of variables, concepts, and roles. A *constraint* is a syntactic object of one of the forms *x: C*, xR_y where *C* is a concept and *R* is a role.

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* the constraint $x{:}C$ if $\alpha(x) \in C^{\mathcal{I}}$, and α *satisfies* xRy if $(\alpha(x), \alpha(y)) \in R^{\mathcal{I}}$. A constraint c is *satisfiable* if there is 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* S if α satisfies every constraint in S. S is *satisfiable* if there is an interpretation $\mathcal{I}$ and an $\mathcal{I}$-assignment α such that α satisfies S. Using the results reported in [Schmidt-Schauß and Smolka, 1988], it is easy to see that a concept C is satisfiable if and only if the constraint system $\{x{:}C\}$ is satisfiable.

In order to check whether a concept C is satisfiable we start with the constraint system $S = \{x{:}C\}$, and in subsequent steps, we add constraints according to a set of propagation rules, until the resulting system is *complete*, i.e. none of the propagation rules is applicable. At that point, either the complete system S' contains a contradiction (called *clash*), or an interpretation satisfying C can be obtained from S'. Each propagation rule deals with one of the constructs allowed in the language, and the computational complexity of the method crucially depends on their form. It is interesting to observe that, from the logical viewpoint, the method is essentially a tableaux calculus modified with suitable control mechanisms.

In order to use the above technique for subsumption, we can rephrase subsumption in terms of unsatisfiability, exploiting the fact that C is subsumed by D if and only if $C \sqcap \neg D$ is unsatisfiable. The problem with this approach is that, in general, $C \sqcap \neg D$ may include constructs which are not present in the language in which C and D are expressed. We show in the following how to deal with this problem for the language $\mathcal{FL}^-$.

We first define a useful property of a class of concept languages, which includes $\mathcal{FL}^-$. A concept is said to be *conjunction-free* if it contains no conjunction of concepts (i.e. it does not contain the symbol $\sqcap$ applied to concepts). It is easy to see that in a language including neither disjunction nor qualified existential quantification, a concept D can be rewritten into an equivalent concept of the form $D_1 \sqcap \ldots \sqcap D_n$, where each D_i is conjunction-free, and is called a *conjunction-free component* of D. This can be done by means of the following rewriting rule:

$$\forall R.(C \sqcap D) \longrightarrow \forall R.C \sqcap \forall R.D$$

Theorem 3.1 *For every pair of concepts C, D of a language including neither union nor qualified existential quantification, C is subsumed by D iff for every conjunction-free component D_i of D, $C \sqcap \neg D_i$ is unsatisfiable.*

Notice that, although the concept $\neg D_i$ may contain the negation of a non-primitive concept, it can be rewritten in linear time into an equivalent *simple* concept (i.e. a concept where negation occurs only in front of primitive concepts) by using suitable rewriting rules.

When $\mathcal{FL}^-$ is the language used to express C and D, the simple concept resulting from the rewriting of $C \sqcap \neg D_i$ does not belong to the original language, not only because primitive concepts are negated, but for new constructs introduced through the rewriting rules. In particular, it is easy to see that $C \sqcap \neg D_i$ is a concept of $\mathcal{ALE}$, an extension of $\mathcal{FL}^-$ with qualified existential quantification and negation of primitive concepts. It follows from the results reported in [Schmidt-Schauß and Smolka, 1988] that the propagation rules for satisfiability checking in $\mathcal{ALE}$ are:

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

   if $x{:}C_1 \sqcap C_2$ is in S, and either $x{:}C_1$ or $x{:}C_2$ is not in S
2. $S \rightarrow_{\exists} \{xRy,\ y{:}C\} \cup S$

   if $x{:}\exists R.C$ is in S, there is no z such that both xRz and $z{:}C$ are in S, and y is a new variable
3. $S \rightarrow_{\forall} \{y{:}C\} \cup S$

   if $x{:}\forall R.C$ is in S, xRy is in S, and $y{:}C$ is not in S.

Notice that the complete constraint system S' obtained from $S = \{x{:}C\}$ by applying the above rules, may have a number of constraints that is exponential in the size of C, in particular because of the number of variables generated by the $\rightarrow_{\exists}$-rule. It has recently been shown that this is due to the interaction of existential and universal quantification, which makes the unsatisfiability problem (and therefore subsumption) in $\mathcal{ALE}$ NP-complete [Donini et al., 1990].

This problem does not arise when computing subsumption in $\mathcal{FL}^-$. Indeed, if we look at the form of the concept obtained by negating the candidate subsumer and rewriting it into a simple concept, we realize that the body of a qualified existential quantification may contain only a universal quantification of the form $\forall R.\bot$. This implies that the inherent complexity given by the interaction of existential and universal quantification in $\mathcal{ALE}$ is not present in the extension of $\mathcal{FL}^-$ used to compute subsumption.

This allows us to modify the $\rightarrow_{\exists}$-rule in such a way that only one variable y is generated for all the constraints of the forms $x{:}\exists R.C$, $x{:}\exists R$. The correctness of this method stems from the fact that, even if different variables are generated for each of the above constraints, they all share the same properties of y. Therefore, in order to check whether a contradiction arises in the system, it is sufficient to consider only the variable y.

The above modification directly leads to a polynomial time algorithm for subumption in $\mathcal{FL}^-$. In the following sections, similar techniques are used to show the tractability of two extensions of $\mathcal{FL}^-$.

4 The Language $\mathcal{PL}_1$

The language considered in this section, called $\mathcal{PL}_1$, is an extension of $\mathcal{FL}^-$ with negation of primitive concepts, number restrictions and inverse roles. Its syntax is specified by the following rules:

$$C, D \longrightarrow A \mid \neg A \mid \top \mid \bot \mid C \sqcap D \mid \forall R.C \mid (\geq n\, R) \mid (\leq n\, R)$$

$$R \longrightarrow P \mid P^{-1}$$

where R denotes a role, and P a primitive role[1].

In order to deal with subsumption in $\mathcal{PL}_1$, we reduce this problem to the unsatisfiability problem in a suitable extension of $\mathcal{PL}_1$.

Notice that, since $\mathcal{PL}_1$ includes neither disjunction nor qualified existential quantification C is subsumed by D iff for every conjunction-free component D_i of D, $C \sqcap \neg D_i$ is unsatisfiable. Moreover, as pointed out in Section 3, $\neg D_i$ can be rewritten into an equivalent simple concept. The resulting concept may not belong to the language $\mathcal{PL}_1$, because $\neg\forall R.C$ is rewritten as $\exists R.\neg C$. We call $\mathcal{PL}_1{}^+$ the language constituted by concepts of one of the form: C, D, $C \sqcap D$, where C is a concept of $\mathcal{PL}_1$, and D is a simple concept resulting from rewriting a conjunction-free $\mathcal{PL}_1$-concept. In the rest of the section, we always refer to concepts of $\mathcal{PL}_1{}^+$, unless otherwise stated. As a notation, we say that xRy *holds* in a $\mathcal{PL}_1{}^+$-constraint system S if:

- R is a primitive role P and $xPy \in S$;
- R is P^{-1} and $yPx \in S$.

The propagation rules for $\mathcal{PL}_1{}^+$-constraint system are as follows:

1. $S \rightarrow_\sqcap \{x{:}C_1,\ x{:}C_2\} \cup S$
   if $x{:}C_1 \sqcap C_2$ is in S, and either $x{:}C_1$ or $x{:}C_2$ is not in S
2. $S \rightarrow_\forall \{y{:}C\} \cup S$
   if $x{:}\forall R.C$ is in S, xRy holds in S, and $y{:}C$ is not in S
3. $S \rightarrow_{-1} \{yPx\} \cup S$
   if $xP^{-1}y$ is in S and yPx is not in S
4. $S \rightarrow_\leq [y/z]S$
   if $x{:}(\leq 1\, R)$ is in S, xRy and xRz hold in S with $y \neq z$, and $[y/z]S$ is obtained from S by replacing y with z
5. $S \rightarrow_\geq \{xRy\} \cup S$
   if $x{:}(\geq n\, R)$ is in S, with $n > 0$, y is a new variable, and there is no z such that xRz holds in S
6. $S \rightarrow_\exists \{xRy,\ y{:}C\} \cup S$
   if $x{:}\exists R.C$ is in S, y is a new variable and there is no z such that both xRz holds in S and $z{:}C$ is in S

Notice that the construct $(\geq n\, R)$ is treated like the unqualified existential $\exists R$, and, as discussed in Section 3 for $\mathcal{FL}^-$, only one variable is generated for all the constraints of the forms $x{:}(\geq n\, R)$, $x{:}\exists R.C$. We show in the sequel that this simplification, which is crucial for the tractability of the method, does not affect its correctness. Notice also that the construct $(\leq n\, R)$ is taken into account only if $n = 1$. This is because using the above rules, no more than 2 variables may be linked to the same variable through a given role.

[1]Note that $\mathcal{PL}_1$ allows for the representation of unqualified existential role quantification: indeed, $\exists R$ can be expressed as $(\geq 1\, R)$.

In a $\mathcal{PL}_1{}^+$-constraint system a clash is a set of constraints of one of the following forms:

1. $\{x{:}\bot\}$;
2. $\{x{:}A, x{:}\neg A\}$;
3. $\{x{:}(\geq n\, R), x{:}(\leq m\, R)\}$, with $n > m$;
4. $\{x{:}(\leq 0\, P), xPy\}$;
5. $\{x{:}(\leq 0\, P^{-1}), yPx\}$.

The next theorem states that the above propagation rules preserve satisfiability.

Theorem 4.1 *Let S be a constraint system. If S' is obtained from S by the application of the above propagation rules, then S is satisfiable if and only if S' is satisfiable.*

We say that S' is a *completion* of $\{x{:}C\}$, if S' is complete, and is obtained from $\{x{:}C\}$ by giving priority to the application of the $\rightarrow_\exists$-rule. It is easy to see that, up to variable renaming, only one completion can be derived from $\{x{:}C\}$.

It is important to notice that, differently from the case of $\mathcal{ALE}$-constraint systems, there is not a direct correspondence between a clash-free completion of $\{x{:}C\}$ and an assignment satisfying it. This is due to the $\rightarrow_\geq$-rule dealing with concepts of the form $(\geq n\, R)$, which are treated as $\exists R$, independently of the value of n. However, the crucial point is that it is always possible to obtain from a clash-free completion S' an interpretation $\mathcal{I}$ and an $\mathcal{I}$-assignment α that satisfies S'. In particular, $\mathcal{I}$ and α are built in such a way that to each variable in S' there corresponds a suitable number of objects in the domain of $\mathcal{I}$, all with the same properties, in order for α to satisfy all constraints of the form $x{:}(\geq n\, R)$.

Theorem 4.2 *Let C be a $\mathcal{PL}_1{}^+$-concept, and let S' be the completion of the constraint system $\{x{:}C\}$. Then S' is satisfiable if and only if it contains no clash.*

The entire method for testing subsumption in $\mathcal{PL}_1$—and satisfiability as a special case—can be summed up as follows. In order to check if C is subsumed by D, we compute the conjunction-free components $D_1,\ldots,D_n$ of D, and for each i we rewrite $\neg D_i$ into the simple concept $D_i{}'$. Finally, we check if for each i, the completion of $C \sqcap D_i{}'$ contains a clash.

The tractability of subsumption in $\mathcal{PL}_1$ can be proved by showing that computing the completion of a constraint system $\{x{:}C \sqcap \neg D_i\}$ requires polynomial time. The proof is based on the following observations. First of all, giving priority to the $\rightarrow_\exists$-rule leads to a constraint system of the form $S = \{x{:}C, xR_1y_1, y_1R_2y_2, \ldots, y_{h-1}R_hy_h, y_h{:}F\}$, where F is not of the form $\exists R.E$. Second, the number of variables generated in the completion of a constraint system of the form S is bounded by $(g+1)\cdot(h+1)$, where g is the number of subconcepts (i.e. substrings that are concepts) of C of the form $(\geq n\, R)$. Finally, the cost of the entire method is polynomially bounded by the number of variables in the computed completion.

Theorem 4.3 *The subsumption problem in $\mathcal{PL}_1$ can be solved in polynomial time.*

It is interesting to observe that none of the constructs presented in Section 2 can be added to $\mathcal{PL}_1$ without sacrificing tractability. In particular, it follows from the results reported in [Nebel, 1988] that the addition of role conjunction leads to co-NP-hardness of subsumption. Also, adding negation of non-primitive concepts leads to PSPACE-hardness, whereas adding union makes subsumption co-NP-hard [Schmidt-Schauß and Smolka, 1988]. Moreover, the addition of qualified existential quantification results in the language $\mathcal{ALE}$, and therefore leads to NP-hardness of subsumption [Donini et al., 1990].

We now show that the same problems arise when $\mathcal{PL}_1$ is extended with role chaining. In particular, any concept D of $\mathcal{ALE}$ can be translated into a concept D' of $\mathcal{PL}_1$ + role chaining such that D is satisfiable if and only if D' is satisfiable. D' is obtained from D by substituting each subconcept of D having the form $\exists P.C$ with $(\geq 1\ (P \circ Q)) \sqcap \forall(P \circ Q \circ Q^{-1}).C$, where Q is a new primitive role used only in the substitution of that subconcept. Since the above translation is clearly polynomial, we can conclude that unsatisfiability—and therefore subsumption—in $\mathcal{PL}_1$ with role chaining is NP-hard.

5 The language $\mathcal{PL}_2$

The language considered in this section, called $\mathcal{PL}_2$, is an extension of $\mathcal{FL}^-$ with role conjunction, role chaining, and inverse roles. Its syntax is specified by the following rules:

$$\begin{array}{rcl} C, D & \longrightarrow & A \mid C \sqcap D \mid \forall R.C \mid \exists R \\ R & \longrightarrow & P \mid R^{-1} \mid R_1 \sqcap R_2 \mid R_1 \circ R_2 \end{array}$$

where P is a primitive role, and R, R_1 and R_2 are arbitrary roles. Notice that, as for $\mathcal{FL}^-$, every $\mathcal{PL}_2$-concept is satisfiable.

It is possible to verify that every role of $\mathcal{PL}_2$ can be written in a so-called normal form, where the constructor for the inverse role is applied only to primitive roles. The following rewriting rules can be used to transform any role into an equivalent role having such a form. In the sequel, we only refer to roles in normal form.

$$\begin{array}{rcl} (R_1 \sqcap R_2)^{-1} & \longrightarrow & R_1^{-1} \sqcap R_2^{-1} \\ (R_1 \circ R_2)^{-1} & \longrightarrow & R_2^{-1} \circ R_1^{-1} \\ (R^{-1})^{-1} & \longrightarrow & R \end{array}$$

As done for $\mathcal{PL}_1$, we reduce the subsumption problem in $\mathcal{PL}_2$ to the unsatisfiability problem in a suitable extension of the language. Moreover, since $\mathcal{PL}_2$ includes neither disjunction nor qualified existential quantification, it is sufficient to consider the unsatisfiability problem for concepts of the form $C \sqcap \neg D$, where D is conjunction-free.

We call $\mathcal{PL}_2{}^+$ the language constituted by concepts of one of the forms: C, $\neg D$, $C \sqcap \neg D$, where C is a concept of $\mathcal{PL}_2$, and D is a conjunction-free concept of $\mathcal{PL}_2$. In the rest of this section we always refer to concepts of $\mathcal{PL}_2{}^+$, unless otherwise stated.

In order to present the propagation rules for $\mathcal{PL}_2{}^+$-constraint systems, we need the following definition. We say that xRy *holds in* a $\mathcal{PL}_2{}^+$-constraint system S if:

- R is a primitive role P and $xPy \in S$;
- R is P^{-1} and $yPx \in S$;
- R is $R_1 \sqcap R_2$, and both xR_1y and xR_2y hold in S;
- R is $R_1 \circ R_2$ and there is a z such that both xR_1z and zR_2y hold in S.

The propagation rules for $\mathcal{PL}_2{}^+$-constraint systems are the $\rightarrow_\sqcap$-rule, $\rightarrow_\forall$-rule, and the $\rightarrow_{\neg\exists}$-rule already defined for $\mathcal{PL}_1$, plus the following ones:

4. $S \rightarrow_{\neg\forall} \{xRy, y{:}\neg C\} \cup S$

   if $x{:}\neg\forall R.C$ is in S, y is a new variable and there is no z such that $z{:}\neg C$ is in S
5. $S \rightarrow_{\sqcap_\rho} \{xR_1y, xR_2y\} \cup S$

   if $x(R_1 \sqcap R_2)y$ is in S, and either xR_1y or xR_2y is not in S
6. $S \rightarrow_\circ \{xR_1z, zR_2y\} \cup S$

   if $x(R_1 \circ R_2)y$ is in S, z is a new variable and there is no w such that both xR_1w and wR_2y are in S

Notice that variables are only generated by the $\rightarrow_{\neg\forall}$-rule. In particular only one variable is generated for each constraint of the form $x{:}\neg\forall R.C$. On the other hand, no rule is needed for constraints of the form $x{:}\exists R$, since we can avoid to produce new variables for them. This is crucial for keeping the size of the system polynomial. Indeed, even if one generates only one variable for each constraint of the form $x{:}\exists R$ (as done in $\mathcal{FL}^-$), the presence of role conjunction may lead to an exponential number of variables in the completion of the system.

It is easy to see that, up to variable renaming, only one completion can be derived from a $\mathcal{PL}_2{}^+$-constraint system. The following theorem states that the above rules preserve satisfiability.

Theorem 5.1 *Let S be a constraint system. Then, if S' is obtained from S by application of the above rules, then S is satisfiable if and only if S' is satisfiable.*

In $\mathcal{PL}_2{}^+$-constraint systems, a *clash* is defined as a set of constraints of one of the following forms:

1. $\{x{:}A, x{:}\neg A\}$;
2. $\{x{:}\exists R_1, x{:}\neg\exists R_2\}$,
   where the completion of the new constraint system $\{xR_1y, x{:}\forall R_2.B\}$ (B is a new symbol denoting a primitive concept) contains a constraint $z{:}B$, for some variable z.

Notice that checking whether a completion of a constraint system S has a clash may require to compute the completion of other systems, each one constituted by two constraints of the form $\{xR_1y, x{:}\forall R_2.B\}$. It is easy to see that if such a completion contains a constraint $z{:}B$, then the existence of an object y related to x through the role R_1 implies that x is also related to an object z through the role R_2, and the pair of constraints $\{x{:}\exists R_1, x{:}\neg\exists R_2\}$ is unsatisfiable.

Theorem 5.2 *Let S be the completion of the constraint system $\{x{:}C \sqcap \neg D\}$. Then S is satisfiable if and only if it contains no clash.*

The tractability of subsumption in $\mathcal{PL}_2$ can be proved by showing that computing the completion S' of a constraint system $\{x: C \sqcap \neg D_i\}$ requires polynomial time, and moreover checking whether S' has a clash is also a polynomial task. With regard to the first point, the basic observation is that for every subconcept C' of $C \sqcap \neg D$ at most one constraint of the form $z: C'$ can be contained in S'. Therefore, the number of constraints in S' is bounded by the size of $C \sqcap \neg D$. With regard to the second point, notice that the worst case requires considering every pair of constraints of S', and checking if they constitute a clash of the form $\{x: \exists R_1, x: \neg\exists R_2\}$. Since the cost of such a check is polynomially bounded by the size of R_1 and R_2, we can state the following theorem.

Theorem 5.3 *The subsumption problem in $\mathcal{PL}_2$ can be solved in polynomial time.*

As for $\mathcal{PL}_1$, it is possible to show that none of the constructs presented in Section 2 can be added to $\mathcal{PL}_2$ without sacrificing tractability. In particular, the addition of number restrictions leads to co-NP-hardness of subsumption [Nebel, 1988]. The addition of the negation of primitive concepts results in a language that is a superset of a language called $\mathcal{ALR}$, shown intractable in [Donini et al., 1991a], whereas the addition of qualified existential quantification results in a superset of $\mathcal{ALE}$. Finally, it is possible to show that adding union to $\mathcal{PL}_2$ makes subsumption intractable. In fact, one can prove that even the simpler problem of checking whether C is subsumed by $D_1 \sqcup \cdots \sqcup D_n$, where $C, D_1, \ldots, D_n$ are $\mathcal{FL}^-$-concepts is already intractable (see [Lenzerini and Schaerf, 1991]).

6 Conclusion

We have presented two concept laguages, called PL_1 and PL_2, that include as many as possible of the constructs usually considered in terminological reasoning, while avoiding combinations that are proved harmful for tract ability.

It is interesting to observe that we can extend both PL_1 and PL_2 with functional roles, functional role value map and disjunction of primitive concepts without endangering the tractability of subsumption.

The algorithms for checking subsumption in PL_1 and PL_2 have been designed by exploiting a general technique for satisfiability checking in concept languages. Such a technique is the basis of recent complexity results about a wide class of concept languages [Schmidt-SchauB and Smolka, 1988,Donini et al., 1991a], These results allow us to derive lower bounds for the complexity of almost all the possible combinations of the constructs presented in Section 2. There is in fact one single exception, because it is still unknown if subsumption is tractable in the language obtained from FL^- by adding number restrictions and role chaining. It turns out that we can state an interesting property of VC\ and PL_2. *Let £ be a language extending FL^- with any combination of the constructs presented in Section* 2, *except for the combination of number restrictions and role chaining; if the subsumption problem in £ is tractable, then £ is a sublanguage of either V£\ or PL_2.*

Therefore, on the assumption that languages are characterized by the set of constructs they provide, the two languages presented in this paper provide a solution to the problem of singling out an optimal trade-off between expressive power and computational complexity.

References

[Brachman and Levesque, 1984] R. J. Brachrnan, II. J. Levesque. "The tractability of subsumption in frame-based description languages." In *Proceedings of the Fourth National Conference, on Artificial Intelligence,* pp. 34-37, Austin, Texas, 1984.

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

[Donini et al., 1990] F.M. Donini, B. Iloilmuler, M. Lenzerini, A. Marchetti Spaccamela, D. Nardi, \V. Nutt, "The complexity of existential quantification in concept languages." DFKI-Report, DFKI, Postfach 2080, D-6750 Kaiserslautern, Germany, 1990.

[Donini et al., 1991a] F.M. Donini, M. Lenzerini, D. Nardi, W. Nutt. "The complexity of concept languages." To appear in *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning,* Boston, 1991.

[Donini et al., 1991b] F.M. Donini, M. Lenzerini, D. Nardi, W. Nutt. "Tractable concept languages." Technical Report D.1.2.b,c, ESPRIT Basic Research Action 3012 Compulog, 1991.

[Lenzerini and Schaerf, 1991] M. Lenzerini, A. Schaerf. "Concept languages as query languages." To appear in *Proceedings of the Ninth National Conference on Artificial Intelligence,* Anaheim, 1991.

[Levesque and Brachman, 1987] H. J. Levesque, R. J. Brachman. "Expressiveness and tractability in knowledge representation and reasoning." *Computational Intelligence,* 3:78-93, 1987.

[Nebel, 1988] B. Nebel. "Computational complexity of terminological reasoning in BACK." *Artificial Intelligence,* 34(3):371-383, 1988.

[Nebel and Smolka, 1990] B. Nebel, G. Smolka. "Representation and reasoning with attributive descriptions. " in K.H. Blasius, U. Hedtstiick, C-R. Rollinger (Eds.) *Sorts and Types in Artificial Intelligence,* Lecture Notes in Artificial Intelligence 418, Springer Verlag, pp. 112-139, 1990.

[Schmidt-SchauB and Smolka, 1988] M. Schmidt-SchauB, G. Smolka. "Attributive concept descriptions with unions and complements." SEKI Report SR-88-21, FB Informatik, Universitat Kaiserslautern, D-6750, Kaiserslautern, Germany, 1988. To appear in *Artificial Intelligence,*