# Decidable Reasoning in Terminological Knowledge Representation Systems

Martin Buchheit*
German Research Center for Artificial Intelligence (DFKI)
Stuhlsatzenhausweg 3, D-6600 Saarbrücken, Germany
e-mail: buchheit@dfki.uni-sb.de

Francesco M. Donini, Andrea Schaerf
Dipartimento di Informatica e Sistemistica
Università di Roma "la Sapienza", Via Salaria 113, I-00198 Roma, Italy
e-mail: {donini,aschaerf}@assi.ing.uniroma1.it

**Abstract**

Terminological Knowledge Representation Systems (TKRS) are tools for designing and using knowledge bases that make use of terminological languages (or concept languages). We analyze from a theoretical point of view a TKRS whose capabilities go beyond the ones of presently available TKRS. The new features studied, all of practical interest, can be summarized in three main points. First, we consider a highly expressive terminological language, called $\mathcal{ALCNR}$, including general complements of concepts, number restrictions and role conjunction. Second, we allow to express inclusion statements between general concepts, and terminological cycles as a particular case. Third, we prove the decidability of a number of desirable TKRS-deduction services (like satisfiability-, subsumption- and instance checking) through a sound, complete and terminating calculus for reasoning in $\mathcal{ALCNR}$-knowledge bases. Our calculus extends the general technique of constraint systems and can be easily turned into a procedure using exponential space. As a byproduct of the proof, we get also the result that inclusion statements in $\mathcal{ALCNR}$ can be simulated by terminological cycles, if descriptive semantics is adopted.

*The research was partly done while the first author was visiting the Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza".

# Contents

# 1 Introduction

A general characteristic of many proposed Terminological Knowledge Representation Systems (TKRSs) such as KRIPTON [BPL85], NIKL [KBR86], BACK [QK90], LOOM [MB87], CLASSIC [BBMAR89], $\mathcal{KRIS}$ [BH91], and others (see [Ric91, WS92]), is that they are made up of two different components. Informally speaking, the first is a general schema concerning the classes of individuals to be represented, their general properties and mutual relationships, while the second is a (partial) instantiation of this schema, containing assertions relating either individuals to classes, or individuals to each other. This characteristic, which the mentioned proposals inherit from the seminal TKRS KL-ONE [BS85], is shared also by several proposals of Database models such as Abrial's [Abr74], CANDIDE [BGN89], and TAXIS [MBW80].

Retrieving information in actual Knowledge Bases (KBs) built up using one of these systems is a deductive process involving both the schema (TBox) and its instantiation (ABox). In fact, the TBox is not just a set of constraints on possible ABoxes, but contains intensional information about classes. This information is taken into account when answering queries to the KB.

During the realization and use of a KB, a TKRS should provide a mechanical solution for at least the following problems (from now on, we use the word *concepts* to refer to classes):

1. *Concept satisfiability*: given a KB and a concept $C$, does there exist at least one model of the KB assigning a non-empty extension to $C$? This is important not only to rule out meaningless concepts in the KB design phase, but also in processing the user's queries, to eliminate parts of a query which cannot contribute to the answer.

2. *Subsumption*: given a KB and two concepts $C$ and $D$, is $C$ more general than $D$ in any model of the KB? Subsumption detects implicit dependencies among the concepts in the KB.

3. *KB-satisfiability*: are an ABox and a TBox consistent with each other? That is, does the KB admit a model? A positive answer is useful in the validation phase, while the negative answer can be used to make inferences in refutation-style. The latter will be precisely the approach taken in this paper.

4. *Instance checking*: given a KB, an individual $a$ and a concept $C$, is $a$ an instance of $C$ in any model of the KB? Note that retrieving all individuals described by a given concept (a *query* in the Database lexicon) can be formulated as many parallel instance checkings.

The above questions can be precisely characterized once the TKRS is given a semantics (see next section), which defines models of the KB and gives a meaning to expressions in the KB. Once the problems are formalized, one can start both a theoretical analysis of them, and—maybe independently—a search for reasoning procedures accomplishing the tasks. Completeness of procedures can be judged with respect to the formal statements of the problems.

Up to now, all the proposed systems (except for $\mathcal{KRIS}$) give incomplete procedures for solving the above problems 1–4. That is, some inferences are missed, in some cases without a precise semantical characterization of which ones are. If the designer or the user needs a (more) complete reasoning, she/he must either write programs in a suitable programming language (as in the Database proposal of Abrial, and in TAXIS), or define appropriate inference rules completing the inference capabilities of the system (as in BACK, LOOM, and CLASSIC). From the theoretical point of view, for several systems (e.g. LOOM) it is not even known if complete procedures can ever exist—i.e., the decidability of the corresponding problems is not known.

Recent research on the computational complexity of subsumption had an influence in many TKRSs on the choice for incomplete procedures. The research started with [BL84], which analyzed complexity of subsumption between pure concept expressions, abstracting from KBs (we call this problem in the sequel as *pure subsumption*). The motivation for focusing on such a small problem was that pure subsumption is a fundamental inference in any TKRS. It turned out that pure subsumption is tractable (i.e. worst-case polynomial-time solvable) for simple languages, and intractable (e.g. NP-hard, coNP-hard or PSPACE-hard) for slight extensions of such languages, as subsequent research definitely confirmed [Neb88, DLNN91a, DLNN91b, SSS91, DHL$^+$92]. Also, beyond computational complexity, pure subsumption was proved undecidable in the two TKRS KL-ONE [SS89] and NIKL [Pat89].

Note that extending the language results in enhancing its expressiveness, therefore the result of this research could be summarized as: The more a TKRS language is expressive, the higher is the computational complexity of reasoning in that language—as Levesque first noted [Lev84]. This result has been interpreted in two different ways, leading to two different TKRSs design philosophies:

1. 'General-purpose languages for TKRSs are intractable, or even undecidable, and tractable languages are not expressive enough to be of practical interest'. Following this interpretation, in several TKRSs (such as NIKL, LOOM and BACK) incomplete procedures for pure sub-

sumption are considered satisfactory (e.g. see [MB92] for LOOM). Once completeness is abandoned for this basic subproblem, completeness of overall reasoning procedures is not an issue anymore; but other issues arise, such as how to compare incomplete procedures [HKNP92], and how to judge a procedure "complete enough" [Mac91]. As a practical tool, inference rules can be used in such systems to achieve the expected behavior of the KB w.r.t. the information contained in it.

2. 'A TKRS is (by definition) general-purpose, hence it must provide tractable and complete reasoning to a user'. Following this line, other TKRSs (such as KRIPTON and CLASSIC) provide limited tractable languages for expressing concepts, following the "small-can-be-beautiful" approach (see [Pat84]). The gap between what is expressible in the TKRS language and what is needed to be expressed for the application is then filled by the user, by a (sort of) programming with inference rules. Of course, the usual problems present in program development and debugging arise [McG92].

What is common to both approaches is that a user must cope with incomplete reasoning. The difference is that in the former approach, the burden of regaining useful yet missed inferences is mostly left to developers of the TKRS (and the user is supposed to specify what is "complete enough"), while in the latter this is mainly left to the user. These are perfectly reasonable approaches in a practical context, where incomplete procedures and specialized programs are often used to deal with intractable problems. In our opinion incomplete procedures are just a provisional answer to the problem—the best possible up to now. In order to improve on such an answer, a theoretical analysis of the general problems 1–4 is to be done. But most importantly, theoretical analysis is needed for making terminological cycles (see [Neb90a, Chapter 5]) fully available in TKRSs. Such a feature is of undoubtable practical interest [Mac92], yet present TKRSs can only approximate cycles, by using forward inference rules.

Previous theoretical results do not deal with the problems 1–4 in their full generality. For example, the problems are studied in [Neb90a, Chapter 4], but only incomplete procedures are given, and cycles are not considered. In [DLNS92] the complexity of instance checking has been analyzed, but only KBs without a TBox are treated. Instance checking has also been analyzed in [Vil91], but addressing only that part of the problem which can be performed as parsing.

Previous theoretical work on cycles was done in [Baa90b, Baa90a, BBH+90, Neb90a, Neb91, Sch91], but considering KBs formed by the TBox alone. Moreover, these approaches do not deal with number restrictions (except

for [Neb90a, Section 5.3.5]), which are a basic feature already provided by TKRSs, and the techniques used do not seem easily extensible to reasoning with ABoxes. We compare in detail several of these works with ours in Section 5.

In this paper, we propose a TKRS equipped with a highly expressive language, including constructs of practical interest, and prove decidability of problems 1–4. In particular, our system uses the language $\mathcal{ALCNR}$, which supports general complements of concepts, number restrictions and role conjunction. Moreover, the system allows one to express inclusion statements between general concepts and, as a particular case, terminological cycles. We prove decidability by means of a suitable calculus, which is developed extending the quite well established framework of constraint systems (see [DLNN91a, SSS91]), thus exploiting a uniform approach to reasoning in TKRSs. Moreover, our calculus can easily be turned into a decision procedure.

The paper is organized as follows. In Section 2 we introduce the language, and we give it a Tarski-style extensional semantics, which is the most commonly used. Using this semantics, we establish relationships between problems 1–4 which allow us to concentrate only on KB-satisfiability. In Section 3 we provide a calculus for KB-satisfiability, and show correctness and termination of the calculus. Hence, we conclude that KB-satisfiability is decidable in $\mathcal{ALCNR}$, which is the main result of this paper. The calculus we provide to show decidability works in double exponential space. In Section 4 we consider a refinement of our calculus, working in exponential space. In Section 5 we compare our approach with previous results on decidable TKRSs, and we establish the equivalence of general (cyclic) inclusion statements and general concept definitions using the descriptive semantics. Finally, we discuss in detail several practical impacts of our results in Section 6.

# 2 Preliminaries

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

## 2.1 Concept Languages

In concept languages, concepts represent the classes of objects in the domain of interest, while roles represent binary relations between objects. Complex concepts and roles can be defined by means of suitable constructors applied to primitive concepts and primitive roles. In particular, concepts and roles in $\mathcal{ALCNR}$ can be formed by means of the following syntax ($A$ denotes a primitive concept, $P_i$ (for $i = 1, \ldots, k$) denotes a primitive role, $C$ and $D$ denote arbitrary concepts and $R$ an arbitrary role):

$$
\begin{array}{rcll}
C, D & \longrightarrow & A \mid & \text{(primitive concept)} \\
 & & \top \mid & \text{(top)} \\
 & & - \mid & \text{(bottom)} \\
 & & (C \sqcap D) \mid & \text{(conjunction)} \\
 & & (C \sqcup D) \mid & \text{(disjunction)} \\
 & & \neg C \mid & \text{(complement)} \\
 & & \forall R.C \mid & \text{(universal quantification)} \\
 & & \exists R.C \mid & \text{(existential quantification)} \\
 & & (\geq n\, R) \mid (\leq n\, R) & \text{(number restrictions)} \\
R & \longrightarrow & P_1 \sqcap \cdots \sqcap P_k & \text{(role conjunction)}
\end{array}
$$

When no confusion arises we drop the brackets around conjunctions and disjunctions. A *subconcept* of a concept $C$ is any substring of $C$ (including $C$ itself) that is a concept, according to the syntax rules. Different occurrences of substrings of $C$ are considered as different subconcepts, even if they are syntactically equal. Notice that the number of subconcepts of $C$ is bounded by the length of the string expressing $C$.

We interpret concepts as subsets of a domain and roles as binary relations over a domain. More precisely, an *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a nonempty set $\Delta^{\mathcal{I}}$ (the *domain* of $\mathcal{I}$) and a function $\cdot^{\mathcal{I}}$ (the *extension function* of $\mathcal{I}$) which 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 ($\sharp\{\}$ denotes the cardinality of a set):

$$
\begin{aligned}
\top^{\mathcal{I}} &= \Delta^{\mathcal{I}} \\
-^{\mathcal{I}} &= \emptyset \\
(C \sqcap D)^{\mathcal{I}} &= C^{\mathcal{I}} \cap D^{\mathcal{I}} \\
(C \sqcup D)^{\mathcal{I}} &= C^{\mathcal{I}} \cup D^{\mathcal{I}} \\
(\neg C)^{\mathcal{I}} &= \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \\
(\forall R.C)^{\mathcal{I}} &= \{d_1 \in \Delta^{\mathcal{I}} \mid \forall d_2 : (d_1, d_2) \in R^{\mathcal{I}} \rightarrow\ d_2 \in C^{\mathcal{I}}\} \\
(\exists R.C)^{\mathcal{I}} &= \{d_1 \in \Delta^{\mathcal{I}} \mid \exists d_2 : (d_1, d_2) \in R^{\mathcal{I}} \wedge d_2 \in C^{\mathcal{I}}\}
\end{aligned}
$$

$$
\begin{aligned}
(\geq n\, R)^{\mathcal{I}} &= \{d_1 \in \Delta^{\mathcal{I}} \mid \quad \sharp\{d_2 \mid (d_1, d_2) \in R^{\mathcal{I}}\} \quad \geq n\} \\
(\leq n\, R)^{\mathcal{I}} &= \{d_1 \in \Delta^{\mathcal{I}} \mid \quad \sharp\{d_2 \mid (d_1, d_2) \in R^{\mathcal{I}}\} \quad \leq n\} \\
(P_1 \sqcap \cdots \sqcap P_k)^{\mathcal{I}} &= P_1^{\mathcal{I}} \cap \cdots \cap P_k^{\mathcal{I}}
\end{aligned}
$$

## 2.2 Knowledge Bases

A knowledge base built by means of concept languages is generally formed by two components: The *intensional* one, called TBox, and the *extensional* one, called ABox.

We first turn our attention to the intensional component of a knowledge base, i.e. the TBox. As we said before, the intensional level specifies the properties of the concepts of interest in a particular application. Syntactically, such properties are expressed in terms of so-called *inclusion statements* (see [Neb90a, Chapter 3]). An inclusion statement (or simply inclusion) has the form

$$C \sqsubseteq D$$

where $C$ and $D$ are two arbitrary concepts. Intuitively, the statement specifies that every instance of $C$ is also an instance of $D$. More precisely, an interpretation $\mathcal{I}$ *satisfies* the inclusion $C \sqsubseteq D$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$.

A TBox is a finite set of inclusions. An interpretation $\mathcal{I}$ is a *model* for a TBox $\mathcal{T}$ if $\mathcal{I}$ satisfies all inclusions in $\mathcal{T}$.

Many TKRSs provide the user with mechanisms for stating *concept definitions* (e.g. [Neb90a, Section 3.2]) of the form $A \doteq D$ (interpreted as set equality), or $A \dot{\leq} D$ (primitive concept definition, interpreted as set inclusion), with the restrictions that the left-hand side concept $A$ must be a concept name[1], that for each concept name at most one definition is allowed, and that no so-called *terminological cycles* are allowed, i.e. no concept name may occur—neither directly nor indirectly—within its own definition.

We do not impose any of these restrictions to the form of inclusions, obtaining statements that are syntactically more expressive than concept definitions. In particular, a definition of the form $A \doteq D$ can be expressed in our system using the pair of inclusions $A \sqsubseteq D$ and $D \sqsubseteq A$, whereas an inclusion of the form $C \sqsubseteq D$, where $C$ and $D$ are arbitrary concepts, cannot be expressed with concept definitions. Moreover, cyclic inclusions are allowed in our statements, realizing terminological cycles.

[1] In many TKRSs, what we call "primitive concepts" are called *concept names*, preserving the term "primitive concept" for concept names that do not appear on the left hand side of a definition. However, in our setting this distinction is not necessary, since we do not use concept definitions.

As shown in [Neb91], there are at least three types of semantics for terminological cycles, namely the least fixed point, the greatest fixed point, and the descriptive semantics. However, fixed point semantics apply only to fixed point statements like $A \doteq D$ (where $D$ is a "function" of $A$, i.e. $A$ appears in $D$), which are less general than our inclusion statements. Instead, the descriptive semantics interprets statements as just restricting the set of possible models, with no definitional import. Hence, it can be suitably extended to our case, and is exactly the one we adopt.

We can now turn our attention to the *extensional level*, i.e. the ABox. The ABox essentially allows one to specify instance-of relations between individuals and concepts, and between pairs of individuals and roles.

Let $\mathcal{O}$ be an alphabet of symbols, called *individuals*. Instance-of relationships are expressed in terms of *membership assertions* of the form:

$$C(a), \qquad R(a, b)$$

where $a$ and $b$ are individuals, $C$ is a concept, and $R$ is a role. Intuitively, the first form states that $a$ is an instance of $C$, whereas the second form states that $a$ is related to $b$ by means of the role $R$.

In order to assign a meaning to membership assertions, the extension function $\cdot^{\mathcal{I}}$ of an interpretation $\mathcal{I}$ is extended to individuals by mapping them to elements of $\Delta^{\mathcal{I}}$ in such a way that $a^{\mathcal{I}} \neq b^{\mathcal{I}}$ if $a \neq b$ (Unique Name Assumption). An interpretation $\mathcal{I}$ *satisfies* the assertion $C(a)$ if $a^{\mathcal{I}} \in C^{\mathcal{I}}$, and *satisfies* $R(a, b)$ if $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$. An ABox is a finite set of membership assertions. $\mathcal{I}$ is a *model* for an ABox $\mathcal{A}$ if $\mathcal{I}$ satisfies all the assertions in $\mathcal{A}$.

An *$\mathcal{ALCNR}$-knowledge base* $\Sigma$ is a pair $\Sigma = \langle \mathcal{T}, \mathcal{A} \rangle$ where $\mathcal{T}$ is a TBox and $\mathcal{A}$ is an ABox. An interpretation $\mathcal{I}$ is a *model* for $\Sigma$ if it is both a model for $\mathcal{T}$ and a model for $\mathcal{A}$.

We can now formally define the problems 1–4 mentioned in the introduction. Given a KB $\Sigma$:

1. *Concept Satisfiability* : $C$ is *satisfiable* w.r.t $\Sigma$, if there exists a model $\mathcal{I}$ of $\Sigma$ such that $C^{\mathcal{I}} \neq \emptyset$;

2. *Subsumption* : $C$ is *subsumed* by $D$ w.r.t. $\Sigma$, if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\Sigma$;

3. *KB-satisfiability* : $\Sigma$ itself is *satisfiable*, if it has a model;

4. *Instance Checking* : $a$ is an instance of $C$, written $\Sigma \models C(a)$, if the assertion $C(a)$ is satisfied in every model of $\Sigma$.

In the sequel, we describe interpretations by giving only $\Delta^{\mathcal{I}}$, and the values of $\mathcal{I}$ on primitive concepts and primitive roles. It is straightforward to see that all values of $\mathcal{I}$ on complex concepts and roles are uniquely determined imposing that $\mathcal{I}$ must satisfy the equations given at the end of previous subsection.

**Example 2.1** Consider the following knowledge base $\Sigma = \langle \mathcal{T}, \mathcal{A} \rangle$:

$$\begin{aligned}
\mathcal{T} = \{&\exists \texttt{TEACHES.Course} \sqsubseteq (\texttt{Student} \sqcap \exists \texttt{DEGREE.BS}) \sqcup \texttt{Prof}, \\
&\texttt{Prof} \sqsubseteq \exists \texttt{DEGREE.MS}, \\
&\exists \texttt{DEGREE.MS} \sqsubseteq \exists \texttt{DEGREE.BS}, \\
&\texttt{MS} \sqcap \texttt{BS} \sqsubseteq -\} \\
\mathcal{A} = \{&\texttt{TEACHES}(\texttt{john}, \texttt{cs156}),\ (\leq 1\, \texttt{DEGREE})(\texttt{john}), \texttt{Course}(\texttt{cs156})\}
\end{aligned}$$

$\Sigma$ is a fragment of an hypothetical knowledge base describing the organization of a university. The first inclusion, for instance, states that the persons teaching a course are either graduate students (students with a BS degree) or professors. It is easy to see that $\Sigma$ is satisfiable. For example, the following interpretation $\mathcal{I}$ satisfies all the inclusions in $\mathcal{T}$ and all the assertions in $\mathcal{A}$, and therefore it is a model for $\Sigma$:

$$\begin{aligned}
&\Delta^{\mathcal{I}} = \{\texttt{john}, \texttt{cs156}, \texttt{csb}\},\ \texttt{john}^{\mathcal{I}} = \texttt{john},\ \texttt{cs156}^{\mathcal{I}} = \texttt{cs156} \\
&\texttt{Student}^{\mathcal{I}} = \{\texttt{john}\},\ \texttt{Prof}^{\mathcal{I}} = \emptyset,\ \texttt{Course}^{\mathcal{I}} = \{\texttt{cs156}\},\ \texttt{BS}^{\mathcal{I}} = \{\texttt{csb}\} \\
&\texttt{MS}^{\mathcal{I}} = \emptyset,\ \texttt{TEACHES}^{\mathcal{I}} = \{(\texttt{john}, \texttt{cs156})\},\ \texttt{DEGREE}^{\mathcal{I}} = \{(\texttt{john}, \texttt{csb})\}
\end{aligned}$$

Notice also that it is possible to draw several non-trivial conclusions from $\Sigma$. For example, we can infer that $\Sigma \models \texttt{Student}(\texttt{john})$. Intuitively this can be shown as follows: john teaches a course, thus he is either a student with a BS or a professor. But he can't be a professor since professors have at least two degrees (BS and MS) and he has at most one, therefore he is a student. □

We now show that, given the previous semantics, the problems 1–4 can all be reduced to KB-satisfiability (or to its complement) in linear time. In fact, given a KB $\Sigma = \langle \mathcal{T}, \mathcal{A} \rangle$, two concepts $C$ and $D$, an individual $a$, and an individual $b$ not appearing in $\Sigma$, the following relations hold:

$$\begin{aligned}
C \text{ is satisfiable w.r.t } \Sigma &\quad \text{iff} \quad \langle \mathcal{T}, \mathcal{A} \cup \{C(b)\} \rangle \text{ is satisfiable} \\
C \text{ is subsumed by } D \text{ w.r.t. } \Sigma &\quad \text{iff} \quad \langle \mathcal{T}, \mathcal{A} \cup \{(C \sqcap \neg D)(b)\} \rangle \text{ is not satisfiable} \\
\Sigma \models C(a) &\quad \text{iff} \quad \langle \mathcal{T}, \mathcal{A} \cup \{(\neg C)(a)\} \rangle \text{ is not satisfiable}
\end{aligned}$$

Consequently, we can concentrate just on KB-satisfiability in the next section.

# 3 Decidability Result

In this section we provide a calculus for deciding KB-satisfiability. In particular, in Subsection 3.1 we present the calculus and we state its correctness. Then, in Subsection 3.2, we prove the termination of the calculus. This will be sufficient to assess the decidability of all problems 1–4, thanks to the relationships between the four problems.

## 3.1 The calculus and its correctness

Our method makes use of the notion of *constraint system* [DLNN91a, SSS91, DLNS91], and is based on a tableau-like calculus [Fit90] that tries to build a model for the logical formula corresponding to a KB.

Consider an alphabet of variable symbols $\mathcal{V}$ (disjoint from the other alphabets defined so far). The elements of $\mathcal{V}$ are denoted by the letters $x, y, z, w$. In the sequel we use the term *object* as an abstraction for individual and variable (i.e. an object is an element of $\mathcal{O} \cup \mathcal{V}$). Objects are denoted by the symbols $s, t$ and, as in Section 2, individuals are denoted by $a, b$.

A *constraint* is a syntactic entity of one of the forms:

$$s{:}\,C, \qquad sPt, \qquad \forall x.x{:}\,C, \qquad s \not\doteq t,$$

where $C$ is a concept and $P$ is a primitive role. Concepts are assumed to be *simple*, i.e. the only complements they contain are of the form $\neg A$, where $A$ is a primitive concept. Arbitrary $\mathcal{ALCNR}$-concepts can be rewritten into equivalent simple concepts in linear time [DLNN91a]. A constraint system is a finite nonempty set of constraints.

Given an interpretation $\mathcal{I}$, we define an *$\mathcal{I}$-assignment* $\alpha$ as a function that maps every variable of $\mathcal{V}$ to an element of $\Delta^{\mathcal{I}}$, and every individual $a$ to $a^{\mathcal{I}}$ (i.e. $\alpha(a) = a^{\mathcal{I}}$ for all $a \in \mathcal{O}$).

A pair $(\mathcal{I}, \alpha)$ *satisfies* the constraint $s{:}\,C$ if $\alpha(s) \in C^{\mathcal{I}}$, the constraint $sPt$ if $(\alpha(s), \alpha(t)) \in P^{\mathcal{I}}$, the constraint $s \not\doteq t$ if $\alpha(s) \neq \alpha(t)$, and finally, the constraint $\forall x.x{:}\,C$ if $C^{\mathcal{I}} = \Delta^{\mathcal{I}}$ (notice that $\alpha$ does not play any role in this case). A constraint system $S$ is *satisfiable* if there is a pair $(\mathcal{I}, \alpha)$ that satisfies every constraint in $S$.

An $\mathcal{ALCNR}$-knowledge base $\Sigma = \langle \mathcal{T}, \mathcal{A} \rangle$ can be translated into a constraint system $S_\Sigma$ by replacing every inclusion $C \sqsubseteq D \in \mathcal{T}$ with the constraint $\forall x.x{:}\,\neg C \sqcup D$, every membership assertion $C(a)$ with the constraint $a{:}\,C$, every $R(a, b)$ with the constraints $aP_1b, \ldots, aP_kb$ if $R = P_1 \sqcap \ldots \sqcap P_k$, and including the constraint $a \not\doteq b$ for every pair $(a, b)$ of individuals appearing in $\mathcal{A}$. It is easy to see that $\Sigma$ is satisfiable if and only if $S_\Sigma$ is satisfiable.

In order to check a constraint system $S$ for satisfiability, our technique adds constraints to $S$ until either an evident contradiction is generated or an interpretation satisfying it can be obtained from the resulting system. Constraints are added on the basis of a suitable set of so-called *propagation rules*.

Before providing the rules, we need some additional definitions. Let $S$ be a constraint system and $R = P_1 \sqcap \ldots \sqcap P_k$ $(k \geq 1)$ be a role. We say that $t$ is an *$R$-successor of $s$ in $S$* if $sP_1t, \ldots, sP_kt$ are in $S$. We say that $t$ is a *1-successor of $s$ in $S$* if for some role $R$, $t$ is an $R$-successor of $s$. We call 1-predecessor the inverse relation of 1-successor. If $S$ is clear from the context we simply say that $t$ is an $R$-successor or a 1-successor of $s$ (or 1-predecessor). Moreover, we denote by *successor* the transitive closure of the relation 1-successor, and we denote by *predecessor* its inverse.

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

We say that $s$ and $t$ are *separated in $S$* if the constraint $s \neq\!\!\!\!\!\cdot\; t$ is in $S$.

Given a constraint system $S$ and an object $s$, we define the function $\sigma(\cdot,\cdot)$ as follows: $\sigma(S,s) := \{C \mid s{:}C \in S\}$. Moreover, we say that two variables $x$ and $y$ are *$S$-equivalent*, written $x \equiv_s y$, if $\sigma(S,x) = \sigma(S,y)$. Intuitively, two S-equivalent variables can represent the same element in the potential interpretation built by the rules, unless they are separated.

The *propagation rules* are:

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

   if 1. $s{:}C_1 \sqcap C_2$ is in $S$,
   2. $s{:}C_1$ and $s{:}C_2$ are not both in $S$

2. $S \rightarrow_{\sqcup} \{s{:}D\} \cup S$

   if 1. $s{:}C_1 \sqcup C_2$ is in $S$,
   2. neither $s{:}C_1$ nor $s{:}C_2$ is in $S$,
   3. $D = C_1$ or $D = C_2$

3. $S \rightarrow_{\forall} \{t{:}C\} \cup S$

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

4. $S \rightarrow_{\exists} \{sP_1y, \ldots, sP_ky,\ y{:}C\} \cup S$

if 1. $s{:}\exists R.C$ is in $S$,
2. $R = P_1 \sqcap \ldots \sqcap P_k$,
3. $y$ is a new variable,
4. there is no $t$ such that $t$ is an $R$-successor of $s$ in $S$ and $t{:}C$ is in $S$,
5. if $s$ is a variable there is no variable $w$ in $S$ such that $w$ is a predecessor of $s$ and $s \equiv_s w$

5. $S \rightarrow_{\geq} \{sP_1y_i, \ldots, sP_ky_i \mid i \in 1..n\} \cup \{y_i \neq y_j \mid i,j \in 1..n, i \neq j\} \cup S$

if 1. $s{:}(\geq n\, R)$ is in $S$,
2. $R = P_1 \sqcap \ldots \sqcap P_k$,
3. $y_1, \ldots, y_n$ are new variables,
4. there do not exist $n$ pairwise separated $R$-successors of $s$ in $S$,
5. if $s$ is a variable there is no variable $w$ such that $w$ is a predecessor of $s$ and $s \equiv_s w$

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

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

7. $S \rightarrow_{\forall x} \{s{:}C\} \cup S$

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

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

The use of the condition based on the $S$-equivalence relation in the generating rules (condition 5) is related to the goal of keeping the constraint system finite even in presence of potentially infinite chains of applications of generating rules. Its role will become clearer in the sequel.

One can verify that rules are always applied to a system $S$ either because of the presence in $S$ of a given constraint $s{:}C$ (condition 1), or, in the case of the $\rightarrow_{\forall x}$-rule, because of the presence of an object $s$ in $S$. When no confusion arises, we will say that a rule is *applied to* the object $s$ (instead of saying that it is applied to the constraint system $S$).

**Proposition 3.1 (Invariance)** *Let $S$ and $S'$ be constraint systems. Then:*

1. *If $S'$ is obtained from $S$ by application of a deterministic rule, then $S$ is satisfiable if and only if $S'$ is satisfiable.*
2. *If $S'$ is obtained from $S$ by application of a nondeterministic rule, then $S$ is satisfiable if $S'$ is satisfiable. Furthermore, if a nondeterministic rule applies to $S$, then it can be applied in such a way that it yields a constraint system $S'$ which is satisfiable if and only if $S$ is satisfiable.*

Given a constraint system, more than one rule might be applicable to it. We define the following *strategy* for the application of rules:

1. apply a rule to a variable only if no rule is applicable to individuals;
2. apply generating rules only if no nongenerating rule is applicable;
3. apply a generating rule to a variable $x$ only if no rule is applicable to a predecessor of $x$.

Notice that a constraint system $S$ can be seen as a directed graph with the objects in $S$ as nodes and an arc from $s$ to $t$, if there is a constraint $sPt$ in $S$. If a constraint system is derived from an $\mathcal{ALCNR}$-knowledge base $\Sigma$ then the corresponding graph has some particular properties. Namely, every variable has a single 1-predecessor and, for each variable, the subgraph composed by itself and its successors is always a tree. We refer to this property in what follows as the *tree structure* property.

In the sequel, we assume that rules are always applied according to this strategy and that we always start with a constraint system $S_\Sigma$ coming from an $\mathcal{ALCNR}$-knowledge base $\Sigma$. The following lemma is a direct consequence of these assumptions.

**Lemma 3.2 (Stability)** *Let $S$ be a constraint system and $x$ be a variable in $S$. Let a generating rule be applicable to $x$ according to the strategy. Let $S'$ be any constraint system derivable from $S$ by any sequence (possibly empty) of applications of rules. Then*

*1) A variable $y$ is a predecessor of $x$ in $S$ iff it is a predecessor of $x$ in $S'$*

*2) No rule is applicable to a predecessor $y$ of $x$ in $S'$*

*3) $\sigma(S, x) = \sigma(S', x)$*

*Proof.* (Sketch)

1) A case analysis considering all rules shows that no predecessors are added or discarded.

2) By contradiction: Suppose $S \equiv S_0 \rightarrow_* S_1 \rightarrow_* \cdots \rightarrow_* S_n \equiv S'$, where $* \in \{\sqcup, \sqcap, \exists, \forall, \geq, \leq, \forall x\}$ and a rule is applicable to a predecessor $y$ of $x$ in $S'$. Then there exists a minimal $i$, $i \leq n$, such that this is the case in $S_i$. Note that $i \neq 0$ because of the strategy. So no rule is applicable to any predecessor of $x$ in $S_0, \ldots, S_{i-1}$. By an exhaustive analysis of all rules we see that—whichever is the rule applied from $S_{i-1}$ to $S_i$—no rule is applicable to any predecessor $y$ of $x$ in $S_i$, contradicting the assumption.

3) By contradiction: Suppose $\sigma(S, x) \neq \sigma(S', x)$. Then a rule must have been applied to the direct predecessor of $x$ or to $x$ itself. The former cannot be because of 2). A case analysis shows that the only rules which can have been applied to $x$ are generating ones and the $\rightarrow_\forall$ and the $\rightarrow_\leq$ rules. But these rules add new constraints only to the successors of $x$ and not to $x$ itself and therefore do not change $\sigma(\cdot, x)$ □

In particular Lemma 3.2 proves that for a variable $x$ which has a successor, $\sigma(\cdot, x)$ is stable, i.e. it will not change because of subsequent applications of rules.

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

- $\{s\!:\!-\}$
- $\{s\!:\!A,\ s\!:\!\neg A\}$, where $A$ is a primitive concept.
- $\{s\!:\!(\leq n\, R)\} \cup \{sP_1t_i, \ldots, sP_kt_i \mid i \in 1..n+1\}$
  $\cup \{t_i \not\equiv t_j \mid i, j \in 1..n+1, i \neq j\}$,
  where $R = P_1 \sqcap \ldots \sqcap P_k$.

A clash is evidently an unsatisfiable constraint system. Therefore, any constraint system containing a clash is unsatisfiable. The purpose of the calculus is to generate completions, and look for the presence of clashes inside. If a completion $S$ contains no clash, we prove that it is always possible to generate a model for $\Sigma$ on the basis of $S$. Before looking at the technical details of the proof, let us consider an example of application of the calculus for checking satisfiability.

**Example 3.3** Consider the following knowledge base $\Sigma = \langle \mathcal{T}, \mathcal{A} \rangle$:

$$\mathcal{T} = \{\texttt{Italian} \sqsubseteq \exists \texttt{FRIEND.Italian}\}$$

$$\begin{aligned}\mathcal{A} = \{&\texttt{FRIEND}(\texttt{peter}, \texttt{susan}),\\ &\forall \texttt{FRIEND}.\neg \texttt{Italian}(\texttt{peter}),\\ &\exists \texttt{FRIEND.Italian}(\texttt{susan})\}\end{aligned}$$

The corresponding constraint system $S_\Sigma$ is:

$$\begin{aligned}S_\Sigma = \{&\forall x.x\colon \neg \texttt{Italian} \sqcup \exists \texttt{FRIEND.Italian},\\ &\texttt{peterFRIENDsusan},\\ &\texttt{peter}\colon \forall \texttt{FRIEND}.\neg \texttt{Italian},\\ &\texttt{susan}\colon \exists \texttt{FRIEND.Italian}\\ &\texttt{peter} \not\doteq \texttt{susan}\}\end{aligned}$$

A sequence of applications of the propagation rules to $S_\Sigma$ is as follows:

$S_1 = S_\Sigma \cup \{\texttt{susan}\colon \neg \texttt{Italian}\}$ ($\rightarrow_\forall$-rule)
$S_2 = S_1 \cup \{\texttt{peter}\colon \neg \texttt{Italian} \sqcup \exists \texttt{FRIEND.Italian}\}$ ($\rightarrow_{\forall x}$-rule)
$S_3 = S_2 \cup \{\texttt{susan}\colon \neg \texttt{Italian} \sqcup \exists \texttt{FRIEND.Italian}\}$ ($\rightarrow_{\forall x}$-rule)
$S_4 = S_3 \cup \{\texttt{peter}\colon \neg \texttt{Italian}\}$ ($\rightarrow_\sqcup$-rule)
$S_5 = S_4 \cup \{\texttt{susanFRIENDx}, \texttt{x}\colon \texttt{Italian}\}$ ($\rightarrow_\exists$-rule)
$S_6 = S_5 \cup \{\texttt{x}\colon \neg \texttt{Italian} \sqcup \exists \texttt{FRIEND.Italian}\}$ ($\rightarrow_{\forall x}$-rule)
$S_7 = S_6 \cup \{\texttt{x}\colon \exists \texttt{FRIEND.Italian}\}$ ($\rightarrow_\sqcup$-rule)
$S_8 = S_7 \cup \{\texttt{xFRIENDy}, \texttt{y}\colon \texttt{Italian}\}$ ($\rightarrow_\exists$-rule)
$S_9 = S_8 \cup \{\texttt{y}\colon \neg \texttt{Italian} \sqcup \exists \texttt{FRIEND.Italian}\}$ ($\rightarrow_{\forall x}$-rule)
$S_{10} = S_9 \cup \{\texttt{y}\colon \exists \texttt{FRIEND.Italian}\}$ ($\rightarrow_\sqcup$-rule)

One can verify that $S_{10}$ is a complete clash-free constraint system. In particular, $\rightarrow_\exists$-rule is not applicable to $y$. In fact, since $x \equiv_{S_{10}} y$ condition 5 is not satisfied. From $S_{10}$ one can build an interpretation $\mathcal{I}$, as follows:

$\Delta^\mathcal{I} = \{\texttt{peter}, \texttt{susan}, \texttt{x}, \texttt{y}\}$
$\texttt{peter}^\mathcal{I} = \texttt{peter}$, $\texttt{susan}^\mathcal{I} = \texttt{susan}$, $\alpha(\texttt{x}) = \texttt{x}$, $\alpha(\texttt{y}) = \texttt{y}$,
$\texttt{Italian}^\mathcal{I} = \{\texttt{x}, \texttt{y}\}$
$\texttt{FRIEND}^\mathcal{I} = \{(\texttt{peter}, \texttt{susan}), (\texttt{susan}, \texttt{x}), (\texttt{x}, \texttt{y}), (\texttt{y}, \texttt{y})\}$

It is easy to see that $\mathcal{I}$ is indeed a model for $\Sigma$. □

In order to prove that it is always possible to obtain an interpretation from a complete constraint system we need some additional notions. Let $S$ be a constraint system and $x$, $w$ be variables in $S$. We call $w$ a *witness of* $x$ *in* $S$ if the three following conditions hold:

1. $x \equiv_s w$

2. $w$ is a predecessor of $x$ in $S$

3. no rule is applicable to any predecessor of $x$.

Notice that the third condition ensures that no new constraint will be imposed on $x$. We say $x$ *is blocked* (*by* $w$) if $x$ has a witness ($w$) in $S$.

The following lemma states some properties of witnesses.

**Lemma 3.4** *Let $S$ be a constraint system, $x$ a variable in $S$. If $x$ has a witness then (i) $x$ has no successor and (ii) $x$ has exactly one witness.*

*Proof.* (i) By contradiction: Suppose that $x$ is blocked in $S$ and $xPy$ is in $S$. During the completion process leading to $S$ a generating rule must have been applied to $x$ in a system $S'$. It follows from the definition of the rules, that in $S'$ for every predecessor $w$ of $x$ we had $x \not\equiv_{s'} w$. Now from Lemma 3.2 we know, that for the constraint system $S$ derivable from $S'$ and for every predecessor $w$ of $x$ in $S$ we also had $x \not\equiv_s w$. Hence there is no witness for $x$ in $S$, contradicting the hypothesis that $x$ is blocked.

(ii) By contradiction: Assume there are two witnesses $w_1$ and $w_2$ of $x$ in $S$, then $w_1 \equiv_s x \equiv_s w_2$. Then one must be the predecessor of the other because no variable has more than one 1-predecessor, i.e. because of the tree structure property. Let's say $w_1$ is the predecessor of $w_2$. Because no rule is applicable to $w_1$, $w_2$ is blocked by $w_1$. Then we have a successor ($x$) for a blocked variable ($w_2$). This contradicts (i). □

As a consequence of Lemma 3.4, in a constraint system $S$, if $w_1$ is a witness of $x$ then $w_1$ cannot have a witness itself, since both the property of being a predecessor and of $S$-equivalence are transitive. The uniqueness of witness for a blocked variable is important for defining the following particular interpretation out of $S$.

Let $S$ be a constraint system. We define the *canonical interpretation* $\mathcal{I}_S$ and the *canonical* $\mathcal{I}_S$*-assignment* $\alpha_S$ as follows:

1. $\Delta^{\mathcal{I}_S} := \{s \mid s \text{ is an object in } S\}$

2. $\alpha_S(s) := s$

3. $s \in A^{\mathcal{I}_S}$ iff $s\colon A$ is in $S$

4. $(s,t) \in P^{\mathcal{I}_S}$ iff

   (a) $sPt$ is in $S$ or

   (b) $s$ is a blocked variable, $w$ is the witness of $s$ in $S$ and $wPt$ is in $S$.

We call $(s,t)$ a *P-role-pair* of $s$ in $\mathcal{I}_S$ if $(s,t) \in P^{\mathcal{I}_S}$, we call $(s,t)$ a *role-pair* of $s$ in $\mathcal{I}_S$ if $(s,t)$ is a *P-role-pair* for a role $P$. We call a role-pair *explicit* if it comes up from case 4.(a) of the definition of the canonical interpretation and we call it *implicit* if it comes up from case 4.(b).

From Lemma 3.4 it is obvious that a role-pair cannot be both explicit and implicit. Moreover, if a variable has an implicit role-pair then all its role-pairs are implicit and they all come from exactly one witness, as stated by the following lemma.

**Lemma 3.5** *Let $S$ be a completion and $x$ a variable in $S$. Let $\mathcal{I}_S$ be the canonical interpretation for $S$. If $x$ has an implicit role-pair $(x,y)$, then*

1. *all role-pairs of $x$ in $\mathcal{I}_S$ are implicit*
2. *there is exactly one witness $w$ of $x$ in $S$ such that for all roles $P$ in $S$ and all $P$-role-pairs $(x,y)$ of $x$ the constraint $wPy$ is in $S$.*

*Proof.* Point 1 follows from (i) of Lemma 3.4 and point 2 follows from (ii) of Lemma 3.4 together with the definition of $\mathcal{I}_S$. □

We have now all the machinery needed to prove the main theorem of this subsection.

**Theorem 3.6 (Correctness)** *Let $S$ be a complete constraint system. $S$ is satisfiable iff it contains no clash.*

*Proof.*
"⇒" Clearly, a system containing a clash is unsatisfiable. Hence, $S$ is satisfiable only if it contains no clash.
"⇐" Suppose $S$ contains no clash. Let $\mathcal{I}_S$ and $\alpha_S$ be the canonical interpretation and $\mathcal{I}$-assignment for $S$. We prove that the pair $(\mathcal{I}_S, \alpha_S)$ satisfies every constraint $c$ in $S$. If $c$ has the form $sPt$ or $s \neq t$, then $(\mathcal{I}_S, \alpha_S)$ satisfies them by definition of $\mathcal{I}_S$ and $\alpha_S$ . If $c$ has the form $s\!:\!C$, we show by induction on the structure of $C$ that $s \in C^{\mathcal{I}_S}$.
*Base case*: If $C$ is a primitive concept, then $s \in C^{\mathcal{I}_S}$ by definition of $\mathcal{I}_S$. If $C = \top$, then obviously $s \in \top^{\mathcal{I}_S}$. Finally, if $C = -$, then $s\!:\!-$ cannot occur in $S$, since $S$ is clash-free.
*Induction step*: we analyze in turn each possible form of the concept $C$.

$\neg A$) if $s\!:\!\neg A$ is in $S$ then $A$ is a primitive concept since all concepts are simple. Then the constraint $s\!:\!A$ is not in $S$ since $S$ is clash-free. Then $s \notin A^{\mathcal{I}_S}$, that is, $s \in \Delta^{\mathcal{I}_S} \setminus A^{\mathcal{I}_S}$. Hence $s \in (\neg A)^{\mathcal{I}_S}$.

$C_1 \sqcap C_2$) if $s\!:\!C_1 \sqcap C_2 \in S$ then (since $S$ is complete) $s\!:\!C_1$ is in $S$ and $s\!:\!C_2$ is in $S$. By induction hypothesis, $s \in C_1^{\mathcal{I}_S}$ and $s \in C_2^{\mathcal{I}_S}$. Hence $s \in (C_1 \sqcap C_2)^{\mathcal{I}_S}$.

$C_1 \sqcup C_2$) Similar to the previous case.

$\forall R.D$) We have to show that for all $t$ with $(s,t) \in R^{\mathcal{I}_S}$ it holds that $t \in D^{\mathcal{I}_S}$. If $(s,t) \in R^{\mathcal{I}_S}$, then according to Lemma 3.5 one of the following two cases must occur:

1. $t$ is an $R$-successor of $s$ in $S$. Since $S$ is complete, $t\!:\!D$ must also be in $S$. Then by induction hypothesis we have $t \in D^{\mathcal{I}_S}$.
2. $s$ is blocked by a witness $w$ in $S$ and $t$ is an $R$-successor of $w$ in $S$. Then by definition of witness, $w\!:\!\forall R.D$ is in $S$ and then because of completeness of $S$, $t\!:\!D$ must be in $S$. By induction hypothesis we have again $t \in D^{\mathcal{I}_S}$.

$\exists R.D$) We have to show that there exists a $t \in \Delta^{\mathcal{I}_S}$ with $(s,t) \in R^{\mathcal{I}_S}$ and $t \in D^{\mathcal{I}_S}$. Since $S$ is complete one of following two cases must occur:

1. There is a $t$ that is an $R$-successor of $s$ in $S$ and $t\!:\!D$ is in $S$. Then by induction hypothesis and the definition of $\mathcal{I}_S$ we have $t \in D^{\mathcal{I}_S}$ and $(s,t) \in R^{\mathcal{I}_S}$.
2. $s$ is a variable blocked by a witness $w$ in $S$. Hence $w\!:\!\exists R.D$ is in $S$. Since $w$ cannot be blocked and $S$ is complete, we have that there is a $t$ that is an $R$-successor of $w$ in $S$ and $t\!:\!D$ is in $S$. So by induction hypothesis we have $t \in D^{\mathcal{I}_S}$ and by the definition of $\mathcal{I}_S$ we have $(s,t) \in R^{\mathcal{I}_S}$.

$(\leq n\, R)$) By contradiction: Assume that $s \notin (\leq n\, R)^{\mathcal{I}_S}$. Then there exist atleast $n+1$ distinct objects $t_1, \ldots, t_{n+1}$ with $(s,t_i) \in R^{\mathcal{I}_S},\ i \in 1..n+1$. This means that, since $R = P_1 \sqcap \ldots \sqcap P_k$ there are pairs $(s,t_i) \in P_j^{\mathcal{I}_S}$, where $i \in 1..n+1,\ j \in 1..k$. Then according to Lemma 3.5 one of the two following cases must occur:

1. All $sP_jt_i$ for $j \in 1..k,\ i \in 1..n+1$ are in $S$. Because of completeness the $\rightarrow_{\leq}$ -rule is not applicable. This means that all the $t_i$'s are pairwise separated, i.e. that $S$ contains the constraints $t_i \not\doteq t_j,\ i,j \in 1..n+1, i \neq j$. This contradicts the fact that $S$ is clash-free.
2. There exists a witness $w$ of $s$ in $S$ with all $wP_it_i$ for $j \in 1..k,\ i \in 1..n+1$ are in $S$. But this leads to the same contradiction.

$(\geq n\, R)$) By contradiction. Assume that $s \notin (\geq n\, R)^{\mathcal{I}_S}$. Then there exist atmost $m < n$ ($m$ possibly 0) distinct objects $t_1, \ldots, t_m$ with $(s,t_i) \in R^{\mathcal{I}_S},\ i \in 1..m$. We have to consider two cases:

1. $s$ is not blocked in $S$. Since there are only $m$ $R$-successors of $s$ in $S$, the $\rightarrow_{\geq}$ -rule is applicable to $s$. This contradicts the fact that $S$ is complete.
2. $s$ is blocked by a witness $w$ in $S$. Since there are $m$ $R$-successors of $w$ in $S$, the $\rightarrow_{\geq}$ -rule is applicable to $w$. But this leads to the same contradiction.

If $c$ has the form $\forall x.x\colon D$ then, since $S$ is complete, for each object $t$ in $S$, $t\colon D$ is in $S$—and, by the previous cases, $t \in D^{\mathcal{I}_S}$. Therefore, the pair $(\mathcal{I}_S, \alpha_S)$ satisfies $\forall x.x\colon D$. Finally, since $(\mathcal{I}_S, \alpha_S)$ satisfies all constraints in $S$, $(\mathcal{I}_S, \alpha_S)$ satisfies $S$. □

## 3.2 Termination of the calculus

**Lemma 3.7** *Let $S$ be a constraint system, let $n$ be the number of concepts appearing in $S$ (including subconcepts), and let $S'$ be derived from $S$ by means of the propagation rules. If in $S'$ there are more than $2^n$ variables, then there are at least two variables $x$,$y$ such that $x \equiv_{s'} y$.*

*Proof.* Each constraint $x\colon C \in S'$ may contain only concepts of the constraint system $S$. Since there are $n$ such concepts, given a variable $x$ there cannot be more than $2^n$ different sets of constraints $x\colon C$ in $S'$. □

**Lemma 3.8** *Let $S$ be a constraint system, let $n$ be the number of concepts in $S$, and let $S'$ be any constraint system derived from $S$ by applying the propagation rules with the given strategy. Then, every variable in $S'$ can have at most $2^n$ predecessor variables.*

*Proof.* Suppose there is a variable $x$ having at least $2^n + 1$ predecessors. From Lemma 3.7, we know that in the set of variables constituted by all predecessors of $x$ there are at least two variables $y_1$, $y_2$ such that $y_1 \equiv_s y_2$, and because of the tree structure property of variables one variable is a predecessor of the other—say, $y_1$ is a predecessor of $y_2$. From the definitions of witness and blocked we know that $y_2$ is blocked and $y_1$ is its witness. Hence, from Lemma 3.4, $y_2$ cannot have any successor, contradicting the hypothesis that $x$ was a successor of $y_2$. □

**Theorem 3.9 (Termination)** *Let $S$ be a constraint system. Every completion of $S$ is finite.*

*Proof.* This follows from Lemma 3.8 and the tree structure of constraint systems. □

We come now to the main result of this section.

**Theorem 3.10 (Decidability)** *Given an $\mathcal{ALCNR}$-knowledge base $\Sigma$, checking whether $\Sigma$ is satisfiable is a decidable problem.*

*Proof.* This follows from Theorems 3.6 and 3.9 and the fact that $\Sigma$ is satisfiable if and only if $S_\Sigma$ is satisfiable. □

Notice that, since the domain of the canonical interpretation $\Delta^{\mathcal{I}_S}$ is always finite, we have also implicitly proved that $\mathcal{ALCNR}$-knowledge bases have the *finite model property*, i.e. any satisfiable knowledge base has a finite model. This property has been extensively studied in modal logics [HC84] and dynamic logics [Har84]. In particular a technique, called *filtration*, has been developed both to prove the finite model property and to build a finite model for a satisfiable formula. This technique allows one to build a finite model from an infinite one by grouping the worlds of a structure in equivalence classes based on the set of formulae that are satisfied in each world. It is interesting to observe that our calculus, based on witnesses, can be considered as a variant of the filtration technique where the equivalence classes are determined on the basis of our $S$-equivalence relation. However, because of number restrictions, variables that are $S$-equivalent cannot be grouped, since they might be separated (e.g. they might have been introduced by the same application of the $\rightarrow_{\geq}$-rule). Nevertheless, they can have the same successors, as stated in point 4.(b) of the definition of canonical interpretation. This would correspond to grouping variables of an infinite model in such a way that separations are preserved.

# 4 A Calculus Working in Exponential Space

The calculus proposed in the previous section requires to compute all the completions of the constraint system $S_\Sigma$. Unfortunately, such completions may be of double exponential size w.r.t. the size of $\Sigma$. This can be seen by considering the tree structure of variables in $S$: each branch may have exponential size, and there can be an exponential number of branches

For an exponential space algorithm it is therefore crucial not to keep an entire complete constraint system in the memory but to store only small portions at a time. Let's make this idea more precise.

We give propagation rules, called *trace rules*, that build up only a portion of complete constraint systems.

The *trace rules* consist of the $\rightarrow_{\sqcap}$-, $\rightarrow_{\sqcup}$-, $\rightarrow_{\forall}$-, $\rightarrow_{\forall x}$- and the $\rightarrow_{\leq}$-rule (the nongenerating rules) together with the following two generating rules that replace the $\rightarrow_{\exists}$- and the $\rightarrow_{\geq}$-rule and are obtained from them by adding an further condition (n. 6):

4′. $S \rightarrow_{T\exists} \{sP_1y, \ldots, sP_ky,\ y{:}C\} \cup S$

if 1. $s{:}\exists R.C$ is in $S$,
2. $R = P_1 \sqcap \ldots \sqcap P_k$,
3. $y$ is a new variable,
4. there is no $t$ such that $t$ is an $R$-successor of $s$ in $S$ and $t{:}C$ is in $S$
5. if $s$ is a variable, then there is no variable $w$ in $S$ such that $w$ is a predecessor of $s$ and $s \equiv_s w$,
6. for all constraints $tPx$ in $S$, $t$ is a predecessor of $s$ or $s = t$

5′. $S \rightarrow_{T\geq} \{sP_1y_i, \ldots, sP_ky_i \mid i \in 1..n\} \cup \{y_i \neq y_j, \mid i, j \in 1..n, i \neq j\} \cup S$

if 1. $s{:}(\geq n\ R)$ is in $S$,
2. $R = P_1 \sqcap \ldots \sqcap P_k$,
3. $y_1, \ldots, y_n$ are new variables,
4. there do not exist $n$ pairwise separated $R$-successors of $s$ in $S$,
5. if $s$ is a variable, then there is no variable $w$ such that $w$ is a predecessor of $s$ and $s \equiv_s w$,
6. for all constraints $tPx$ in $S$, $t$ is a predecessor of $s$ or $s = t$

Let $T$ be a constraint system obtained from $S_\Sigma$ by application of the trace rules. We call $T$ a *trace* of $S_\Sigma$ if no trace rule applies to $T$.

If the trace rules are applied according to the strategy, they exhibit the following behavior (see Figure 1): Given an object $s$, if at least one generating rule is applicable, all its 1-successors $y_1, \ldots, y_n$ are introduced. Then, after nongenerating rules are applied, one variable $y_i$ is (nondeterministically) chosen, and all 1-successors of $y_i$ are introduced. Unlike normal propagation rules, no successor is introduced for any object different from $y_i$. Then, one variable is chosen among the 1-successors of $y_i$, only its 1-successors are added to the constraint system, and so on.

The reason why we introduce all the 1-successors of the "chosen" object is the following. For every chosen object $s$ all 1-successors of $s$ must be present simultaneously at some stage of the computation, since only the interplay of role conjunction and number restrictions forces us to identify certain successors. This is important because, when identifying variables,

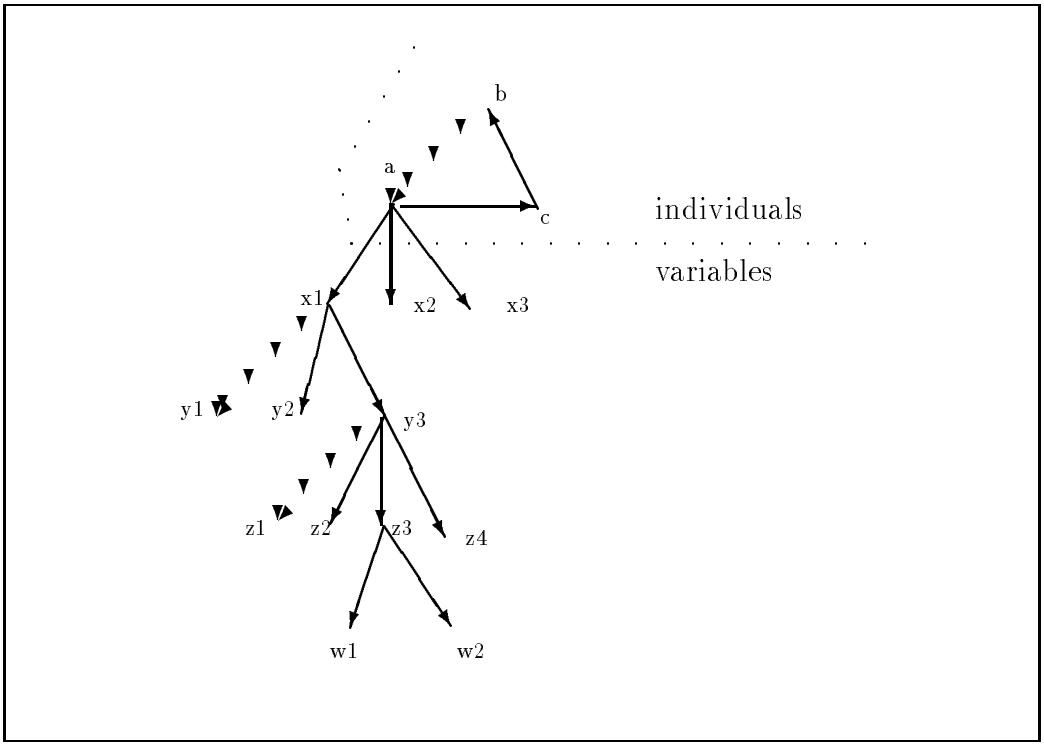


Figure 1: A trace

the constraints imposed on them are combined, which may lead to clashes that otherwise would not have occurred.

Trace rules have been defined in [DLNN91a] for deciding satisfiability and pure subsumption between $\mathcal{ALCNR}$-concepts. Algorithms that generate all complete constraint systems derivable from a constraint system while keeping only one trace in memory at a time have been given (for sublanguages of $\mathcal{ALCNR}$) in [SSS91, HN90, HNSS90].

**Proposition 4.1** *Let $\Sigma$ be an $\mathcal{ALCNR}$-knowledge base, $S_\Sigma$ its associated constraint system, and let $n$ be the size of $\Sigma$. Then:*

1. *The length of a trace rule derivation issuing from $S_\Sigma$ is bounded by $2^n$.*
2. *Every complete constraint system extending $S_\Sigma$ can be obtained as the union of finitely many traces.*
3. *Suppose $S$ is a complete constraint system extending $S_\Sigma$ and $\mathcal{T}$ is a finite set of traces such that $S = \bigcup_{T \in \mathcal{T}} T$. Then $S$ contains a clash if and only if some $T \in \mathcal{T}$ contains a clash.*

*Proof.* (sketch) For part (1), reminding that the number of concepts (including subconcepts) appearing in $\Sigma$ is bounded by $n$, the claim follows

from Lemma 3.8. Part (2) is obvious considering the tree structure of a constraint system. Part (3) is obvious considering the definition of clash and trace. □

Part (3) of the above proposition says that to detect clashes in a constraint system $S$, it suffices to inspect the traces $S$ is formed by, one trace at a time.

Since an $\mathcal{ALCNR}$-knowledge base $\Sigma$ is satisfiable if and only if there exists a complete constraint system derivable from $S_\Sigma$ without a clash, it follows from Proposition 4.1 that satisfiability of an $\mathcal{ALCNR}$-knowledge base can be decided nondeterministically with exponential space. A possible algorithm using space $2^{p(n)}$ where $p(n)$ is a polynomial in the size of $S_\Sigma$ may be the following: compute one complete constraint system, one trace at a time, guessing (in the application of the nondeterministic rules $\rightarrow_{\leq}$, $\rightarrow_{\sqcup}$) the choices leading to traces without a clash.

Then from Savitch's theorem (see e.g. [HU79, Theorem 12.11]) it is well known that

$$\text{NSPACE}\,(2^{p(n)}) \subseteq \text{DSPACE}\,(2^{2 \cdot p(n)}).$$

Hence we can conclude with the main result of this section.

**Theorem 4.2** *Satisfiability of an $\mathcal{ALCNR}$-knowledge base $\Sigma$ can be decided with exponential space.*

A lower bound of the complexity of KB-satisfiability in obtained exploiting previous results about the language $\mathcal{ALC}$, which is a sublanguage of $\mathcal{ALCNR}$ that does not include number restrictions and role conjunction. We know from McAllester [McA92], and (independently) from an observation of Werner Nutt [Nut92] that KB-satisfiability in $\mathcal{ALC}$-knowledge bases is EXPTIME-hard (and hence it is hard for $\mathcal{ALCNR}$-knowledge bases, too). Hence, we do not expect to find any algorithm solving the problem in polynomial space, unless PSPACE=EXPTIME. Nevertheless, the algorithm outlined above may require double exponential time. It is still open whether there exists an algorithm working in (simple) exponential time.

# 5 Relation to previous work

In this section, we discuss the relation of our paper to previous work about reasoning with inclusions. In particular, we first consider previously proposed reasoning techniques that deal with inclusions and terminological cycles, then we discuss the relation between inclusions and terminological cycles.

## 5.1 Reasoning Techniques

As mentioned in the introduction, there is work done by Baader et. al. [BBH+90], Baader [Baa90b, Baa90a], Nebel [Neb90b, Neb91] and Schild [Sch91].

In [Neb90b] the language $\mathcal{TF}$, containing concept conjunction, universal quantification and number restrictions, and TBoxes containing (possibly cyclic) concept definitions, role definitions and disjointness axioms (stating that two primitive concepts are disjoint) are considered. Nebel shows that subsumption of $\mathcal{TF}$-concepts w.r.t. TBoxes is decidable. However, the argument he uses is not constructive. He shows that it is sufficient to consider finite interpretations of a size bounded by the size of the TBox in order to decide subsumption.

In [Baa90b] the effect of the three types of semantics—descriptive, greatest fixed-point and least-fixed-point semantics—for the language $\mathcal{FL}_0$, containing concept conjunction and universal quantification, is described with the help of *finite automata*. Baader reduces subsumption of $\mathcal{FL}_0$-concepts w.r.t. TBoxes containing (possibly cyclic) equivalences of the form $A \doteq C$ (which he calls terminological axioms) to decision problems for finite automata. In particular, he shows that subsumption w.r.t. descriptive semantics can be decided in polynomial space using *Büchi automata*.

Using results from [Baa90b], in [Neb91] a characterization of the above subsumption problem w.r.t. descriptive semantics is given with the help of deterministic automata (whereas Büchi automata are nondeterministic). This also yields a PSPACE-algorithm for deciding subsumption.

In [BBH+90] the attention is restricted to the language $\mathcal{ALC}$. In particular, that paper considers the problem of checking the satisfiability of a single equation of the form $C = \top$, where $C$ is an $\mathcal{ALC}$-concept. This problem, called the *universal satisfiability problem*, is shown to be equivalent to checking the satisfiability of an $\mathcal{ALC}$-TBox (see Proposition 5.1 in the sequel).

In [Baa90a], an extension of $\mathcal{ALC}$, called $\mathcal{ALC}_{reg}$, is introduced, which supports a constructor to express the transitive closure of roles. By means of transitive closure of roles it is possible to replace cyclic definitions of the form $A \sqsubseteq D$ with equivalent acyclic ones. The problem of checking the satisfiability of an $\mathcal{ALC}_{reg}$-concept is solved in that paper. It is also shown that using transitive closure it is possible to reduce satisfiability of an $\mathcal{ALC}$-concept w.r.t. an $\mathcal{ALC}$-TBox $\mathcal{T} = \{C_1 \sqsubseteq D_1, \ldots, C_n \sqsubseteq D_n\}$ into the concept satisfiability problem in $\mathcal{ALC}_{reg}$ (w.r.t. the empty TBox). Since the problem of concept satisfiability w.r.t. a TBox is trivially harder than checking the satisfiability of a TBox, this paper extends the result given in [BBH+90].

The technique exploited in [BBH+90] and [Baa90a] is based on the notion

of *concept tree*. A concept tree is generated starting from a concept $C$ in order to check its satisfiability (or universal satisfiability). The way a concept tree is generated from a concept $C$ is similar in flavor to the way a complete constraint system is generated from the constraint system $\{x{:}C\}$. However, the extension of the concept tree method to deal with number restrictions and individuals in the knowledge base is neither obvious, nor suggested in the cited papers; on the other hand, the extension of the calculus based on constraint systems is immediate, provided that additional features have a counterpart in First Order Logic.

In [Sch91] some results more general than those in [Baa90a] are obtained by considering languages more expressive than $\mathcal{ALC}_{reg}$ and dealing with the concept satisfiability problem in such languages.

The results in [Sch91] are obtained by establishing a correspondence between concept languages and Propositional Dynamic Logics (PDL), and reducing the given problem to a satisfiability problem in PDL. Such an approach allows Schild to find several new results exploiting known results in the PDL framework. However, it cannot be used to deal with every concept language. In fact, the correspondence cannot be established when the language includes some concept constructors having no counterpart in PDL (e.g. number restrictions, or individuals in an ABox).

In conclusion, all these approaches, i.e. reduction to automata problems, concept trees and reduction to PDL, deal only with TBoxes and they don't seem to be suitable to deal also with ABoxes. On the other hand, the constraint system technique, even though it was conceived for TBox-reasoning, can be easily extended to ABox-reasoning, as also shown in [Hol90], [BH91] and [DLNS92].

## 5.2 Inclusions versus Concept Definitions

Now we compare the expressive power of TBoxes defined as a set of inclusions (as done in this paper) and TBoxes defined as a set of (possibly cyclic) concept definitions of the form $A \doteq D$ and $A \doteq D$.

Unlike [Baa90a] and [Sch91], we consider reasoning problems dealing with TBox and ABox together. Moreover, we use the descriptive semantics for the concept definitions, as we do for the inclusions. The result we have obtained is that inclusion statements and concept definitions actually have the same expressive power. In details, we show that the satisfiability of a knowledge base $\Sigma = \langle \mathcal{A}, \mathcal{T} \rangle$, where $\mathcal{T}$ is a set of inclusion statements can be reduced to the satisfiability of a knowledge base $\Sigma' = \langle \mathcal{A}', \mathcal{T}' \rangle$ such that $\mathcal{T}'$ is a set of concept definitions. The other direction, from concept definitions to inclusions, is trivial since definitions of the form $A \doteq D$ can be expressed by

the pair of inclusions $A \sqsubseteq D$ and $D \sqsubseteq A$, while a primitive concept definition $A \dot{\leq} D$ can be rewritten as the inclusion $A \sqsubseteq D$ (as already mentioned in Section 2).

As a notation, given a TBox $\mathcal{T} = \{C_1 \sqsubseteq D_1, \ldots, C_n \sqsubseteq D_n\}$, we define the concept $C_\mathcal{T}$ as $C_\mathcal{T} = (\neg C_1 \sqcup D_1) \sqcap \cdots \sqcap (\neg C_n \sqcup D_n)$. As pointed out in [Baa90a] for $\mathcal{ALC}$, an interpretation satisfies a TBox $\mathcal{T}$ iff it satisfies the equation $C_\mathcal{T} = \top$. This result easily extends to $\mathcal{ALCNR}$, as stated in the following proposition.

**Proposition 5.1** *Given a TBox $\mathcal{T} = \{C_1 \sqsubseteq D_1, \ldots, C_n \sqsubseteq D_n\}$, an interpretation $\mathcal{I}$ satisfies $\mathcal{T}$ iff it satisfies the equation $C_\mathcal{T} = \top$.*

*Proof.* An interpretation $\mathcal{I}$ satisfies an inclusion $C \sqsubseteq D$ iff it satisfies the equation $\neg C \sqcup D = \top$; $\mathcal{I}$ satisfies the set of equations $\neg C_1 \sqcup D_1 = \top, \ldots,$ $\neg C_n \sqcup D_n = \top$ iff $\mathcal{I}$ satisfies $(\neg C_1 \sqcup D_1) \sqcap \cdots \sqcap (\neg C_n \sqcup D_n) = \top$. The claim follows. □

Given a knowledge base $\Sigma = \langle \mathcal{A}, \mathcal{T} \rangle$ and a concept $A$ not appearing in $\Sigma$, we define the knowledge base $\Sigma' = \langle \mathcal{A}', \mathcal{T}' \rangle$ as follows:

$$\begin{aligned} \mathcal{A}' &= \mathcal{A} \cup \{A(b) \mid b \text{ is an individual in } \Sigma\} \\ \mathcal{T}' &= \{A \dot{\leq} C_\mathcal{T} \sqcap \forall P_1.A \sqcap \cdots \sqcap \forall P_n.A\} \end{aligned}$$

where $P_1, P_2, \ldots, P_n$ are all the primitive roles appearing in $\Sigma$. Note that $\mathcal{T}'$ has a single inclusion, which could be also thought of as one primitive concept definition.

**Theorem 5.2** *$\Sigma = \langle \mathcal{A}, \mathcal{T} \rangle$ is satisfiable iff $\Sigma' = \langle \mathcal{A}', \mathcal{T}' \rangle$ is satisfiable.*

*Proof.* In order to simplify the machinery of the proof, we will use for $\mathcal{T}'$ the following (logically equivalent) form:

$$\mathcal{T}' = \{A \sqsubseteq C_\mathcal{T}, A \sqsubseteq \forall P_1.A, \ldots, A \sqsubseteq \forall P_n.A\}$$

"$\Leftarrow$" Suppose $\Sigma = \langle \mathcal{A}, \mathcal{T} \rangle$ satisfiable. For Theorem 3.6, there exists a complete constraint system $S$ without clash, which defines a canonical interpretation $\mathcal{I}_S$ which is a model of $\Sigma$. Define the constraint system $S'$ as follows:

$$S' = S \cup \{w{:}\, A \mid w \text{ is an object in } S\}$$

and call $\mathcal{I}_{S'}$ the canonical interpretation associated to $S'$. We prove that $\mathcal{I}_{S'}$ is a model of $\Sigma'$.

First observe that every assertion in $\mathcal{A}$ is satisfied by $\mathcal{I}_{S'}$ since $\mathcal{I}_{S'}$ is equal to $\mathcal{I}_S$ except for the interpretation of $A$, and $A$ does not appear in $\mathcal{A}$.

Therefore, every assertion in $\mathcal{A}'$ is also satisfied by $\mathcal{I}_{S'}$, either because it is an assertion of $\mathcal{A}$, or (if it is an assertion of the form $A(b)$) by definition of $S'$.

Regarding $\mathcal{T}'$, note that by definition of $S'$, we have $A^{\mathcal{I}_{S'}} = \Delta^{\mathcal{I}_{S'}} = \Delta^{\mathcal{I}_S}$; therefore both sides of the inclusions of the form $A \sqsubseteq \forall P_i.A$ $(i = 1, \ldots, n)$ are interpreted as $\Delta^{\mathcal{I}_{S'}}$, hence they are satisfied by $\mathcal{I}_{S'}$. Since $A$ does not appear in $C_{\mathcal{T}}$, we have that $(C_{\mathcal{T}})^{\mathcal{I}_{S'}} = (C_{\mathcal{T}})^{\mathcal{I}_S}$. Moreover, since $\mathcal{I}_S$ satisfies $\mathcal{T}$, we also have, by Proposition 5.1, that $(C_{\mathcal{T}})^{\mathcal{I}_S} = \Delta^{\mathcal{I}_S}$, therefore $(C_{\mathcal{T}})^{\mathcal{I}_{S'}} = (C_{\mathcal{T}})^{\mathcal{I}_S} = \Delta^{\mathcal{I}_S} = \Delta^{\mathcal{I}_{S'}}$. It follows that also both sides of the definition $A \sqsubseteq C_{\mathcal{T}}$ are interpreted as $\Delta^{\mathcal{I}_{S'}}$. In conclusion, $\mathcal{I}_{S'}$ satisfies $\mathcal{T}'$.

"$\Rightarrow$" Suppose $\Sigma' = \langle \mathcal{A}', \mathcal{T}' \rangle$ satisfiable. Again, because of Theorem 3.6, there exists a complete constraint system $S'$ without clash, which defines a canonical interpretation $\mathcal{I}_{S'}$ which is a model of $\Sigma'$. We show that $\mathcal{I}_{S'}$ is also a model of $\Sigma$.

First of all, the assertions in $\mathcal{A}$ are satisfied because $\mathcal{A} \subseteq \mathcal{A}'$, and $\mathcal{I}_{S'}$ satisfies every assertion in $\mathcal{A}'$. To prove that $\mathcal{I}_{S'}$ satisfies $\mathcal{T}$, we first prove the following equation:

$$A^{\mathcal{I}_{S'}} = \Delta^{\mathcal{I}_{S'}} \tag{1}$$

Equation 1 is proved by showing that, for every object $s \in \Delta^{\mathcal{I}_{S'}}$, $s$ is in $A^{\mathcal{I}_{S'}}$. In order to do that, observe a general property of constraint systems: every variable in $S'$ is a successor of an individual. This comes by the definition of the generating rules, which add variables to the constraint system only as 1-successors of existing objects, and at the beginning $S_{\Sigma'}$ contains only individuals.

Then, the above equation is proved by observing the following three facts:

1. for every individual $b$ in $\Delta^{\mathcal{I}_{S'}}$, $b \in A^{\mathcal{I}_{S'}}$;
2. if an object $s$ is in $A^{\mathcal{I}_{S'}}$, then because $\mathcal{I}_{S'}$ satisfies the inclusions $A^{\mathcal{I}_{S'}} \subseteq (\forall P_1.A)^{\mathcal{I}_{S'}}, \ldots, A^{\mathcal{I}_{S'}} \subseteq (\forall P_n.A)^{\mathcal{I}_{S'}}$, every 1-successor of $s$ is in $A^{\mathcal{I}_{S'}}$;
3. the successor relation is closed under the 1-successor relation

From the Fundamental Theorem on Induction (e.g. cfr. [Wan80, page 41] we conclude that every object $s$ of $\Delta^{\mathcal{I}_{S'}}$ is in $A^{\mathcal{I}_{S'}}$. This proves that Equation 1 holds.

From Equation 1, and the fact that $\mathcal{I}_{S'}$ satisfies the inclusion $A^{\mathcal{I}_{S'}} \subseteq (C_{\mathcal{T}})^{\mathcal{I}_{S'}}$, we derive that $(C_{\mathcal{T}})^{\mathcal{I}_{S'}} = \Delta^{\mathcal{I}_{S'}}$, that is $\mathcal{I}_{S'}$ satisfies the equation $C_{\mathcal{T}} = \top$. Hence, from Proposition 5.1, $\mathcal{I}_{S'}$ satisfies $\mathcal{T}$, and this completes the proof of the theorem. □

Note that the above reduction strongly relies on the fact that disjunction '$\sqcup$' is within the language (in order to express all inclusions inside the concept

$C_{\mathcal{T}}$). Therefore, the proof does not hold for those TKRS not allowing for disjunction of concepts (such as BACK).

The machinery present in this proof is not new. In fact, realizing that the inclusions $A \sqsubseteq \forall P_1.A, \ldots, A \sqsubseteq \forall P_n.A$ simulate a transitive closure on the roles $P_1, \ldots, P_n$, one can recognize similarities with the proofs given by Schild [Sch91] and Baader [Baa90a]. The difference is that their proofs rely on the notion of *connected model* (Baader uses the equivalent notion of *rooted* model). In contrast, the models we obtain are not connected, when the individuals in the knowledge base are not. What we exploit is the weaker property that every variable in the model is a successor of an individual.

# 6 Discussion

In this paper we proved the decidability of the main inference services of a TKRS based on the concept language $\mathcal{ALCNR}$. We believe that this result is not only of theoretical importance, but has the following impacts on existing TKRS.

First of all, a complete procedure working in exponential space can be easily devised from the calculus provided in Section 4. From this procedure, one can build more efficient (but still complete) ones by applying standard optimization techniques such as those described in [BHN+92]. Such procedure might work well in practical cases, despite their worst case intractability.

Secondly, a complete procedure (possibly optimized) offers a benchmark for comparing incomplete procedures, not only in terms of performance, but also in terms of missed inferences. Let us illustrate this point in detail, by providing a blatant paradox: consider the mostly incomplete constant-time procedure, answering always "No" to any check. Obviously this useless procedure outperforms any other one, if missed inferences are not taken into account. This paradox shows that incomplete procedures can be meaningfully compared only if missed inferences are considered. But to recognize missed inferences over large examples, one needs exactly a complete procedure—even if not an efficient one—like ours.

Thirdly, new incomplete procedures can be obtained from the calculus by modifying some of the propagation rules. Since the rules build up a model, modifications to them have a semantical counterpart which gives a precise account of the incomplete procedures obtained. For instance, define the depth of a variable $x$ as the number of variables which are predecessors of $x$. Then, an incomplete calculus could be devised, which generates variables only to a given depth—say, linear depth in the size of the KB. This calculus would miss contradictions (and hence inferences, by refutation) occurring in

variables which are "far away" from the known individuals of the KB, and this is a meaningful explanation of the incompleteness, even for a non-expert user. From a computational point of view, an immediate consequence of the complexity analysis carried over in this paper is that such an incomplete procedure would run in polynomial space.

# Acknowledgements

We thank Maurizio Lenzerini for the inspiration of this work, as well as for several discussions that contributed to the paper. Werner Nutt pointed us the observation mentioned at the end of Section 4, and we thank him and Franz Baader for many helpful comments on earlier drafts. The third author also acknowledges Yoav Shoham for his hospitality at the Computer Science Department of Stanford University, while the author was developing part of this research.

This work has been supported by the ESPRIT Basic Research Action N.6810 (COMPULOG 2) and by the Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo of the CNR (Italian Research Council).

# References

[Abr74] J.R. Abrial. Data semantics. In J.W. Klimbie and K.L. Koffeman, editors, *Data Base Management*, pages 1–59. North-Holland, 1974.

[Baa90a] Franz Baader. Augmenting concept languages by transitive closure of roles: An alternative to terminological cycles. Technical Report RR-90-13, Deutsches Forschungszentrum für Künstliche Intelligenz, Postfach 2080, D-6750 Kaiserslautern, Germany, 1990.

[Baa90b] Franz Baader. Terminological cycles in KL-ONE-based KR-languages. Technical Report RR-90-01, Deutsches Forschungszentrum für Künstliche Intelligenz, Postfach 2080, D-6750 Kaiserslautern, Germany, 1990.

[BBH+90] Franz Baader, Hans-Jürgen Bürckert, Bernhard Hollunder, Werner Nutt, and Jörg H. Siekmann. Concept logics. In John W. Lloyd, editor, *Computational Logics, Symposium Proceedings*, pages 177–201. Springer Verlag, 1990.

[BBMAR89] Alexander Borgida, Ronald J. Brachman, Deborah L. McGuinness, and Lori Alperin Resnick. CLASSIC: A structural data model for objects. In *ACM SIGMOD*, 1989.

[BGN89] H.W. Beck, S.K. Gala, and S.B. Navathe. Classification as a query processing technique in the CANDIDE semantic data model. In *Fifth IEEE International Conference on Data Engineering, Los Angeles*, 1989.

[BH91] Franz Baader and Bernhard Hollunder. A terminological knowledge representation system with complete inference algorithm. In *Proc. of the Workshop on Processing Declarative Knowledge, PDK-91*, Lecture Notes in Artificial Intelligence. Springer Verlag, 1991.

[BHN+92] Franz Baader, Bernhard Hollunder, Bernhard Nebel, Hans-Jürgen Profitlich, and Enrico Franconi. An empirical analysis of optimization techniques for terminological representation systems, or making KRIS get a move on. In *Proc. of the 3rd Int. Conf. on Principles of Knowledge Representation and Reasoning KR-92*. Morgan Kaufmann, 1992.

[BL84] Ronald J. Brachman and Hector J. Levesque. The tractability of subsumption in frame-based description languages. In *Proc. of the 4th Nat. Conf. on Artificial Intelligence AAAI-84*, 1984.

[BPL85] Ronald J. Brachman, Victoria Pigman Gilbert, and Hector J. Levesque. An essential hybrid reasoning system: Knowledge and symbol level accounts in KRYPTON. In *Proc. of the Int. Joint Conf. on Artificial Intelligence*, pages 532–539, Los Angeles, Cal., 1985.

[BS85] Ronald J. Brachman and James G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2):171–216, 1985.

[DHL+92] Francesco M. Donini, Bernhard Hollunder, Maurizio Lenzerini, Alberto Marchetti Spaccamela, Daniele Nardi, and Werner Nutt. The complexity of existential quantification in concept languages. *Artificial Intelligence*, 53(2-3):309–328, 1992.

[DLNN91a] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Werner Nutt. The complexity of concept languages. In James Allen, Richard Fikes, and Erik Sandewall, editors, *Proc. of the*

*2nd Int. Conf. on Principles of Knowledge Representation and Reasoning KR-91*, pages 151–162. Morgan Kaufmann, 1991.

[DLNN91b] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Werner Nutt. Tractable concept languages. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence IJCAI-91*, Sydney, 1991.

[DLNS91] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Andrea Schaerf. A hybrid system integrating datalog and concept languages. In *Proc. of the 2nd Italian Conf. on Artificial Intelligence*, number 549 in Lecture Notes in Artificial Intelligence. Springer Verlag, 1991. An extended version appeared also in the Working Notes of the AAAI Fall Symposium "Principles of Hybrid Reasoning", 1991.

[DLNS92] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Andrea Schaerf. From subsumption to instance checking. Technical Report 15.92, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", 1992.

[Fit90] Melvin Fitting. *First-Order Logic and Automated Theorem Proving*. Springer Verlag, New York, 1990.

[Har84] David Harel. Dynamic logic. In *Handbook of Philosophical Logic*, volume 2, pages 497–640. D. Reidel, Dordrecht, Holland, 1984.

[HC84] George E. Hughes and M. J. Cresswell. *A Companion to Modal Logic*. Methuen, London, 1984.

[HKNP92] Jochen Heinson, Daniel Kudenko, Bernhard Nebel, and Hans-Jürgen Profitlich. An empirical analysis of terminological representation systems. In *Proc. of the 10th Nat. Conf. on Artificial Intelligence AAAI-92*, pages 767–773. AAAI Press/The MIT Press, 1992.

[HN90] Bernhard Hollunder and Werner Nutt. Subsumption Algorithms for Concept Languages. Technical Report RR-90-04, Deutsches Forschungszentrum für Künstliche Intelligenz, Postfach 2080, D-6750 Kaiserslautern, Germany, 1990.

[HNSS90] Bernhard Hollunder, Werner Nutt, and Manfred Schmidt-Schauß. Subsumption algorithms for concept description languages. In *Proc. of 9th the European Conf. on Artificial Intelligence ECAI-90*, pages 348–353, London, 1990. Pitman.

[Hol90] Bernhard Hollunder. Hybrid inferences in KL-ONE-based knowledge representation systems. In *German Workshop on Artificial Intelligence*. Springer Verlag, 1990.

[HU79] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, Mass., 1979.

[KBR86] Thomas S. Kaczmarek, Raymond Bates, and Gabriel Robins. Recent developments in NIKL. In *Proc. of the 5th Nat. Conf. on Artificial Intelligence AAAI-86*, pages 978–985, 1986.

[Lev84] Hector J. Levesque. A fundamental tradeoff in knowledge representation and reasoning. Technical Report 658, Fairchild, 1984. Also available as FLAIR Technical Report No. 35.

[Mac91] Robert M. MacGregor. Inside the LOOM description classifier. *SIGART Bulletin*, 2(3):88–92, June 1991.

[Mac92] Robert MacGregor. What's needed to make a description logic a good KR citizen. In *Working Notes of the AAAI Fall Symposium on Issue on Description Logics: Users meet Developers*, pages 53–55, 1992.

[MB87] Robert MacGregor and R. Bates. The Loom knowledge representation language. Technical Report ISI/RS-87-188, University of Southern California, Information Science Institute, Marina del Rey, Cal., 1987.

[MB92] Robert MacGregor and David Brill. Recognition algorithms for the Loom classifier. In *Proc. of the 10th Nat. Conf. on Artificial Intelligence AAAI-92*, pages 774–779. AAAI Press/The MIT Press, 1992.

[MBW80] J. Mylopoulos, P.A. Bernstein, and E. Wong. A language facility for designing database-intensive applications. *ACM Transactions on Database Systems*, 5(2):185–207, June 1980.

[McA92] David McAllester. Unpublished manuscript. 1992.

[McG92] Deborah L. McGuinness. Making description logic based knowledge representation systems more usable. In *Working Notes of the AAAI Fall Sysmposium on Issue on Description Logics: Users meet Developers*, pages 56–58, 1992.

[Neb88] Bernhard Nebel. Computational complexity of terminological reasoning in BACK. *Artificial Intelligence*, 34(3):371–383, 1988.

[Neb90a] Bernhard Nebel. *Reasoning and Revision in Hybrid Representation Systems*. Number 422 in Lecture Notes in Artificial Intelligence. Springer Verlag, 1990.

[Neb90b] Bernhard Nebel. Terminological reasoning is inherently intractable. *Artificial Intelligence*, 43:235–249, 1990.

[Neb91] Bernhard Nebel. Terminological cycles: Semantics and computational properties. In John F. Sowa, editor, *Principles of Semantic Networks*, pages 331–361. Morgan Kaufmann, 1991.

[Nut92] Werner Nutt. Personal communication, 1992.

[Pat84] Peter F. Patel-Schneider. Small can be beautiful in knowledge representation. In *Proc. of the IEEE Workshop on Knowledge-Based Systems*, 1984. An extended version appeared as Fairchild Tech. Rep. 660 and FLAIR Tech. Rep. 37, October 1984.

[Pat89] Peter F. Patel-Schneider. Undecidability of subsumption in NIKL. *Artificial Intelligence*, 39:263–272, 1989.

[QK90] Joachim Quantz and Carsten Kindermann. Implementation of the BACK system version 4. Technical Report KIT-Report 78, FB Informatik, Technische Universität Berlin, Berlin, Germany, 1990.

[Ric91] Charles Rich, editor. SIGART bulletin. Special issue on implemented knowledge representation and reasoning systems. (2)3, June 1991.

[Sch91] Klaus Schild. A correspondence theory for terminological logics: Preliminary report. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence IJCAI-91*, Sydney, 1991.

[SS89] Manfred Schmidt-Schauß. Subsumption in KL-ONE is undecidable. In Ron J. Brachman, Hector J. Levesque, and Ray Reiter, editors, *Proc. of the 1st Int. Conf. on Principles of Knowledge Representation and Reasoning KR-89*. Morgan Kaufmann, 1989.

[SSS91] Manfred Schmidt-Schauß and Gert Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26, 1991.

[Vil91] Marc Vilain. Deduction as parsing: Tractable classification in the KL-ONE framework. In *Proc. of the 9th Nat. Conf. on Artificial Intelligence AAAI-91*, pages 464–470, 1991.

[Wan80] Mitchell Wand. *Induction, Recursion, and Programming*. North-Holland, 1980.

[WS92] William A. Woods and James G. Schmolze. The KL-ONE family. In F.W. Lehmann, editor, *Semantic Networks in Artificial Intelligence*, pages 133–178. Pergamon Press, 1992. Published as a special issue of *Computers & Mathematics with Applications*, Volume 23, Number 2–9.