Fachbereich Informatik der Universität Hamburg

Vogt-Kölln-Str. 30 ◊ D-22527 Hamburg / Germany

University of Hamburg - Computer Science Department

Mitteilung Nr. 276 • Memo No. 276

A Concept Language with Role-Forming Predicate Restrictions

Carsten Lutz, Volker Haarslev, Ralf Möller

Arbeitsbereich KOGS

FBI-HH-M-276/97

Dezember 1997

A Concept Language with Role-Forming Predicate Restrictions

Carsten Lutz, Volker Haarslev, Ralf Möller
University of Hamburg, Computer Science Department,
Vogt-Kölln-Str. 30, 22527 Hamburg, Germany
`http://kogs-www.informatik.uni-hamburg.de/~<name>/`

Abstract

The development of language constructs for defining concept and role terms is an important goal of research on description logic formalisms. However, most decidable descriptions logics only support the definition of roles with very limited properties. For more complex roles, e.g. roles needed to represent Allen's temporal relations, a higher expressivity is required. This paper formally introduces a new description logic formalism called $\mathcal{ALCRP}(\mathcal{D})$. It is a descendant of $\mathcal{ALC}(\mathcal{D})$ and thus allows one to represent abstract and concrete information. Furthermore, it contains a new operator for defining roles based on predicates over (concrete) properties of objects. In previous work by the authors, reasoning in $\mathcal{ALCRP}(\mathcal{D})$ was proven to be undecidable in general. In this report we show that reasoning in $\mathcal{ALCRP}(\mathcal{D})$ is decidable if certain restrictions are posed on the structure of terminologies. In fact, the free combinability of some operators has to be restricted. The representational expressiveness of so-called "restricted terminologies" obtained in this way is of course lower than the expressiveness of unrestricted ones. Nevertheless, the resulting formalism is still a powerful and usable tool for conceptual reasoning that supports the definition of roles with very complex properties.

1 Introduction

Description logics (DLs) which are also known as terminological logics or KL-ONE-like systems are a family of formalisms commonly used for the representation of and reasoning about structured knowledge.[1] In comparison to other formalisms, a DL is based on a formal semantics, i.e. the reasoning services to be provided by an inference engine can be specified without reference to a specific implementation. An important goal behind description logic research is to design formalisms in which reasoning is decidable. Many results from description logic theory concerning decidability, expressiveness and computational complexity of inference algorithms are available. Description

[1]For an introduction see e.g. Woods and Schmolze (1992).

logics follow a dichotomic paradigm for knowledge representation: General conceptual knowledge about a domain is represented separately from specific, assertional knowledge about a certain world. The first type of knowledge is located in the TBox, which contains a taxonomy of concepts that is called the terminology. The second kind of knowledge is represented in the so-called ABox. In the TBox, definitions are checked for consistency and concepts are hierarchically organized with respect to "specificity" (see below for a formal definition). Concepts can be thought of as unary predicates in first order logic. They can be defined using roles, which are interpreted as relations or binary predicates of first order logic. For example, universal and existential quantification over roles can be used for concept definitions. In the ABox, concept membership can be asserted for individual domain objects.

In order to demonstrate modeling with description logics, we briefly discuss some examples. Let *male* be a concept. Then the concept $male \sqcap \exists offspring.human$ has exactly the extension[2] of all males which are related to at least one human via the offspring role (existential quantification). Thus, the concept term given above could have been named *father*.

It can be seen as a limitation that standard description logics can only handle abstract knowledge. Imagine that we want to represent the ages of humans as natural numbers. This cannot be done in most description logics. There are, however, some DL formalisms which overcome this limitation and are able to additionally represent knowledge about so-called concrete objects such as numbers and polygons. One important formalism of this type is the language $\mathcal{ALC}(\mathcal{D})$ defined by Baader and Hanschke (1991a). With this language, one could define an old person as $human \sqcap >_{60} (age)$. Here, *age* is a single-valued role (those roles are called features). The feature *age* attaches concrete objects that represent natural numbers to abstract objects (in this case of type *human*). The extension of the above-mentioned concept term is "All humans who are older than 60 years." The example demonstrates that defining concept terms based on predicates over concrete objects (e.g. "$>_{60}$") is a valuable tool for knowledge representation.

The language $\mathcal{ALCRP}(\mathcal{D})$ defined in [Lutz and Möller 1997] goes one step further. It also allows one to define roles based on predicates over concrete objects. Like in the $\mathcal{ALC}(\mathcal{D})$ example above, these predicates over concrete objects that are attached to abstract objects via features can be seen as properties of these abstract objects. Take again humans and their ages as an example. The age is a property of each object which is of type *human* (it is a concrete object attached via the *age* feature). Assume that we would like

[2]The notion of an "extension" is used here in the sense of semiotics.

to define the concept *oldest-person*.[3] In $\mathcal{ALCRP}(\mathcal{D})$, one could use the term $human \sqcap \neg\exists older.human$, where *older* is a defined role whose extension is the set of those pairs of objects (a, b) such that the natural number attached to object b via the feature *age* is greater then the natural number attached to object a via the same feature. Thus, only those objects of type *human* are inside the extension of the concept for which no other object exists that is older and also of type *human*. An equivalent formalization that takes all domain objects into account cannot be expressed using $\mathcal{ALC}(\mathcal{D})$.

$\mathcal{ALCRP}(\mathcal{D})$ is a very powerful language for reasoning about abstract and concrete knowledge. Like $\mathcal{ALC}(\mathcal{D})$ it can be parameterized with a concrete domain, which is a set of concrete objects plus a set of predicates over these concrete objects. Unfortunately, reasoning in $\mathcal{ALCRP}(\mathcal{D})$ is undecidable in general as proven in [Lutz and Möller 1997]. In this paper we propose syntactic restrictions to be posed on $\mathcal{ALCRP}(\mathcal{D})$-terminologies. We show that w.r.t. these so-called restricted terminologies sound and complete algorithms for deciding the common reasoning problems exist. Decidability is achieved by restricting the free combinability of operators in restricted terminologies. Some combinations of value and exists restrictions are not allowed if they quantify over defined roles. Furthermore, the use of the concept forming predicate operator known from $\mathcal{ALC}(\mathcal{D})$ has to be restricted, too. These restrictions are solely motivated by decidability issues. From the knowledge engineer's point of view they are relatively strong constraints on the possible structure of concept terms. Another approach for defining a decidable version of $\mathcal{ALCRP}(\mathcal{D})$ would have been to pose limitations on the allowed set of predicates that can be used with concept- and role-forming operators. But this seems to be less promising because the intended areas of application, representing time and space, already require fairly complex predicates which presumably cause undecidability of the resulting language.

In [Möller, Haarslev, and Lutz 1997] it is shown that restricted terminologies are still a powerful tool for knowledge representation. Apart from toy domains such as the one discussed above, $\mathcal{ALCRP}(\mathcal{D})$ with restricted terminologies can (with an appropriate parameterization) immediately be used to represent temporal as well as spatial (topological) relations (see [Allen 1983] and [Egenhofer 1991; Randell, Cui, and Cohn 1992], respectively). Throughout this paper, we will use examples from the domain of temporal reasoning.

The rest of this paper is organized as follows. In Section 2 and 3 the language $\mathcal{ALCRP}(\mathcal{D})$ is formally introduced and the notion of a restricted terminology is defined. Then, in Section 4 the reasoning services for $\mathcal{ALCRP}(\mathcal{D})$

[3]The extension of this concept does not have a cardinality greater than one unless there are some people which have the same age.

are introduced. By representing Allen's relations we demonstrate the expressiveness of restricted terminologies in Section 5. Afterwards, in Section 6 an algorithm for deciding subsumption is proposed which is proven to be sound and complete in Section 7.

2 Terminologies

In this section, the language for defining concepts in $\mathcal{ALCRP}(\mathcal{D})$ is presented. $\mathcal{ALCRP}(\mathcal{D})$ is a descendant of $\mathcal{ALC}(\mathcal{D})$. A formal definition of $\mathcal{ALC}(\mathcal{D})$ can be found in [Baader and Hanschke 1991a]. The syntax and semantics defined there have been used as a starting-point for the definition of $\mathcal{ALCRP}(\mathcal{D})$.

$\mathcal{ALCRP}(\mathcal{D})$ can be parameterized with a concrete domain which consists of a set of concrete objects and a set of predicates.

Definition 1. [Baader and Hanschke 1991a] A *concrete* domain $\mathcal{D}$ is a pair $(\Delta_{\mathcal{D}}, \Phi_{\mathcal{D}})$, where $\Delta_{\mathcal{D}}$ is a set called the domain, and $\Phi_{\mathcal{D}}$ is a set of predicate names. Each predicate name P from $\Phi_{\mathcal{D}}$ is associated with an arity n, and an n-ary predicate $P^{\mathcal{D}} \subseteq \Delta_{\mathcal{D}}^n$.
A concrete domain $\mathcal{D}$ is called *admissible* iff

1. the set of its predicate names is closed under negation and contains a name for $\Delta_{\mathcal{D}}$.

2. the satisfiability problem for finite conjunctions of predicates is decidable.

The fact that $\mathcal{ALCRP}(\mathcal{D})$ can be parameterized by only one concrete domain is not a limitation because in [Baader and Hanschke 1991b] it is shown that the union of two admissible concrete domains again yields an admissible concrete domain. We are now ready to define role terms in $\mathcal{ALCRP}(\mathcal{D})$.

Definition 2. Let R and F be disjoint sets of role and feature names, respectively. Any element of R and any element of F is a *role term* (*atomic* role term). The elements of F are also called *features*. A composition of features (written $f_1 f_2 \cdots$) is called a feature chain. A feature chain of length one is also a feature chain. If $P \in \Phi_{\mathcal{D}}$ is a predicate name with arity $n+m$ and u_1, ... ,u_n as well as v_1, ... ,v_m are $n+m$ feature chains, then the expression $\exists(u_1, \ldots, u_n)(v_1, \ldots, v_m).P$ (*role-forming predicate restriction*) is a role term as well (*complex* role term).[4] Let S be a role name and let T be a role term. Then $S \doteq T$ is a terminological axiom.

[4]Note that there have to be at least one u and one v.

The next definition introduces concept terms of $\mathcal{ALCRP}(\mathcal{D})$. As we will see later, if decidable reasoning algorithms are needed for a certain terminology, not all of the given operators can be freely combined.

Definition 3. Let C be a set of concept names which is disjoint from R and F. Any element of C is a *concept term* (*atomic* concept term). If C and D are concept terms, R is an arbitrary role term or a feature, $P \in \Phi_{\mathcal{D}}$ is a predicate name with arity n, and $u_1, \ldots, u_n$ are feature chains, then the following expressions are also concept terms:

1. $C \sqcap D$ (*conjunction*),
2. $C \sqcup D$ (*disjunction*),
3. $\neg C$ (*negation*),
4. $\exists R.C$ (*concept exists restriction*),
5. $\forall R.C$ (*concept value restriction*), and
6. $\exists u_1, \ldots, u_n.P$ (*predicate exists restriction*).

For all kinds of exists and value restrictions, the role term or list of feature chains may be written in parentheses. Let A be a concept name and let D be a concept term. Then $A \doteq D$ is a terminological axiom as well. A finite set of terminological axioms $\mathcal{T}$ is called a *terminology* or *TBox* if no concept or role name in $\mathcal{T}$ appears more than once on the left-hand side of a definition and, furthermore, if no cyclic definitions are present.

The syntax we use for the concept-forming predicate restriction operator is taken from [Hanschke 1996]. Please note that an all-quantified concept-forming predicate operator $\forall f_1, \ldots, f_m.P$ can be expressed as $\exists f_1, \ldots, f_m.P \sqcup \forall f_1.\bot \sqcup \ldots \sqcup \forall f_m.\bot$, where $\bot^{\mathcal{I}} = \emptyset$.[5]

We do not allow cyclic concept definitions. Please refer to [Nebel 1991] for an investigation of terminological cycles. In the following, we will introduce a special form of terminology called restricted terminology. It will be shown later that reasoning with respect to this special kind of terminology is decidable.

A terminology $\mathcal{T}$ is said to be in *unfolded* form iff none of the concept and role names used on the right sides of the terminological axioms in $\mathcal{T}$ occurs also on the left side of any terminological axiom in $\mathcal{T}$. Any terminology can be transformed into an unfolded form by iteratively replacing

[5] $\bot$ can be expressed as $A \sqcap \neg A$.

concept and role names by their defining terms. This algorithm terminates since the terminology is required to be acyclic. It is deterministic since every concept name is allowed only once on the left-hand side of a terminological axiom. Any unfolded terminology can then be transformed to *negation normal form* (NNF). An unfolded terminology is said to be in NNF iff negation is only applied to atomic concept terms. As we will see in Section 4, the two transformations of unfolding and conversion to NNF are also needed as preprocessing steps for terminological reasoning. The next lemma gives an algorithm for converting an arbitrary concept term to an equivalent one that is in NNF. We omit the proof of correctness which is trivial.

Lemma 4. *Let $\mathcal{D}$ be an admissible concrete domain. Let the name for $\Delta_{\mathcal{D}}$ in $\Phi_{\mathcal{D}}$ be $\top_{\mathcal{D}}$. Let C and D be concept terms, $\hat{R}$ be a role term which is not a feature, f (possibly with index) be a feature name, P be a predicate name with arity n from $\Phi_{\mathcal{D}}$, $\overline{P}$ be the negation of P in $\Phi_{\mathcal{D}}$ and $u_1, \ldots, u_n$ be feature chains. Then, if the following transformation rules are applied to the concept terms of an unfolded TBox $\mathcal{T}$ as often as possible, the resulting TBox $\mathcal{T}'$ is equivalent to $\mathcal{T}$ and in negation normal form (NNF).*

1. $\neg(C \sqcap D) \Longrightarrow (\neg C \sqcup \neg D)$, $\neg(C \sqcup D) \Longrightarrow (\neg C \sqcap \neg D)$, $\neg\neg C \Longrightarrow C$
2. $\neg(\exists \hat{R}.C) \Longrightarrow \forall \hat{R}.\neg C$, $\neg(\forall \hat{R}.C) \Longrightarrow \exists \hat{R}.\neg C$,
   $\neg(\exists f.C) \Longrightarrow \forall f.\neg C \sqcup \exists f.\top_{\mathcal{D}}$, $\neg(\forall f.C) \Longrightarrow \exists f.\neg C \sqcup \exists f.\top_{\mathcal{D}}$
3. $\neg(\exists u_1, \ldots, u_n.P) \Longrightarrow \exists u_1, \ldots, u_n.\overline{P} \sqcup \forall u_1.\top \sqcup \ldots \sqcup \forall u_n.\top$

The term $\forall u.C$ with a feature chain u is an abbreviation for $\forall f_1. \cdots \forall f_n.C$ if $u = f_1 \cdots f_n$.

Based on the definition of an unfolded terminology we now define the notion of a restricted terminology.

Definition 5. A terminology $\mathcal{T}$ is called *restricted* iff its equivalent in unfolded NNF fulfills the following conditions:

1. For any (sub)concept term C in $\mathcal{T}$ that is of the form $\forall R_1.D$ where R_1 is a complex role term, D does not contain any terms of the form $\exists R_2.D$ where R_2 is also a complex role term.
2. For any (sub)concept term C in $\mathcal{T}$ that is of the form $\exists R_1.D$ where R_1 is a complex role term, D does not contain any terms of the form $\forall R_2.D$ where R_2 is also a complex role term.

3. For any (sub)concept term C in $\mathcal{T}$ that is of the form $\forall R.D$ or $\exists R.D$ where R is a complex role term, D contains only predicate exists restrictions that quantify over attribute chains of length of 1 and furthermore do not occur inside any value and exists restrictions that are also contained in D.

A grammar in EBNF notation that formalizes these conditions is given in Appendix A. Using this grammar, an alternate definition of restrictedness can be given as follows: A terminology $\mathcal{T}$ is restricted iff the unfolded NNF equivalents of the right-hand sides of all axioms in $\mathcal{T}$ are words of the language defined by this grammar.

Due to the nature of the restrictions, it may not be obvious for a knowledge engineer to determine whether a new terminological axiom can safely be added to a restricted terminology without losing the restrictedness property. This is the case because the unfolded NNF of the concept term has to be considered w.r.t. the whole terminology. We illustrate the resulting modeling constraints by considering three very simple terminologies. The terminologies are not restricted because they all violate one of the conditions given in Definition 5. Let C and D be concept names, R_a be an atomic role term, R_c be a complex role term, P be a predicate name, f be a feature, and u be a feature chain with a length greater than 1.

$$\mathcal{T}_1 : \{C \doteq \forall R_c.\exists R_c.D\}$$
$$\mathcal{T}_2 : \{C \doteq \exists R_c.\exists u.P\}$$
$$\mathcal{T}_3 : \{C \doteq \forall R_c.\forall R_a.\exists f.P\}$$

The next definition gives a set-theoretic semantics for the language just introduced.

Definition 6. An *interpretation* $\mathcal{I} = (\Delta_{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a set $\Delta_{\mathcal{I}}$ (the abstract domain) and an interpretation function $\cdot^{\mathcal{I}}$. The sets $\Delta_{\mathcal{D}}$ (see Definition 1) and $\Delta_{\mathcal{I}}$ must be disjoint. The interpretation function maps each concept name C to a subset $C^{\mathcal{I}}$ of $\Delta_{\mathcal{I}}$, each role name R to a subset $R^{\mathcal{I}}$ of $\Delta_{\mathcal{I}} \times \Delta_{\mathcal{I}}$, and each feature name f to a partial function $f^{\mathcal{I}}$ from $\Delta_{\mathcal{I}}$ to $\Delta_{\mathcal{D}} \cup \Delta_{\mathcal{I}}$, where $f^{\mathcal{I}}(a) = x$ will be written as $(a, x) \in f^{\mathcal{I}}$. If $u = f_1 \cdots f_n$ is a feature chain, then $u^{\mathcal{I}}$ denotes the composition $f_1^{\mathcal{I}} \circ \cdots \circ f_n^{\mathcal{I}}$ of the partial functions $f_1^{\mathcal{I}}, \ldots, f_n^{\mathcal{I}}$. Let the symbols C, D, R, P, u_1, … ,u_m, and v_1, … ,v_m be defined as in Definition 2 and 3, respectively. Then the interpretation function can be extended to arbitrary concept and role terms as follows:

$$\begin{aligned}
(C \sqcap D)^{\mathcal{I}} &:= C^{\mathcal{I}} \cap D^{\mathcal{I}} \\
(C \sqcup D)^{\mathcal{I}} &:= C^{\mathcal{I}} \cup D^{\mathcal{I}} \\
(\neg C)^{\mathcal{I}} &:= \Delta_{\mathcal{I}} \setminus C^{\mathcal{I}} \\
(\exists R.C)^{\mathcal{I}} &:= \{a \in \Delta_{\mathcal{I}} \mid \exists b \in \Delta_{\mathcal{I}} \colon (a,b) \in R^{\mathcal{I}}, b \in C^{\mathcal{I}}\} \\
(\forall R.C)^{\mathcal{I}} &:= \{a \in \Delta_{\mathcal{I}} \mid \forall b \in \Delta_{\mathcal{I}} \colon (a,b) \in R^{\mathcal{I}} \rightarrow b \in C^{\mathcal{I}}\} \\
(\exists u_1, \ldots, u_n.P)^{\mathcal{I}} &:= \{a \in \Delta_{\mathcal{I}} \mid \exists x_1, \ldots, x_n \in \Delta_{\mathcal{D}} \colon \\
&\qquad (a, x_1) \in u_1^{\mathcal{I}}, \ldots, (a, x_n) \in u_n^{\mathcal{I}}, (x_1, \ldots, x_n) \in P^{\mathcal{D}}\} \\
(\exists (u_1, \ldots, u_n)(v_1, \ldots, v_m).P)^{\mathcal{I}} &:= \\
&\{(a,b) \in \Delta_{\mathcal{I}} \times \Delta_{\mathcal{I}} \mid \exists x_1, \ldots, x_n, y_1, \ldots, y_m \in \Delta_{\mathcal{D}} \colon \\
&\quad (a, x_1) \in u_1^{\mathcal{I}}, \ldots, (a, x_n) \in u_n^{\mathcal{I}}, \\
&\quad (b, y_1) \in v_1^{\mathcal{I}}, \ldots, (b, y_m) \in v_m^{\mathcal{I}}, \\
&\quad (x_1, \ldots, x_n, y_1, \ldots, y_m) \in P^{\mathcal{D}}\}
\end{aligned}$$

An interpretation $\mathcal{I}$ is a *model* of a TBox $\mathcal{T}$ iff it satisfies $A^{\mathcal{I}} = D^{\mathcal{I}}$ for all terminological axioms $A \doteq D$ in $\mathcal{T}$.

3 The Assertional Language

In this section, the language for representing knowledge about individual worlds is introduced. An *ABox* is a finite set of assertional axioms which are defined as follows:

Definition 7. Let O_D and O_A be two disjoint sets of object names. If A is a concept name, R an atomic or complex role term, f a feature name, P a predicate name with arity n, a and b are elements of O_A and x, x_1, … ,x_n are elements of O_D, then the following expressions are *assertional axioms.*

$$a : A,\ (a,b) : R,\ (a,x) : f,\ (x_1, \ldots, x_n) : P$$

In axioms of the form $a : A$, usually concept terms instead of concept names are allowed. However, the more restricted form used here is not a limitation because any concept term that would otherwise be used directly in the ABox can first be defined as a concept in the TBox. Then, in the ABox, it can simply be referred to by the associated name. If we supported concept terms in the ABox we would have to ensure that the concept terms used there do not violate the restrictedness criterion w.r.t. the TBox. For the sake of brevity this is not discussed in this paper. In order to deal with ABox assertions, we now extend the interpretation function.

Definition 8. An *interpretation* for the assertional language is an interpretation for the concept language which additionally maps every object name from O_A to a single element of $\Delta_{\mathcal{I}}$ and every object name from O_D to a single element from $\Delta_{\mathcal{D}}$. We assume that the unique name assumption does not hold, that is $a^{\mathcal{I}} = b^{\mathcal{I}}$ may hold even if $a \neq b$. An interpretation satisfies an assertional axiom

$$a : C \text{ iff } a^{\mathcal{I}} \in C^{\mathcal{I}}, \quad (a, b) : R \text{ iff } (a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}, \quad (a, x) : f \text{ iff } f^{\mathcal{I}}(a^{\mathcal{I}}) = x^{\mathcal{I}},$$
$$(x_1, \ldots, x_n) : P \text{ iff } (x_1^{\mathcal{I}}, \ldots, x_n^{\mathcal{I}}) \in P^{\mathcal{D}}$$

An interpretation is a *model* of an ABox $\mathcal{A}$ w.r.t. a TBox $\mathcal{T}$, iff it is a model of $\mathcal{T}$ and furthermore satisfies all assertional axioms in $\mathcal{A}$. An ABox is *consistent* w.r.t. a TBox $\mathcal{T}$ iff it has a model.

4 Reasoning

In this section the standard inference problems on TBoxes and ABoxes are formally defined. We will introduce all reasoning problems with reference to a certain TBox. This is necessary because the reasoning problems are known to be decidable only if the TBox is restricted (see Section 7).

Equivalence Two concept terms C and D are *equivalent* w.r.t. a TBox $\mathcal{T}$, iff $D^{\mathcal{I}} = C^{\mathcal{I}}$ for all models $\mathcal{I}$ of $\mathcal{T}$.

Subsumption A concept term C *subsumes* a concept term D w.r.t. a TBox $\mathcal{T}$ (written $D \preceq_{\mathcal{T}} C$), iff $D^{\mathcal{I}} \subseteq C^{\mathcal{I}}$ for all models $\mathcal{I}$ of $\mathcal{T}$.

Satisfiability A concept term C is *satisfiable* w.r.t. a TBox $\mathcal{T}$ iff there exists a model $\mathcal{I}$ of $\mathcal{T}$ such that $C^{\mathcal{I}} \neq \emptyset$.

ABox consistency An ABox $\mathcal{A}$ is consistent w.r.t. a TBox $\mathcal{T}$ iff there exists a model $\mathcal{I}$ of $\mathcal{A}$ w.r.t. $\mathcal{T}$.

Equivalence is the problem of deciding if two different concept descriptions in a TBox denote the same concept. Using subsumption one can determine whether, in a given TBox, a concept is a subconcept of another concept. This inference is needed to compute the taxonomy in the TBox. Satisfiability is needed to detect concepts in a TBox that cannot have any instances. And last, ABox consistency can be used to decide if a given world contains contradictions w.r.t. a certain TBox.

All reasoning problems introduced here are often defined without reference to a TBox. As Baader and Hanschke (1991b) note, reasoning w.r.t. a

TBox can be reduced to reasoning without reference to a TBox by unfolding the TBox and then operating directly on the concept terms. But, as has been mentioned before, in the special case of $\mathcal{ALCRP}(\mathcal{D})$, reasoning is only known to be decidable w.r.t. restricted TBoxes. Unfolding a restricted TBox yields a restricted form of concept terms. The restricted form is formally defined by the grammar given in Appendix A. Reasoning w.r.t. restricted TBoxes is a more specific problem than reasoning without reference to a TBox.

There are intimate relationships between the inferences defined above. In the following, we will give only those reductions that are needed in this paper. In fact, equivalence, subsumption and satisfiability form a set of mutually reducible inferences. ABox consistency is a more difficult problem.

- Equivalence can be reduced to subsumption since a concept C is equivalent to a concept D w.r.t. a TBox $\mathcal{T}$ iff $C \preceq_{\mathcal{T}} D$ and $D \preceq_{\mathcal{T}} C$.

- Subsumption can be reduced to satisfiability since $C \preceq_{\mathcal{T}} D$ iff $C \sqcap \neg D$ is not satisfiable w.r.t. $\mathcal{T}$.

- Satisfiability can be reduced to ABox consistency since a concept C is satisfiable w.r.t. a TBox $\mathcal{T}$ iff the ABox $\{a : C\}$ is consistent w.r.t. $\mathcal{T}$.

Hence an algorithm for deciding ABox consistency can be used to decide all of the above reasoning problems. Such an algorithm is developed in Section 6.

The next section shows an application of $\mathcal{ALCRP}(\mathcal{D})$ using only restricted terminologies. Some examples for the reasoning problems introduced above are also given.

5 An Example Application

In [Baader and Hanschke 1991b], the expressiveness of $\mathcal{ALC}(\mathcal{D})$ is demonstrated using the predicate exists concept operator in order to formally represent the thirteen mutually disjoint relations that may hold between two time intervals as *concepts*. For instance, these concepts are used to prove the exhaustiveness of the relations (see [Allen 1983]). We will show how Allen's relations can be represented as *roles* using $\mathcal{ALCRP}(\mathcal{D})$'s role-forming predicate exists restrictions. In $\mathcal{ALCRP}(\mathcal{D})$-terminologies, concepts for temporal domain objects can be defined by quantifying over these roles. This cannot be done in $\mathcal{ALC}(\mathcal{D})$. In particular, unlike any other language we know of, $\mathcal{ALCRP}(\mathcal{D})$ supports universal quantification (value restrictions) over roles with very complex properties.

In order to represent temporal relations, we need an appropriate concrete domain. As the domain $\Delta_{\mathcal{D}}$ we choose the set of real numbers and as the

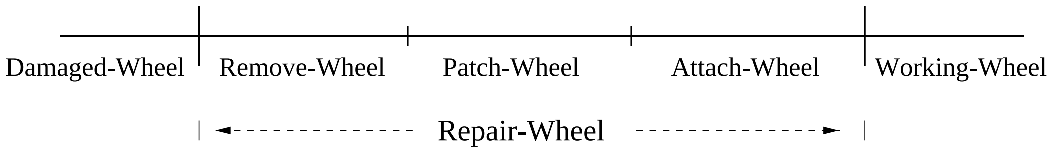

Figure 1: Visualization of the *Repair-Wheel* concept.

set of predicates we use the set of first order formulae (using junctors and quantifiers) over equalities and inequalities between linear integer polynomials. This concrete domain is discussed as an example for admissible concrete domains in [Baader and Hanschke 1991b]. More details on the decidability of this kind of formulae can be found in [Tarski 1951].

We can define an interval as $\mathit{interval} \doteq \exists \mathit{start}, \mathit{end}. \leq$. Allen's relations can then easily be defined:

$$\begin{aligned} \mathit{equal\text{-}p}(a,b,c,d) &: a-c=0 \wedge b-d=0 \\ \mathit{before\text{-}p}(a,b) &: b-a>0 \\ \mathit{meets\text{-}p}(a,b) &: a-b=0 \\ \mathit{equal} &\doteq \exists(\mathit{start,end})(\mathit{start,end}).\mathit{equal\text{-}p} \\ \mathit{before} &\doteq \exists(\mathit{end})(\mathit{start}).\mathit{before\text{-}p} \\ \mathit{meets} &\doteq \exists(\mathit{end})(\mathit{start}).\mathit{meets\text{-}p} \\ &\ldots \end{aligned}$$

The definition of all thirteen roles is given in Appendix B. Defined roles can be used in concept definitions. Let us consider a repair action of a car wheel. The action *Repair-Wheel* is composed of several subactions which are temporally related to one another. Figure 1 shows a visualization of the temporal structure induced by the concept *Repair-Wheel*. Let *remove*, *patch* and *attach* be features.

$$\begin{aligned} \mathit{Car\text{-}Repair} &\doteq \exists \mathit{met\text{-}by}.\mathit{Not\text{-}Working} \sqcap \exists \mathit{meets}.\mathit{Working} \\ \mathit{Repair\text{-}Wheel} &\doteq \exists \mathit{met\text{-}by}.\mathit{Damaged\text{-}Wheel} \sqcap \exists \mathit{meets}.\mathit{Working\text{-}Wheel} \sqcap \\ &\quad \exists \mathit{remove}.\mathit{Remove\text{-}Wheel} \sqcap \exists \mathit{patch}.\mathit{Patch\text{-}Wheel} \sqcap \\ &\quad \exists \mathit{attach}.\mathit{Attach\text{-}Wheel} \sqcap \\ &\quad \exists(\mathit{start}, \mathit{end}, \mathit{remove} \circ \mathit{start}, \mathit{remove} \circ \mathit{end}).\mathit{started\text{-}by\text{-}p} \sqcap \\ &\quad \exists(\mathit{start}, \mathit{end}, \mathit{patch} \circ \mathit{start}, \mathit{patch} \circ \mathit{end}).\mathit{contains\text{-}p} \sqcap \\ &\quad \exists(\mathit{start}, \mathit{end}, \mathit{attach} \circ \mathit{start}, \mathit{attach} \circ \mathit{end}).\mathit{finished\text{-}by\text{-}p} \sqcap \\ &\quad \exists(\mathit{remove} \circ \mathit{end}, \mathit{patch} \circ \mathit{start}).\mathit{meets\text{-}p} \sqcap \\ &\quad \exists(\mathit{patch} \circ \mathit{end}, \mathit{attach} \circ \mathit{start}).\mathit{meets\text{-}p} \end{aligned}$$

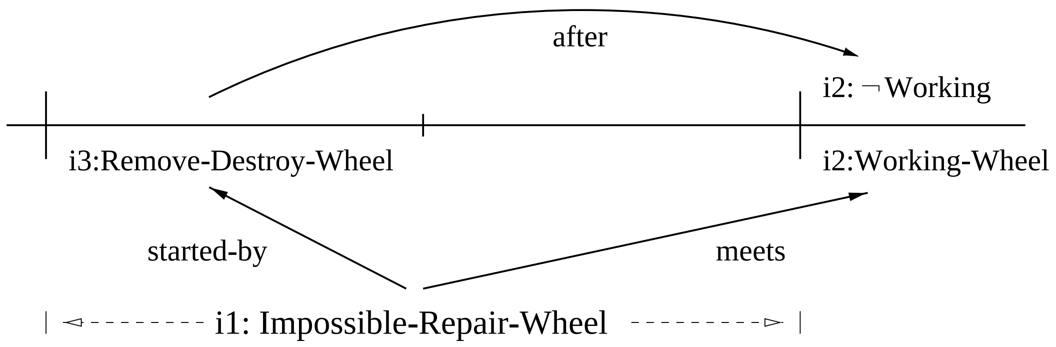

Figure 2: Unsatisfiability of the *Impossible-Repair-Wheel* concept.

In order to improve readability, attribute chains are written as $f_1 \circ f_2 \circ \cdots$ instead of $f_1 f_2 \cdots$ in this example. The two axioms form a restricted terminology since predicate exists restrictions are not used inside any quantifiers over complex roles and, furthermore, there are exists restrictions over complex roles but no value restrictions at all.

If we assume that *Damaged-Wheel* is subsumed by *Not-Working* and *Working-Wheel* is subsumed by *Working*, then the concept *Car-Repair* subsumes *Repair-Wheel*. The features *remove*, *patch*, and *attach* used in the definition of the concept *Repair-Wheel* are needed to "label" temporally related entities uniquely. It is possible to employ the predicate exists restriction operator to state temporal relations between entities labeled in this way. This modeling technique is required to specify the order of deattachment, patching and assembly of the wheel. Now let us consider the following concept definitions (the restrictedness criterion is fulfilled in the resulting terminology).

$$\begin{aligned}
\mathit{Remove\text{-}Destroy\text{-}Part} &\doteq \mathit{Remove\text{-}Part} \sqcap \\
&\quad \forall \mathit{meets}.\neg \mathit{Working} \sqcap \forall \mathit{after}.\neg \mathit{Working} \\
\mathit{Impossible\text{-}Repair\text{-}Wheel} &\doteq \mathit{Repair\text{-}Wheel} \sqcap \\
&\quad \forall \mathit{remove}.\mathit{Remove\text{-}Destroy\text{-}Part}
\end{aligned}$$

Remove-Destroy-Part describes the action of ultimately destroying the wheel during removal. *Impossible-Repair-Wheel* is an inconsistent concept. This can be seen as follows (see Figure 2). For any interval i_1 of type *Repair-Wheel* there has to be an interval i_2 that is met by i_1. Furthermore, i_2 has to be of type *Working-Wheel* and hence also of type *Working*. There also has to exist an interval i_3 of type *Remove-Wheel* that starts i_1. If i_1 is specialized to *Impossible-Repair-Wheel*, then i_3 has to be of type *Remove-Destroy-Part*.

Then the reasoner has to infer that i_2 is located temporally after i_3 and thus by definition of *Remove-Destroy-Part* i_2 has to be of type $\neg$ *Working*. But this is a contradiction since i_2 would have to be of both types *Working* and $\neg$ *Working*. Hence, *Impossible-Repair-Wheel* cannot have any instances.

In [Artale and Franconi 1997], a decidable temporal description logic is presented which contains Allen's relations. The logic incorporates a special operator that allows one to define complex temporal constraints between concepts. It does, however, not allow universal quantification over temporal roles. A good overview over existing temporal description logics can also be found there. Logic of time is a very lively field. Logics of all kinds (e.g. modal logics which are closely related to description logics, see [Schild 1991]) have been developed for temporal reasoning (see e.g. [Manna and Pnueli 1992]). Since $\mathcal{ALCRP}(\mathcal{D})$ is not primarily a temporal logic, we will not discuss related work from this field. In [Möller, Haarslev, and Lutz 1997], the applicability of restricted $\mathcal{ALCRP}(\mathcal{D})$ terminologies for spatial reasoning is demonstrated. In the next section we develop an algorithm for reasoning with restricted terminologies.

6 The Calculus

In this section, an algorithm for deciding ABox consistency w.r.t. restricted terminologies of $\mathcal{ALCRP}(\mathcal{D})$ is presented. In the next section, we prove soundness and completeness of the algorithm.

To work with a given pair of an ABox $\mathcal{A}$ and a TBox $\mathcal{T}$, it is necessary to unfold all concept-terms in $\mathcal{T}$. In Section 2, we have already seen that unfolding is always possible. It has to be noted that, in general, unfolding has exponential complexity (see [Nebel 1990]). Furthermore, it is necessary to convert all concept terms in the unfolded TBox into their equivalent ones in negation normal form (see Section 2 for details on the algorithm).

Let $\mathcal{A}_0$ be the ABox for which the consistency problem is to be decided. Assume that all concept names in $\mathcal{A}_0$ have been replaced by their defining terms in the unfolded TBox $\mathcal{T}$, which was previously converted to NNF. We use a standard tableau-based algorithm (see e.g. [Fitting 1996] and [Schmidt-Schauss and Smolka 1991]). Algorithms of this type are often called *semantic tableaux*. The algorithm works as follows. The ABox axioms are viewed as constraints on individual objects. Starting with the initial ABox $\mathcal{A}_0$, the algorithm iteratively applies completion rules to transform a given ABox into one or more descendant ABoxes. The descendant ABoxes are constructed by adding at least one additional constraint to the original ABox. The new constraints explicitly represent knowledge which was only "contained" implicitly in the ABox before the application of the rule. Sometimes there is more than one possible way to extend the original ABox. In this case, more than one descendant ABox is created by the application of a single rule. By iteratively applying rules, the algorithm thus produces a tree of ABoxes Υ. The applications of some rules also add new individuals to the corresponding ABoxes. Iterative rule application can be understood as an attempt to build a model for the initial ABox $\mathcal{A}_0$. The iteration continues until either an ABox is produced that is *complete*, i.e. to which no more rules are applicable, or all leaf ABoxes in the tree Υ contain a clash. In the former case, the complete ABox defines a model for the initial ABox $\mathcal{A}_0$, in the latter case, no model for $\mathcal{A}_0$ exists. We will formally introduce the notion of a clash below. Whether the algorithm terminates or not obviously depends on the set of completion rules used in the algorithm. We have to ensure that eventually one of the two cases mentioned above will occur.

Before the completion rules can be defined, we introduce some technical terms. Let $\mathcal{A}$ be an ABox, R be a role term, a and b be object names from O_A, γ be a symbol that is not element of O_D, u be a feature chain $f_1 \ldots f_k$, and let $u_1, \ldots, u_n$ and $v_1, \ldots, v_m$ (possibly with index) be arbitrary feature chains. For convenience we define three functions as follows:

$$\mathit{filler}_{\mathcal{A}}(a,u) := \begin{cases} x \text{ where } x \in \mathsf{O}_D \text{ such that} \\ \quad \exists b_1, \ldots, b_{k-1} \in \mathsf{O}_A : \\ \quad\quad ((a,b_1) : f_1 \in \mathcal{A}, \ldots, (b_{k-1},x) : f_k \in \mathcal{A}) \\ \gamma \text{ if no such } x \text{ exists.} \end{cases}$$

$$\begin{aligned} \mathit{createchain}_{\mathcal{A}}(a,x,u) := {} & \{(a,c_1) : f_1, \ldots, (c_{k-1},x) : f_k\} \\ & \text{where } c_1, \ldots, c_{k-1} \in \mathsf{O}_A \text{ are not used in } \mathcal{A}. \end{aligned}$$

$$\mathit{related}_{\mathcal{A}}(a,b,R) := \begin{cases} \mathit{true} & \text{if } (a,b) : R \in \mathcal{A} \\ \mathit{true} & \text{if } R \text{ is of the form } \exists(u_1, \ldots, u_n)(v_1, \ldots, v_m).P, \\ & \text{and } \exists x_1, \ldots, x_n, y_1, \ldots, y_m \in \mathsf{O}_D \text{ such that} \\ & \mathit{filler}_{\mathcal{A}}(a,u_1) = x_1, \ \ldots \ , \mathit{filler}_{\mathcal{A}}(a,u_n) = x_n, \\ & \mathit{filler}_{\mathcal{A}}(b,v_1) = y_1, \ \ldots \ , \mathit{filler}_{\mathcal{A}}(b,v_m) = y_m, \\ & (x_1, \ldots, x_n, y_1, \ldots, y_m) : P \in \mathcal{A} \\ \mathit{false} & \text{otherwise} \end{cases}$$

Let $\mathcal{A}$ be an ABox, f be a feature name, a, b, c be object names from O_A, and x, y be object names from O_D. If $\mathcal{A}$ contains the constraints $(a,b) : f$ and $(a,c) : f$ (resp. $(a,x) : f$ and $(a,y) : f$) then this pair of constraints is called a *fork* in $\mathcal{A}$. Since f is interpreted as a partial function, b and c (resp. x and y) have to be interpreted as the same objects. Each ABox is checked for forks immediately after a completion rule was applied. If a fork is detected, all occurrences of c in $\mathcal{A}$ are replaced by b (resp. y by x). Before any rule is applied to the initial ABox $\mathcal{A}_0$, any forks in $\mathcal{A}_0$ have to be eliminated. It is easy to prove that fork elimination preserves (in)consistency by showing that a model $\mathcal{I}$ for an ABox $\mathcal{A}$ is also a model for an ABox $\mathcal{A}'$ which is obtained from $\mathcal{A}$ by fork elimination.

Definition 9. The following *completion rules* will replace an ABox $\mathcal{A}$ by an ABox $\mathcal{A}'$ or by two ABoxes $\mathcal{A}'$ and $\mathcal{A}''$ (*descendants* of $\mathcal{A}$). In the following C and D denote concept terms, R denotes a role term, and P denotes a predicate name from $\Phi_{\mathcal{D}}$. Let $f_1, \ldots, f_n$ as well as $g_1, \ldots, g_n$ denote feature names, and $u_1, \ldots, u_m$ denote feature chains. a and b denote object names from O_A.

R$\sqcap$ The conjunction rule.
Premise: $a : C \sqcap D \in \mathcal{A}, \quad a : C \notin \mathcal{A} \vee a : D \notin \mathcal{A}$
Consequence: $\mathcal{A}' = \mathcal{A} \cup \{a : C,\ a : D\}$

R$\sqcup$ The disjunction rule.
Premise: $a : C \sqcup D \in \mathcal{A}, \quad a : C \notin \mathcal{A} \wedge a : D \notin \mathcal{A}$
Consequence: $\mathcal{A}' = \mathcal{A} \cup \{a : C\}$, $\mathcal{A}'' = \mathcal{A} \cup \{a : D\}$

R$\exists$C The concept exists restriction rule.
Premise: $a : \exists R.C \in \mathcal{A}, \quad \neg\exists b \in \mathsf{O}_A\colon (\mathit{related}_{\mathcal{A}}(a, b, R) \wedge b : C \in \mathcal{A})$
Consequence: $\mathcal{A}' = \mathcal{A} \cup \{(a, b) : R\ ,\ b : C\}$ where $b \in \mathsf{O}_A$ is not used in $\mathcal{A}$.
This rule may create a fork if R is a feature.

R$\forall$C The concept value restriction rule.
Premise: $a : \forall R.C \in \mathcal{A}, \quad \exists b \in \mathsf{O}_A\colon (\mathit{related}_{\mathcal{A}}(a, b, R), \wedge b : C \notin \mathcal{A})$
Consequence: $\mathcal{A}' = \mathcal{A} \cup \{b : C\}$

R$\exists$P The predicate restriction rule.
Premise: $a : \exists u_1, \ldots, u_n.P \in \mathcal{A}, \neg\exists x_1, \ldots, x_n \in \mathsf{O}_D\colon$
$(\mathit{filler}_{\mathcal{A}}(a, u_1) = x_1 \wedge \ldots \wedge \mathit{filler}_{\mathcal{A}}(a, u_n) = x_n \wedge$
$(x_1, \ldots, x_n) : P \in \mathcal{A})$
Consequence: $\mathcal{A}' = \mathcal{A} \cup \{(x_1, \ldots, x_n) : P\} \cup$
$\mathit{createchain}_{\mathcal{A}}(a, x_1, u_1) \cup \ldots \cup \mathit{createchain}_{\mathcal{A}}(a, x_n, u_n)$
where the $x_i \in \mathsf{O}_D$ are not used in $\mathcal{A}$.
This rule may create forks.

Rr$\exists$P The role-forming predicates restriction rule.
Premise: $(a, b) : \exists(u_1, \ldots, u_n)(v_1, \ldots, v_m).P \in \mathcal{A}$,
$\neg\exists x_1, \ldots, x_n, y_1, \ldots, y_m \in \mathsf{O}_D\colon$
$(\mathit{filler}_{\mathcal{A}}(a, u_1) = x_1 \wedge \ldots \wedge \mathit{filler}_{\mathcal{A}}(a, u_n) = x_n \wedge$
$\mathit{filler}_{\mathcal{A}}(b, v_1) = y_1 \wedge \ldots \wedge \mathit{filler}_{\mathcal{A}}(b, v_m) = y_m \wedge$
$(x_1, \ldots, x_n, y_1, \ldots, y_m) : P \in \mathcal{A})$

Consequence: $\mathcal{A}' = \mathcal{A} \cup \{(x_1, \ldots, x_n, y_1, \ldots, y_m) : P\} \cup$
$\mathit{createchain}_{\mathcal{A}}(a, x_1, u_1) \cup \ldots \cup \mathit{createchain}_{\mathcal{A}}(a, x_n, u_n) \cup$
$\mathit{createchain}_{\mathcal{A}}(b, y_1, v_1) \cup \ldots \cup \mathit{createchain}_{\mathcal{A}}(b, y_m, v_m)$
where the $x_i \in \mathsf{O}_D$ and $y_i \in \mathsf{O}_D$ are not used in $\mathcal{A}$.
This rule may create forks.

RChoose The choose rule.
Premise: $a : \forall(\exists(u_1, \ldots, u_n)(v_1, \ldots, v_m).P).C \in \mathcal{A}$,
$\exists b \in \mathsf{O}_A,\ x_1, \ldots, x_n, y_1, \ldots, y_m \in \mathsf{O}_D :$
$(\mathit{filler}_{\mathcal{A}}(a, u_1) = x_1 \wedge \ldots \wedge \mathit{filler}_{\mathcal{A}}(a, u_n) = x_n \wedge$
$\mathit{filler}_{\mathcal{A}}(b, v_1) = y_1 \wedge \ldots \wedge \mathit{filler}_{\mathcal{A}}(b, v_m) = y_m \wedge$
$(x_1, \ldots, x_n, y_1, \ldots, y_m) : P \notin \mathcal{A} \wedge$
$(x_1, \ldots, x_n, y_1, \ldots, y_m) : \overline{P} \notin \mathcal{A})$
Consequence: $\mathcal{A}' = \mathcal{A} \cup \{(x_1, \ldots, x_n, y_1, \ldots, y_m) : P\}$,
$\mathcal{A}'' = \mathcal{A} \cup \{(x_1, \ldots, x_n, y_1, \ldots, y_m) : \overline{P}\}$

Termination of the tableau algorithm using this set of rules is proven in the following section. The proof shows that after a finite number of rule applications a tree Υ of ABoxes is obtained for which one of the following conditions holds: (i) it contains an ABox $\mathcal{A}$ which is complete or (ii) all leaf ABoxes in the tree contain a clash. In both cases no more completion rules will be applied. In the following we will formalize the notion "to contain a clash" and then discuss the various transformation rules.

Definition 10. Let the same naming conventions be given as in Definition 9. Additionally, let f be a feature. An ABox $\mathcal{A}$ contains a *clash* if any of the following *clash triggers* are applicable:

Primitive Clash $a : C \in \mathcal{A}, \quad a : \neg C \in \mathcal{A}$

Feature Domain Clash $(a, x) : f \in \mathcal{A}, \quad (a, b) : f \in \mathcal{A}$

All Domain Clash $(a, x) : f \in \mathcal{A}, \quad a : \forall f.C \in \mathcal{A}$

Concrete Domain Clash $(x^{(1)}_1, \ldots, x^{(1)}_{n_1}) : P_1 \in \mathcal{A}, \ldots, (x^{(k)}_1, \ldots, x^{(k)}_{n_k}) : P_k \in \mathcal{A}$ and the corresponding conjunction $\bigwedge_{i=1}^{k} P_i(x^{(i)})$ is not satisfiable in $\mathcal{D}$. This can be decided because $\mathcal{D}$ is required to be admissible.

Most of the rules can also be found in algorithms for related languages such as $\mathcal{ALC}(\mathcal{D})$ and $\mathcal{ALC}$ (see [Baader and Hanschke 1991b] and [Schmidt-Schauss and Smolka 1991], respectively). The new rules are Rr∃P and RChoose. The use of the *related* function is also new and was necessary because the new role-forming operator was introduced. In the following we will exemplarily discuss some of the rules. We will give a detailed explanation of all novelties compared to the completion rules needed for $\mathcal{ALC}(\mathcal{D})$. Consider the rule R∀C. Let us assume there is an object a, for which the constraint $a : \forall R.C$ is in $\mathcal{A}$. We further assume the constraint $(a, b) : R$ is also in $\mathcal{A}$. Then we know by the semantics of the ∀ operator that the object b must be in the extension of the concept C and thus we can add the constraint $b : C$ to $\mathcal{A}$. But the rule R∀C is in fact a little more complicated since it uses the *related* function. This is necessary because not all role fillers might appear explicitly as constraints of the form $(a, b) : R$. If R is a complex role then b could be an R role filler of a, although there is no constraint of the above form. Two objects can also be related simply because a predicate holds over the concrete fillers of feature chains starting from a and b, respectively. This can be seen by considering the semantics of the role-defining operator. The two possible types of role fillers (explicit and implicit) are both captured by the *related* function.

The meaning of the rule Rr∃P is straightforward. If we know that two objects are related via a complex role, we can immediately create concrete objects as fillers of those feature chains being used in the definition of the complex role. We then also know that the predicate used in the definition of the role holds over the newly created concrete objects and thus can add an appropriate constraint. This is what Rr∃P does.

The meaning of RChoose is a little less obvious. *Choose rules* are used when, at a certain point of computation, each of several possibilities has to be examined. In this sense, R⊔ can also be seen as a choose rule. A choose rule is also needed when dealing with qualified number restrictions (see e.g. [Hollunder and Baader 1991]). In our case, the choose rule can be understood as follows. If a set of feature chains is used in the definition of a complex role R and (loosely spoken) there are the appropriate concrete fillers for two objects a and b for all the feature chains in this set, then b might be an R role filler of a. But unless there is an explicit constraint which states that the predicate P used in the definition of the role R holds (or does not hold), we do not know if this is really the case. So if there is no such constraint we have to try both alternatives and test whether P holds or does not hold. If any of these two alternatives is the wrong one, we will end up with a concrete domain clash in this branch of the ABox tree Υ. Like R⊔, an application of RChoose creates a branching in Υ.

The presence of branching rules like R⊔ and RChoose has been identified as a source of complexity [Donini, Lenzerini, Nardi, and Nutt 1995]. Schmidt-Schauss and Smolka (1991) show that the satisfiability problem for $\mathcal{ALC}$ is PSPACE-hard. Hence deciding ABox consistency for restricted terminologies of $\mathcal{ALCRP}(\mathcal{D})$ is at least PSPACE-hard because the language is a proper superset of $\mathcal{ALC}$ and deciding ABox consistency is a more difficult problem than deciding satisfiability. But like $\mathcal{ALC}(\mathcal{D})$, our language has at least one additional source of complexity: The test for concrete domain clashes. This task requires checking the satisfiability of finite conjunctions of predicates of the concrete domain (this is why a concrete domain is required to be admissible). The complexity of this test depends entirely on the concrete domain. This shows that there exist instantiations of $\mathcal{ALCRP}(\mathcal{D})$ for which deciding ABox consistency w.r.t. restricted terminologies is arbitrarily complex. It is yet unknown how the complexity of deciding ABox consistency w.r.t. restricted terminologies in $\mathcal{ALCRP}(\mathcal{D})$ can be derived from the complexity of deciding satisfiability of finite conjunctions of predicates of the concrete domain.

7 Soundness and Completeness

In this section we prove the soundness and completeness of the algorithm for deciding ABox consistency w.r.t. restricted terminologies in $\mathcal{ALCRP}(\mathcal{D})$, as it was defined in the previous section.

Proposition 11. *The algorithm presented in Section 6 is sound and complete.*

First, termination has to be proven. We explain the motivation for the definition of the restricted terminologies because the restrictions are reflected in the termination proof.

There exist a few decidable languages that can be used to construct concepts whose models are not necessarily trees (see $\mathcal{ALC}_{trans}$ [Baader 1991] and $\mathcal{TSL}$ [Schild 1991]). Deciding subsumption in these languages requires extensions to standard semantic tableaux algorithms to ensure termination. For other logics even the finite model property does not hold. Although there are decidable languages without finite model property (e.g. $\mathcal{FSL}$ with inverse roles [Schild 1991]), for other languages (e.g. MIRTL [Buongarzoni, Meghini, Salis, Sebastiani, and Straccia 1995]) it is unknown whether subsumption is decidable. Proving termination of decision algorithms is extremely difficult if the finite model property does not hold. For non-restricted TBoxes in $\mathcal{ALCRP}(\mathcal{D})$, subsumption is known to be undecidable (see [Lutz and Möller 1997]). Depending on the concrete domain, roles can be defined in such a way that concepts can be fulfilled only by infinite models. A tableau algorithm can be found which, in principle, would be sound and complete, but termination cannot be guaranteed due to the presence of infinite models. One possible way to overcome this problem is to restrict the structure of TBoxes such that unfolding will not produce any concepts that can be fulfilled only by infinite models. This is exactly what is achieved by defining the notion of a restricted terminology. The drawback of this approach is limited expressivity, i.e. modeling becomes harder if seen from the knowledge engineer's point of view.

A simple example for a concept only fulfilled by an infinite model in unrestricted terminologies of $\mathcal{ALCRP}(\mathcal{D})$ is $\exists \mathit{before}.\top \sqcap \forall \mathit{before}.\exists \mathit{before}.\top$, where *before* is one of Allen's relations as defined in Section 5. This concept cannot be defined in restricted terminologies. Let us see how the tableau algorithm would handle the concept above. All the new objects that are created by the R∃C rule have concrete objects attached via the *end* feature because the rule Rr∃P fires (see the definition of the *before* role in Section 5). The role *before* is a transitive (and irreflexive) role and thus an implicit role filler relationship is established between the original object and the newly created object (the predicate *before-p* holds for the corresponding concrete objects). This means the rule R∀C can be applied and in the following another new object is created. Thus, an infinite chain of objects is created and the algorithm does not terminate. The restrictions posed on restricted terminologies are constructed in such a way that beyond a certain point none of the new objects can have any concrete objects attached via feature chains as it happened with the *end* feature above. The restrictions ensure that any new objects being created beyond this point will not be connected to already existing objects via an implicit role filler relationship. Thus, we cannot run into a cycle as with the concept given above. The rules responsible for creating abstract objects with concrete fillers being attached via feature chains are R∃P and Rr∃P together with R∃C. Hence, it was the use of the concept forming predicate restriction and of exists and value restrictions[6] quantifying over complex roles that had to be restricted. The properties of models of restricted terminologies will become clear when, in the following, the termination proof is given.

Lemma 12. *The tree of ABoxes Υ computed by the algorithm presented in Section 6 has no infinite branch. Thus by König's Lemma the algorithm terminates in finite time.*

It will be shown that there can be no infinite sequence of ABoxes $\mathcal{A}_0, \mathcal{A}_1, \ldots$ where $\mathcal{A}_{k+1}$ is obtained from $\mathcal{A}_k$ by the application of a completion rule. Such a sequence corresponds to a branch in Υ that begins at the root of the tree. Before we can start the proof we need some definitions.

[6]Value restriction has to be considered because of the presence of general negation.

A constraint that is already present in the initial ABox $\mathcal{A}_0$ is called *old*. All other constraints are called *new*. We need some upper bounds for the number of concept terms and other entities that can appear during the computation. Let NC be the number of concept terms that are present directly or as subterms in $\mathcal{A}_0$. Let NF be the number of distinct attributes and $NP(n)$ be twice the number of n-ary distinct predicates occurring in $\mathcal{A}_0$. Twice because the negations may also be used during the computation. Let NR be NF plus the number of distinct atomic roles occurring in $\mathcal{A}_0$ plus the number of distinct role-forming operators in the initial ABox.

We define a graph $G_{\mathcal{A}} = (V_{\mathcal{A}}, E_{\mathcal{A}})$ for every ABox $\mathcal{A}$ as follows. For each abstract or concrete object o in $\mathcal{A}$, the graph contains a vertex $\nu(o)$. The graph contains an edge $\rho(ax)$ only for each *new* constraint ax that is of the form $(a, b) : R$. Please note that edges in the graph correspond to constraints of the above form where R may be atomic or complex. In the following we will (not quite accurately) use the term edge to refer to a constraint of the above form. There is a sequence of graphs $G_{\mathcal{A}_0}, G_{\mathcal{A}_1}, \ldots$ corresponding to the sequence of ABoxes $\mathcal{A}_0, \mathcal{A}_1, \ldots$. The graph $G_{\mathcal{A}_0}$ for the ABox $\mathcal{A}_0$ does not contain any edges because there are no new constraints at all in this ABox. The graph only grows if one of the rules R∃C, R∃P, or Rr∃P is applied, since only these rules add new objects and edges. The rules R∃C, R∃P, and Rr∃P are called *generating* rules. All other rules are called *non-generating*. The application of any of the generating rules may create forks. It may even be the case that only forks but no new objects (abstract or concrete) and no new edges are created. In these cases we will, without loss of generality, consider these rules as non-generating ones. An examination of the generating rules reveals that the graphs $G_{\mathcal{A}_i}$ in the sequence have the form of a forest (a collection of unconnected trees[7]): Each of the rules generates exactly one incoming edge for each new object. None of the rules generates any incoming edges for already existing objects. Each object in the ABox $\mathcal{A}_0$ is an initial tree consisting only of a root node. Since no rule introduces a new object without an incoming edge, the number of trees cannot grow by the application of completion rules. Hence, since the ABox $\mathcal{A}_0$ has to be finite, there is only a finite number of trees in any of the forests $G_{\mathcal{A}_i}$.

Lemma 13. *If the sequence of ABoxes $\mathcal{A}_1, \mathcal{A}_2, \ldots$ is infinite, then at least one of the trees in the corresponding sequence of forests $G_{\mathcal{A}_0}, G_{\mathcal{A}_1}, \ldots$ grows infinitely.*

Proof: An application of any of the generating rules leads to the growth of exactly one tree. Because there are only finitely many trees in the forests $G_{\mathcal{A}_i}$,

[7]These trees should not be confused with the tree of ABoxes Υ.

infinitely many applications of generating rules lead to the infinite growth of at least one of the trees. Hence, an infinite sequence of ABoxes that corresponds to a sequence of graphs in which no tree grows infinitely can only exist if it is possible to apply the non-generating rules an infinite number of times without applying a generating rule. Assume that this is the case for a given computation. Note that none of the non-generating rules generates objects (neither abstract nor concrete). Furthermore, every rule application adds a constraint to the ABox which was not already present. On the other hand, only finitely many distinct constraints can be asserted for a given, finite number of objects: For any single abstract object o, no more than NC distinct constraints can be asserted. For each pair of abstract objects, no more than NR distinct constraints can be asserted. For any pair of an abstract and a concrete object, no more than NF distinct constraints can be asserted. And last, for any n concrete objects, no more than $NP(n)$ distinct constraints can be asserted. This shows that there can only be finitely many applications of non-generating rules without adding new objects. That is a contradiction to the assumption and thus completes the proof. □

We now show that none of the trees in the forest sequence $G_{\mathcal{A}_0}, G_{\mathcal{A}_1}, \ldots$ can grow infinitely. From this and Lemma 13 it follows that the algorithm terminates in finite time.

Proposition 14. *In the sequence of forests $G_{\mathcal{A}_0}, G_{\mathcal{A}_1}, \ldots$, none of the trees grows infinitely.*

Proof: We show that each of the trees is finitely branching and has a finite depth. By König's Lemma it follows that the trees cannot grow infinitely. In the following, we can safely ignore concrete objects since an examination of the completion rules reveals that the nodes corresponding to these objects cannot have any successors.

Lemma 15. *There is an upper limit for the branching factor of each of the trees in each of the graphs $G_{\mathcal{A}_i}$.*

Proof: We have to examine each of the generating rules and show that it can only be applied finitely many times to a single object.

Let us first consider the R∃C rule. Let NE be the number of subterms of the form $\exists R.C$ (with R atomic or complex) which occur in the initial ABox $\mathcal{A}_0$. For each of these terms the R∃C rule may be applied at most once to a single object because the its premise is not fulfilled afterwards. Thus, there can exist at most NE successors of each node generated by the R∃C rule.

The same argumentation holds for the R∃P rule. But this rule may create more than one object per application. These objects are organized in chains and only a finite number of these chains are created per rule application because the operator only accepts a finite number of arguments. Thus, only finitely many successors per object are generated by this rule.

Now we consider the Rr∃P rule. Let NF be the number of feature names used in the initial ABox $\mathcal{A}_0$. Like R∃P (for which we could also have argued this way), Rr∃P only creates successors that are fillers of features. However, features may have at most one filler. If there is more than one filler, fork elimination immediately occurs and we can consider the Rr∃P rule as non-generating. Since the number of features is limited by NF, at most NF successors may be created per object by applications of the Rr∃P rule.

Summing up, any node in the tree has at most $NE + NF$ successors. □

Lemma 16. *There is an upper limit for the depth of each of the trees in each of the graphs $G_{\mathcal{A}_i}$.*

Proof: We show that there exists a number n such that the rule cannot generate any objects that correspond to nodes $\nu(o)$ with a depth greater than n. But first, the notion of a phantom edge is introduced. We say, there exists a *phantom edge* between two abstract objects in an ABox $\mathcal{A}$ iff there exists a complex role R in the initial ABox $\mathcal{A}_0$ such that $(a, b) : R$ is not in $\mathcal{A}$ but nevertheless $\mathit{related}_{\mathcal{A}}(a, b, R)$ holds (see chapter 6). In the following, when we refer to the *level* of an object o, we mean the distance between the node $\nu(o)$ and the root of the tree containing $\nu(o)$.

Let CD be the maximum nesting depth of any concept term in $\mathcal{A}_0$. Let CL be the maximum length of a feature chain present in $\mathcal{A}_0$. In order to prove Lemma 16, constraint propagation along the edges of the tree is examined. First, we assume there exist no phantom edges.

Let us suppose a concept term of the form $\exists R.C$ is propagated along the edges of the graph. The concept triggers the R∃C rule which generates new abstract objects. However, a concept $\exists R.C$ cannot be propagated to a level deeper than $CD - 1$ because to be propagated over an edge either the R∃C or the R∀C rule has to be applied and hence the nesting depth is decreasing by one for each propagation over an edge in the graph. The terms of the above form which are propagated along the graph edges can obviously not lead to the creation of any objects with a level greater than CD. This can easily be shown formally using an induction proof. When the R∃C rule is applied to a constraint of the above form on level $CD - 1$, it may, however, propagate a predicate exists restriction to an object on level CD. It may also be the case, that the incoming edge of the level CD object is a complex one. In both cases chains of abstract objects with a level greater than CD may be created by applying the rules R∃P and Rr∃P, respectively. But in either case the maximum length of the feature chains used in the according concept and role terms forms a limit for the maximum object level reachable in this way. Namely, no objects will be created on any level greater than $CD + CL$ (on the deepest level only concrete objects can be created). This is the maximum depth that may be reached without considering phantom edges.

But then, a phantom edge may exist between an object o_0 on level 0 and an abstract object o_1 on level $CD + CL - 1$ that was created as described above. Along this edge, a concept term C may be propagated to the object o_1. Let us analyze this concept term C. Its structure has a restricted form because of the properties of restricted terminologies. The syntactic structure of the concept term C is characterized formally by the nonterminal "EFreeCT" of the EBNF grammar in Appendix A. It has a maximum nesting depth of $CD - 1$ because it was already propagated over an edge by application of the R∀C rule. It may only contain predicate exists restrictions that are not contained inside any value or exists restrictions which are also subconcepts of C. The feature chains used as arguments in these predicate exists restrictions have a maximum length of 1. Thus, these predicate exists restrictions may only lead to the creation of concrete objects on level $CD + CL$ which may in turn lead to new incoming phantom edges for the object o_1. This is the case we are just examining. The concept C may also contain exists and value restrictions. Hence, starting from o_1, one or more chains of abstract objects with a maximum length of $CD - 1$ could be created.[8] But since C has a restricted form, there are no predicate exists restrictions propagated to levels deeper than the level of the object o_1. Moreover, no exists restrictions with complex roles occur as subconcepts in C. Hence none of the abstract objects in the chains being created will have any concrete objects attached via features because concrete objects are only created by the rules R∃P and Rr∃P together with R∃C. But this means that none of the objects in these chains can have any incoming phantom edges and thus there are no alternate paths (except the one we are just considering) over which concept terms can be propagated into the chains being created. We have just seen that by propagating constraints over phantom edges, chains of abstract objects with a length of at most $CD - 1$ may be attached to any abstract object on level $CD + CL - 1$.

To summarize, the maximum depth of any tree in the graph sequence $G_{\mathcal{A}_0}, G_{\mathcal{A}_1}, \ldots$ is $2 * CD + CL - 2$. $\square$

[8]The -1 results from the fact that the concept C has already been propagated over a phantom edge by a value restriction.

Now, after termination has been proven, we can finish the proof of Proposition 11. The proof idea is taken from [Baader and Hanschke 1991b]. The purpose of our algorithm is to decide whether a given ABox $\mathcal{A}$ is consistent. In order to prove soundness, we have to show that if the algorithm terminates because a complete ABox was computed, then $\mathcal{A}$ has a model. To prove completeness, we have to show that an ABox which contains a clash cannot have a model. We first need to establish a theorem known as the invariance theorem (see e.g. [Donini, Lenzerini, Nardi, and Nutt 1995]).

Theorem 17. *(Invariance)*

1. *If an ABox $\mathcal{A}'$ is obtained from an ABox $\mathcal{A}$ by a rule generating only a single descendant ABox, then $\mathcal{A}$ has a model iff $\mathcal{A}'$ has a model.*

2. *If two ABoxes $\mathcal{A}'$ and $\mathcal{A}''$ are obtained from an ABox $\mathcal{A}$ by a rule which generates two descendant ABoxes, then $\mathcal{A}$ has a model iff at least one of $\mathcal{A}'$ and $\mathcal{A}''$ has a model.*

Proof. This lemma has to be proven for each completion rule separately. We will only give the proof in detail for the R∃C rule. All other rules can be treated similarly. We have to prove two directions:

(i) Assume that an ABox $\mathcal{A}'$ was obtained from an ABox $\mathcal{A}$ by applying the R∃C rule to an axiom $a : \exists R.C$. Assume furthermore that $\mathcal{A}'$ is known to have a model $\mathcal{I}$. $\mathcal{A}'$ differs from $\mathcal{A}$ only in having two additional axioms $(a, o) : R$ and $o : C$, where o is a new individual. If $\mathcal{I}$ is considered as an interpretation for $\mathcal{A}$, then the fourth equation in Definition 6 is satisfied for the above axiom $a : \exists R.C$, although the two explicit axioms "enforcing" this are not present. The other axioms in $\mathcal{A}$ are not involved. Hence, $\mathcal{I}$ is also a model for $\mathcal{A}$. This illustrates the view that the application of a completion rule makes implicit knowledge explicit.

(ii) Assume that an ABox $\mathcal{A}$ that contains the axiom $a : \exists R.C$ is known to have a model $\mathcal{I}$. Assume furthermore that the R∃C rule has been applied to $\mathcal{A}$ yielding the ABox $\mathcal{A}'$. Then $\mathcal{A}'$ differs from $\mathcal{A}$ again in having two additional axioms $(a, o) : R$ and $o : C$, where o is a new individual. We are done if we can show that the model $\mathcal{I}$ of $\mathcal{A}$ can be extended to also satisfy these two axioms. That this is possible can be seen by looking at the fourth equation in Definition 6. Since the axiom $a : \exists R.C$ is satisfied the equation reveals that there exists an element b of $\Delta_{\mathcal{I}}$ for which the following holds: If we extend $\mathcal{I}$ to $\mathcal{I}'$ by setting $o^{\mathcal{I}} := b$, then the resulting interpretation $\mathcal{I}'$ is a model for $\mathcal{A}'$ because by equation four, it also satisfies the two new axioms. $\square$

We now prove the soundness of our tableau calculus.

Proposition 18. *(Soundness) When the algorithm terminates because a complete ABox $\mathcal{A}_c$ has been derived (i.e. an ABox to which no more completion rules are applicable) then the initial ABox $\mathcal{A}_0$ has a model.*

Proof. The complete ABox $\mathcal{A}_c$ is used to define an interpretation $\mathcal{I} = (\Delta_\mathcal{I}, \cdot^\mathcal{I})$ that is also a model:

1. $\Delta_\mathcal{I}$ consists of all the objects of O_A which occur in $\mathcal{A}_c$.
2. If C is a concept name then $a \in C^\mathcal{I}$ iff $a : C \in \mathcal{A}_c$.
3. If R is a role or feature name then $(a, b) \in R^\mathcal{I}$ iff $(a, b) : R \in \mathcal{A}_c$.
4. Because there is no concrete domain clash in $\mathcal{A}_c$, there is a variable assignment α that satisfies the conjunction of all occurring axioms $(x_1, \ldots, x_n) : P$. So we set $x^\mathcal{I} = \alpha(x)$ iff $x \in \mathsf{O}_D$.

Since $\mathcal{I}$ is obviously an interpretation function, it can be extended to arbitrary concept and role terms as defined in Definition 6. We have to show that $\mathcal{I}$ is also a model of the ABox $\mathcal{A}_c$, i.e. that it satisfies all assertional axioms in $\mathcal{A}_c$ in the sense of Definition 8. It is then an immediate consequence of Theorem 17 that $\mathcal{I}$ is also a model for $\mathcal{A}_0$. The proof can be done by induction on the size of the axioms ax in $\mathcal{A}_c$ as follows:

1. Axioms ax of the form $a : C$, where C is an atomic concept term, or of the form $(a, b) : R$, where R is an atomic role term, are satisfied by $\mathcal{I}$ by definition.
2. Let ax be of the form $a : \neg C$ (C can only be an atomic concept term), then the axiom $a : C$ is not in $\mathcal{A}_c$ since the primitive clash trigger is not applicable. Hence, ax is satisfied by $\mathcal{I}$.
3. Let ax be of the form $a : C \sqcap D$, then the rule R$\sqcap$ has been applied and the axioms $a : C$ and $a : D$ are also in $\mathcal{A}_c$. By induction, these axioms are also satisfied by $\mathcal{I}$ and hence ax is satisfied by $\mathcal{I}$.
4. Let ax be of the form $a : C \sqcup D$, then the rule R$\sqcup$ has been applied and at least one of the axioms $a : C$ or $a : D$ is also in $\mathcal{A}_c$ and, in turn, is by induction satisfied by $\mathcal{I}$. Hence, ax is satisfied by $\mathcal{I}$.
5. Let ax be of the form $a : \exists R.C$. Then the R$\exists$C rule has been applied and two axioms $(a, b) : R$ and $b : C$ are in $\mathcal{A}_c$. Both axioms are satisfied by induction. Thus, $\mathcal{I}$ satisfies ax.

6. Let ax be of the form $a : \exists u_1, \ldots, u_n.P$. Then the R∃P rule has been applied and an axiom $(x_1, \ldots, x_n) : P$ is in $\mathcal{A}_c$ and $u_i^{\mathcal{I}}(a) = x_i^{\mathcal{I}}$ holds for $i = 1, \ldots, n$. Hence by definition of α, $\mathcal{I}$ satisfies ax.

7. Let ax be of the form $a : \forall R.C$. First assume that R is an atomic or complex role and $(a, b) : R$ is in $\mathcal{A}_c$. Then b is in O_A because there is no all domain clash in $\mathcal{A}_c$. Then, for ax together with $(a, b) : R$ the rule R∀C has been applied. By induction, $\mathcal{I}$ satisfies $b : C$ for all these b and hence ax is satisfied. Now assume R is of the form $\exists(u_1, \ldots, u_n)(v_1, \ldots, v_m).P$ and $(a, b) : R$ is not in $\mathcal{A}_c$ but there exist objects $x_1, \ldots ,x_n$, $y_1, \ldots ,y_m$ in O_D such that $u_i^{\mathcal{I}}(a) = x_i^{\mathcal{I}}$ and $v_j^{\mathcal{I}}(b) = y_j^{\mathcal{I}}$ holds for $i = 1, \ldots, n$ and $j = 1, \ldots, m$, respectively (i.e. possibly there exists a phantom edge between a and b). Then b is in O_A because there are axioms of the form $(b, x) : f$. Furthermore, there have to be exactly one of the two axioms $(x_1, \ldots, x_n, y_1, \ldots, y_m) : P$ and $(x_1, \ldots, x_n, y_1, \ldots, y_m) : \overline{P}$ in $\mathcal{A}_c$, since the rule RChoose is not applicable and there is no concrete domain clash in $\mathcal{A}_c$. If the first of these two axioms is in $\mathcal{A}_c$, the rule R∀C has been applied. Like above, the axiom ax is satisfied by $\mathcal{I}$. In the other case the rule R∀C is not applicable to the objects a and b.

8. Let ax be of the form $(a, b) : \exists(u_1, \ldots, u_n)(v_1, \ldots, v_m).P$. Then the rule Rr∃P has been applied, the axiom $(x_1, \ldots, x_n) : P$ is in $\mathcal{A}_c$ and $u_i^{\mathcal{I}}(a) = x_i^{\mathcal{I}}$ as well as $v_j^{\mathcal{I}}(b) = y_j^{\mathcal{I}}$ holds for $i = 1, \ldots, n$ and $j = 1, \ldots, m$. By definition of α I satisfies ax as well. □

Proposition 19. *(Completeness) If the algorithm does not derive a complete ABox $\mathcal{A}_c$, then every leaf ABox in Υ contains a clash (see the termination proof). In this case, the initial ABox $\mathcal{A}_0$ does not have a model.*

Proof. Termination was already proven. We still have to show that when all leaves in the tree Υ contain a clash, then the initial A-Box $\mathcal{A}_0$ does not have a model. There are two possible cases. First, $\mathcal{A}_0$ itself could contain a clash which means that no descendant ABoxes have been computed. In this case, a look at the definition of the clash triggers reveals that $\mathcal{A}_0$ cannot have a model. In the second case, $\mathcal{A}_0$ does not contain a clash itself but all the leaf ABoxes in Υ do. Like $\mathcal{A}_0$ in the first case, none of the leaf ABoxes can have a model. In this case, using Theorem 17 it can easily be proven by induction that $\mathcal{A}_0$ cannot have a model as well.

8 Conclusions

$\mathcal{ALCRP}(\mathcal{D})$ is a concept language which supports the representation of abstract and concrete knowledge. It provides an operator for defining roles based on properties of objects. This allows one to define roles with very complex properties. $\mathcal{ALCRP}(\mathcal{D})$ was first introduced in [Lutz and Möller 1997]. It was proven that reasoning in $\mathcal{ALCRP}(\mathcal{D})$ is undecidable in the general case. In this paper we have shown that reasoning in $\mathcal{ALCRP}(\mathcal{D})$ is decidable if a restricted form of terminology is used.

Restricted terminologies are defined by posing constraints on the form of concept terms being used in a TBox. The constraints restrict the combinability of exists and value restrictions with defined roles and the use of the concept-forming predicate operator known from $\mathcal{ALC}(\mathcal{D})$. Although $\mathcal{ALCRP}(\mathcal{D})$ is more powerful than $\mathcal{ALC}(\mathcal{D})$, a knowledge engineer has to pay the price of dealing with more complex syntactic restrictions on terminologies. However, even with restricted terminologies, $\mathcal{ALCRP}(\mathcal{D})$ is still a powerful tool for representing conceptual knowledge in domains where complex relationships between concepts have to be represented. As an example, we have shown in this paper that, with the restricted formalism, Allen's temporal relations can be defined as roles which, in turn, can be used to define concepts. In [Möller, Haarslev, and Lutz 1997] the usefulness of restricted terminologies for reasoning in spatial domains is demonstrated. We expect that additional operators such as qualified number restrictions for atomic roles can be included without losing decidability yielding a yet more expressive language. We do not have a formal proof for this, however. The restricted combinability of operators can be seen from different perspectives. From the logical point of view, the restrictions are unusually complicated and not very elegant. The practical point of view is somewhat different. A knowledge engineer who uses an actual system which implements $\mathcal{ALCRP}(\mathcal{D})$ has to fully understand the restrictions to work efficiently. This may not be an easy task. On the other hand, a concrete implementation of $\mathcal{ALCRP}(\mathcal{D})$ can support a knowledge engineer in the process of defining a restricted terminology by labeling concepts with certain attributes indicating how these concepts may be used in new concept definitions.

With restricted terminologies of $\mathcal{ALCRP}(\mathcal{D})$, a decidable DL-formalism is presented that is capable of representing roles based on properties of objects. To the best of our knowledge, no other of the description logics defined until now offers this kind of expressivity. The presented formalism is very general since it can be parameterized with an arbitrary admissible concrete domain. In future work, we will further examine the relationship between necessary restrictions for terminologies and properties of concrete domain predicates being used in a certain language instantiation.

A EBNF Syntax Specification

Let in the following C, D, $\ldots$ be the elements of C. Let R_p denote an atomic role (it may also be a feature) and R_d denote a complex role. Let P denote a predicate, $f_1, \ldots, f_n$ denote features, and $u_1, \ldots, u_n$ denote feature chains.

ConceptTerm	::=	$C \mid D \mid \ldots$
ConceptTerm	::=	“$\neg$” ConceptTerm
ConceptTerm	::=	ConceptTerm (“$\sqcap$” $\mid$ “$\sqcup$”) ConceptTerm
ConceptTerm	::=	(“$\exists$” $\mid$ “$\forall$”) R_p “.” ConceptTerm
ConceptTerm	::=	“$\exists$” R_d “.” AFreeCT
ConceptTerm	::=	“$\forall$” R_d “.” EFreeCT
ConceptTerm	::=	“$\exists$” u_1 “,” $\ldots$ “,” u_n “.” P
AFreeCT	::=	$C \mid D \mid \ldots$
AFreeCT	::=	“$\neg$” EFreeCT
AFreeCT	::=	AFreeCT (“$\sqcap$” $\mid$ “$\sqcup$”) AFreeCT
AFreeCT	::=	(“$\exists$” $\mid$ “$\forall$”) R_p “.” AFCTBody
AFreeCT	::=	“$\exists$” R_d “.” AFCTBody
AFreeCT	::=	“$\exists$” f_1 “,” $\ldots$ “,” f_n “.” P
EFreeCT	::=	$C \mid D \mid \ldots$
EFreeCT	::=	“$\neg$” AFreeCT
EFreeCT	::=	EFreeCT (“$\sqcap$” $\mid$ “$\sqcup$”) EFreeCT
EFreeCT	::=	(“$\exists$” $\mid$ “$\forall$”) R_p “.” EFCTBody
EFreeCT	::=	“$\forall$” R_d “.” EFCTBody
EFreeCT	::=	“$\exists$” f_1 “,” $\ldots$ “,” f_n “.” P
AFCTBody	::=	$C \mid D \mid \ldots$
AFCTBody	::=	“$\neg$” EFCTBody
AFCTBody	::=	AFCTBody (“$\sqcap$” $\mid$ “$\sqcup$”) AFCTBody
AFCTBody	::=	(“$\exists$” $\mid$ “$\forall$”) R_p “.” AFCTBody
AFCTBody	::=	“$\exists$” R_d “.” BodyCT
EFCTBody	::=	$C \mid D \mid \ldots$
EFCTBody	::=	“$\neg$” AFCTBody
EFCTBody	::=	EFCTBody (“$\sqcap$” $\mid$ “$\sqcup$”) EFCTBody
EFCTBody	::=	(“$\exists$” $\mid$ “$\forall$”) R_p “.” EFCTBody
EFCTBody	::=	“$\forall$” R_d “.” BodyCT

B Allen's Relations as Roles

$$
\begin{aligned}
\mathit{equal\text{-}p}(a,b,c,d) &: a-c=0 \wedge b-d=0\\
\mathit{before\text{-}p}(a,b) &: b-a>0\\
\mathit{after\text{-}p}(a,b) &: a-b>0\\
\mathit{meets\text{-}p}(a,b) &: a-b=0\\
\mathit{overlaps\text{-}p}(a,b,c,d) &: c-a>0 \wedge b-c>0 \wedge d-b>0\\
\mathit{overlapped\text{-}by\text{-}p}(a,b,c,d) &: a-c>0 \wedge d-a>0 \wedge b-d>0\\
\mathit{during\text{-}p}(a,b,c,d) &: a-c>0 \wedge d-a>0 \wedge b-c>0 \wedge d-b>0\\
\mathit{contains\text{-}p}(a,b,c,d) &: c-a>0 \wedge b-c>0 \wedge d-a>0 \wedge b-d>0\\
\mathit{starts\text{-}p}(a,b,c,d) &: c-a=0 \wedge d-b>0\\
\mathit{started\text{-}by\text{-}p}(a,b,c,d) &: c-a=0 \wedge b-d>0\\
\mathit{finishes\text{-}p}(a,b,c,d) &: d-b=0 \wedge a-c>0\\
\mathit{finished\text{-}by\text{-}p}(a,b,c,d) &: d-b=0 \wedge c-a>0\\
\mathit{equal} &\doteq \exists(\mathit{start,end})(\mathit{start,end}).\mathit{equal\text{-}p}\\
\mathit{before} &\doteq \exists(\mathit{end})(\mathit{start}).\mathit{before\text{-}p}\\
\mathit{after} &\doteq \exists(\mathit{start})(\mathit{end}).\mathit{after\text{-}p}\\
\mathit{meets} &\doteq \exists(\mathit{end})(\mathit{start}).\mathit{meets\text{-}p}\\
\mathit{met\text{-}by} &\doteq \exists(\mathit{start})(\mathit{end}).\mathit{meets\text{-}p}\\
\mathit{overlaps} &\doteq \exists(\mathit{start},\mathit{end})(\mathit{start},\mathit{end}).\mathit{overlaps\text{-}p}\\
\mathit{overlapped\text{-}by} &\doteq \exists(\mathit{start},\mathit{end})(\mathit{start},\mathit{end}).\mathit{overlapped\text{-}by\text{-}p}\\
\mathit{during} &\doteq \exists(\mathit{start},\mathit{end})(\mathit{start},\mathit{end}).\mathit{during\text{-}p}\\
\mathit{contains} &\doteq \exists(\mathit{start},\mathit{end})(\mathit{start},\mathit{end}).\mathit{contains\text{-}p}\\
\mathit{starts} &\doteq \exists(\mathit{start},\mathit{end})(\mathit{start},\mathit{end}).\mathit{starts\text{-}p}\\
\mathit{started\text{-}by} &\doteq \exists(\mathit{start},\mathit{end})(\mathit{start},\mathit{end}).\mathit{started\text{-}by\text{-}p}\\
\mathit{finishes} &\doteq \exists(\mathit{start},\mathit{end})(\mathit{start},\mathit{end}).\mathit{finishes\text{-}p}\\
\mathit{finished\text{-}by} &\doteq \exists(\mathit{start},\mathit{end})(\mathit{start},\mathit{end}).\mathit{finished\text{-}by\text{-}p}
\end{aligned}
$$

References

Allen, J. (1983). Maintaining knowledge about temporal intervals. *Communications of the ACM 26*(11), 832–843.

Artale, A. and E. Franconi (1997). A temporal description logic for reasoning about actions and plans. Preprint.

Baader, F. (1991, August). Augmenting concept languages by transitive closure of roles: An alternative to terminological cycles. In *Twelfth International Conference on Artificial Intelligence, Darling Harbour, Sydney, Australia, Aug. 24-30, 1991*, pp. 446–451.

Baader, F. and P. Hanschke (1991a, August). A scheme for integrating concrete domains into concept languages. In *Twelfth International Conference on Artificial Intelligence, Darling Harbour, Sydney, Australia, Aug. 24-30, 1991*, pp. 452–457.

Baader, F. and P. Hanschke (1991b). A scheme for integrating concrete domains into concept languages. Technical Report DFKI-RR-91-10, German Center for AI (DFKI).

Buongarzoni, P., C. Meghini, R. Salis, F. Sebastiani, and U. Straccia (1995, June). Logical and computational properties of the description logic MIRTL. In A. Borgida et al. (Ed.), *Working notes of the 1995 International Workshop on Description Logics DL'97, Rome, Italy, June 1995*, pp. 80–84.

Donini, F., M. Lenzerini, D. Nardi, and W. Nutt (1995). The complexity of concept languages. Technical Report RR-95-07, German Center for AI (DFKI).

Egenhofer, M. (1991, August). Reasoning about binary topological relations. In O. Günther and H.-J. Schek (Eds.), *Advances in Spatial Databases, Second Symposium, SSD'91, Zurich, Aug. 28-30, 1991*, Volume 525 of *Lecture Notes in Computer Science*, pp. 143–160. Springer Verlag, Berlin.

Fitting, M. (1996). *First-Order Logic and Automated Theorem Proving.* Springer Verlag, Berlin.

Hanschke, P. (1996). *A Declarative Integration of Terminological, Constraint-based, Data-driven, and Goal-directed Reasoning.* Sankt Augustin: Infix.

Hollunder, B. and F. Baader (1991, April). Qualifying number restrictions in concept languages. In J. Allen, R. Fikes, and E. Sandewall (Eds.), *Second International Conference on Principles of Knowledge Representation, Cambridge, Mass., April 22-25, 1991*, pp. 335–346. A detailed version appeared as DFKI Research Report RR-91-03, Kaiserslautern.

Lutz, C. and R. Möller (1997, September). Defined topological relations in description logics. In M.-C. Rousset et al. (Ed.), *Proceedings of the International Workshop on Description Logics, DL'97, Sep. 27-29, 1997, Gif sur Yvette, France*, pp. 15–19. Universite Paris-Sud, Paris.

Manna, Z. and A. Pnueli (1992). *The Temporal Logic of Reactive and Concurrent Systems*. Springer Verlag, Berlin.

Möller, R., V. Haarslev, and C. Lutz (1997). Spatioterminological reasoning based on geometric inferences: The $\mathcal{ALCRP}(\mathcal{D})$ approach. Technical Report FBI-HH-M-277/97, University of Hamburg, Computer Science Department.

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

Nebel, B. (1991). Terminological cycles: Semantics and computational properties. In J. Sowa (Ed.), *Principles of Semantic Networks: Explorations in the Representation of Knowledge*, pp. 331–361. Morgan Kaufmann Publishers, San Mateo.

Randell, D., Z. Cui, and A. Cohn (1992, October). A spatial logic based on regions and connections. In B. Nebel, C. Rich, and W. Swartout (Eds.), *Principles of Knowledge Representation and Reasoning, Cambridge, Mass., Oct. 25-29, 1992*, pp. 165–176.

Schild, K. (1991, August). A correspondence theory for terminological logics: Preliminary report. In *Twelfth International Conference on Artificial Intelligence, Darling Harbour, Sydney, Australia, Aug. 24-30, 1991*, pp. 466–471.

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

Tarski, A. (1951). *A Decision Method for Elementary Algebra and Geometry*. University of California Press, Berkeley, CA.

Woods, W. and J. Schmolze (1992). The KL-ONE family. In F. Lehmann (Ed.), *Semantic Networks in Artificial Intelligence*, pp. 133–177. Pergamon Press, Oxford.