RWTH
LTCS–Report

Aachen University of Technology
Research group for
Theoretical Computer Science

The Complexity of Reasoning with Concrete Domains (Revised Version)

Carsten Lutz

LTCS-Report 99-01

This is an extended version of the article in: Proceedings of IJCAI-99, Stockholm, Sweden, July 31-August 6, Morgan Kaufmann Publ. Inc., San Mateo, CA, 1999

RWTH Aachen
LuFg Theoretische Informatik
http://www-lti.informatik.rwth-aachen.de

Ahornstr. 55
52074 Aachen
Germany

The Complexity of Reasoning with Concrete Domains (Revised Version)

Carsten Lutz
RWTH Aachen, LuFg Theoretical Computer Science
Ahornstr. 55, 52074 Aachen

July 21, 1999

Abstract

Description logics are knowledge representation and reasoning formalisms which represent conceptual knowledge on an abstract logical level. Concrete domains are a theoretically well-founded approach to the integration of description logic reasoning with reasoning about concrete objects such as numbers, time intervals or spatial regions. In this paper, the complexity of combined reasoning with description logics and concrete domains is investigated. We extend $\mathcal{ALC}(\mathcal{D})$, which is the basic description logic for reasoning with concrete domains, by the operators "feature agreement" and "feature disagreement". For the extended logic, called $\mathcal{ALCF}(\mathcal{D})$, an algorithm for deciding the ABox consistency problem is devised. The strategy employed by this algorithm is vital for the efficient implementation of reasoners for description logics incorporating concrete domains. Based on the algorithm, it is proved that the standard reasoning problems for both logics $\mathcal{ALC}(\mathcal{D})$ and $\mathcal{ALCF}(\mathcal{D})$ are PSPACE-complete - provided that the satisfiability test of the concrete domain used is in PSPACE.

1 Introduction

Description logics are formalisms for reasoning about conceptual knowledge on an abstract level. However, for a variety of applications, it is essential

to integrate the abstract knowledge with knowledge of a more concrete nature. Examples of such "concrete knowledge" include all kinds of numerical data as well as temporal and spatial information. Important application areas which have been found to depend on integrated reasoning with concrete knowledge are, e.g., mechanical engineering [7], reasoning about aggregation in databases [10], reasoning with physical laws [21] as well as temporal and spatial reasoning (see [16] and [24]). Description logic systems like, e.g., CLASSIC, $\mathcal{KRIS}$, and TAXON (see [11], [8], and [2], resp.), provide more or less elaborated interfaces that allow the attachment of external reasoning facilities which are capable of dealing with concrete information. Surprisingly, the complexity of combined reasoning with abstract and concrete knowledge has, to the best of our knowledge, never been formally analyzed and provably optimal algorithms have not been developed. Recent efficient implementations of expressive description logics like FACT (see [20]) concentrate on logics for which reasoning is "empirically tractable". The starting point for developing these efficient implementations are usually algorithms which are optimal w.r.t. worst case complexity. One important reason why these systems fail to integrate concrete knowledge is that no complexity results and no efficient algorithms are available.

Baader and Hanschke [6] introduce concrete domains as an approach to integrated reasoning with abstract and concrete knowledge. They define the basic description logic $\mathcal{ALC}(\mathcal{D})$, which can be parameterized with a concrete domain $\mathcal{D}$. A concrete domain defines a set of concrete objects and predicates over these objects. Baader and Hanschke prove that the standard reasoning problems concept satisfiability, concept subsumption and ABox consistency are decidable for the logic $\mathcal{ALC}(\mathcal{D})$ if an "admissible" concrete domain $\mathcal{D}$ (i.e., $\mathcal{D}$ fulfills a certain set of requirements) is used. However, to the best of our knowledge, the exact complexity of reasoning with $\mathcal{ALC}(\mathcal{D})$ has never been formally analyzed. The logic $\mathcal{ALC}(\mathcal{D})$ uses features (single-valued roles) to establish the connection between the abstract and the concrete domain. It does not, however, include two of the basic operators on features called feature agreement and feature disagreement. These operators first appeared in $\mathcal{ALCF}$, which is the basic extension of $\mathcal{ALC}$ with features [19].

In this paper, two issues are treated: First, $\mathcal{ALC}(\mathcal{D})$ is extended with the feature agreement and feature disagreement operators yielding the new logic $\mathcal{ALCF}(\mathcal{D})$, which is a combination of $\mathcal{ALC}(\mathcal{D})$ and $\mathcal{ALCF}$. Algorithms for deciding the concept satisfiability and ABox consistency problems for this logic are devised and their soundness and completeness is proved. The main

strategy employed by the algorithms is to divide the concrete domain satisfiability test into polynomial chunks. This technique is vital for efficient implementations of description logics incorporating concrete domains. Second, the complexity of reasoning with $\mathcal{ALCF}(\mathcal{D})$ and $\mathcal{ALC}(\mathcal{D})$ is examined and a tight complexity bound is established. Since deciding ABox consistency involves a satisfiability check for the concrete domain, the complexity of the combined formalism depends on the complexity of reasoning in the concrete domain. It is proved that concept satisfiability, concept subsumption and ABox consistency are PSPACE-complete for the description logics $\mathcal{ALC}(\mathcal{D})$ and $\mathcal{ALCF}(\mathcal{D})$, provided that the complexity of the concrete domain satisfiability test is in PSPACE. A lower complexity cannot be achieved since reasoning in $\mathcal{ALC}$, which is a proper subset of $\mathcal{ALC}(\mathcal{D})$, $\mathcal{ALCF}$, and $\mathcal{ALCF}(\mathcal{D})$, is already PSPACE-complete. The complexity results show that the proposed algorithms are optimal.

2 The Description Logic $\mathcal{ALCF}(\mathcal{D})$

In this section, the description logic $\mathcal{ALCF}(\mathcal{D})$ is introduced. The logic $\mathcal{ALCF}(\mathcal{D})$ extends $\mathcal{ALC}(\mathcal{D})$, as given in [6], by the operators feature agreement and disagreement (see [19]). First, concrete domains need to be defined.

Definition 1. 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 $\top_\mathcal{D}$ for $\Delta_\mathcal{D}$ and (2) the satisfiability of finite conjunctions of predicates is decidable.

On the basis of concrete domains, the syntax of $\mathcal{ALCF}(\mathcal{D})$ concepts can be formally introduced.

Definition 2. Let C, R, and F be disjoint sets of concept, role, and feature names[1]. A composition of features $f_1 f_2 \cdots f_n$ is called a *feature chain*. Any element of C is a *concept* (*atomic* concept). If C and D are concepts, R is a role or 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 concepts:

[1] In the following, the notion *role* (*feature*) is used synonymously for role name (feature name).

- $\neg C$ (negation), $C \sqcap D$ (conjunction), $C \sqcup D$ (disjunction), $\forall R.C$ (value restriction), $\exists R.C$ (exists restriction),
- $\exists u_1, \ldots, u_n.P$ (predicate operator)
- $u_1 \downarrow u_2$ (feature agreement), $u_1 \uparrow u_2$ (feature disagreement).

Please note that a simple feature can be viewed as a feature chain of length one. The predicate operator is written as $P(u_1, \ldots, u_n)$ in [6]. For a feature chain $u = f_1 \cdots f_n$, $\exists u.C$ and $\forall u.C$ will be used as abbreviations for $\exists f_1 \ldots \exists f_n.C$ and $\forall f_1 \ldots \forall f_n.C$, respectively. $\mathcal{ALLC}(\mathcal{D})$ concepts are $\mathcal{ALLCF}(\mathcal{D})$ concepts in which neither the feature agreement nor the feature disgreement operator appears. $\mathcal{ALLCF}$ concepts are $\mathcal{ALLCF}(\mathcal{D})$ concepts in which the predicate operator does not appear.

As usual, a set theoretic semantics is given. The semantics for $\mathcal{ALLCF}(\mathcal{D})$ is a combination of the semantics for $\mathcal{ALLC}(\mathcal{D})$ and $\mathcal{ALLCF}$.

Definition 3. 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 $\Delta_\mathcal{D}$ 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 $u = f_1 \cdots f_k$ is a feature chain, then $u^\mathcal{I}$ is defined as the composition $f_1^\mathcal{I} \circ \ldots \circ f_k^\mathcal{I}$ of the partial functions $f_1^\mathcal{I}, \ldots, f_k^\mathcal{I}$. Let the symbols C, D, R, P, and $u_1, \ldots, u_n$ be defined as in Definition 2. Then the interpretation function can be extended to complex concepts as follows:

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

$$(u_1{\downarrow}u_2)^{\mathcal{I}} := \{a \in \Delta_{\mathcal{I}} \mid \exists b \in \Delta_{\mathcal{I}}\colon (a,b) \in u_1^{\mathcal{I}} \wedge (a,b) \in u_2^{\mathcal{I}}\}$$
$$(u_1{\uparrow}u_2)^{\mathcal{I}} := \{a \in \Delta_{\mathcal{I}} \mid \exists b_1, b_2 \in \Delta_{\mathcal{I}}\colon b_1 \neq b_2 \wedge (a,b_1) \in u_1^{\mathcal{I}} \wedge (a,b_2) \in u_2^{\mathcal{I}}\}$$

An interpretation $\mathcal{I}$ is a *model* of a concept C iff $C^{\mathcal{I}} \neq \emptyset$. A concept C is *satisfiable* iff there exists a model $\mathcal{I}$ of C. A concept C *subsumes* a concept D (written $D \preceq C$) iff $D^{\mathcal{I}} \subseteq C^{\mathcal{I}}$ for all interpretations $\mathcal{I}$.

In the presence of negation, subsumption can be reduced to satisfiability since $D \preceq C$ iff the concept $D \sqcap \neg C$ is unsatisfiable.

Please note that the feature agreement and feature disagreement operators consider only objects from $\Delta_{\mathcal{I}}$ and no objects from $\Delta_{\mathcal{D}}$. If agreement and disagreement over concrete objects is desired, a concrete domain which includes an equality predicate has to be used. Feature agreement and disagreement over concrete objects, only, may then be expressed as $\exists u_1, u_2.\, =$ and $\exists u_1, u_2.\, \neq$, respectively. This can also be disjunctively combined with the feature agreement and disagreement operators to obtain "global" agreement and disagreement over both domains. Global agreement is expressed by the concept $u_1{\downarrow}u_2 \sqcup \exists u_1, u_2.\, =$ and global disagreement is expressed by $u_1{\uparrow}u_2 \sqcup \exists u_1, u_2.\, \neq \sqcup (\exists u_1.\top \sqcap \exists u_2.\top_{\mathcal{D}}) \sqcup (\exists u_2.\top \sqcap \exists u_1.\top_{\mathcal{D}})$, where $\top$ is an abbreviation for $A \sqcap \neg A$.[2]

In $\mathcal{ALCF}$, the additional operator $u{\uparrow}$ (feature undefinedness) is introduced [19]. Its semantics is

$$(u{\uparrow})^{\mathcal{I}} := \{a \in \Delta_{\mathcal{I}} \mid \neg\exists b \in \Delta_{\mathcal{I}}\colon (a,b) \in u^{\mathcal{I}}\}.$$

We do not consider this operator since it is obviously just syntactic sugar for $\forall u.\bot \sqcup \exists u.\top_{\mathcal{D}}$. Next, the assertional formalism is introduced.

Definition 4. Let O_D and O_A be disjoint sets of object names. Elements from O_D are called *concrete objects* while Elements from O_A are called *abstract objects*. If C is a concept, R a role or feature name, f a feature name,

[2] In fact, feature agreement and disagreement could also have been defined to take into account both abstract and concrete objects. This would have led to a language with exactly the same expressivity but would have made some technical issues more complicated. For example, the definition of a concrete domain $\mathcal{D}$ to be admissible would have had to be extended: $\Phi_{\mathcal{D}}$ would also be required to include an equality predicate. This approach was not followed because global agreement and disagreement are not considered to be very "natural" operators.

P a predicate name with arity n, a and b are elements of O_A and x, and x_1, ... ,x_n are elements of O_D, then the following expressions are *assertional axioms*.

$$a:C, \quad (a,b):R, \quad (a,x):f, \quad a \neq b, \quad (x_1,\ldots,x_n):P$$

A finite set of assertional axioms is called an $\mathcal{ALCF}(\mathcal{D})$ *ABox*. An *interpretation* for the concept language can be extended to the assertional language by mapping every object name from O_A to an element of $\Delta_\mathcal{I}$ and every object name from O_D to an element of $\Delta_\mathcal{D}$. The unique name assumption is not imposed, i.e. $a^\mathcal{I} = b^\mathcal{I}$ may hold even if a and b are distinct object names. An interpretation satisfies an assertional axiom

$$\begin{aligned} a:C &\text{ iff } a^\mathcal{I} \in C^\mathcal{I}, \\ (a,b):R &\text{ iff } (a^\mathcal{I}, b^\mathcal{I}) \in R^\mathcal{I}, \\ (a,x):f &\text{ iff } (a^\mathcal{I}, x^\mathcal{I}) \in f^\mathcal{I}, \\ a \neq b &\text{ iff } a^\mathcal{I} \neq b^\mathcal{I}, \\ (x_1,\ldots,x_n):P &\text{ iff } (x_1^\mathcal{I},\ldots,x_n^\mathcal{I}) \in P^\mathcal{D}. \end{aligned}$$

An interpretation is a *model* of an ABox $\mathcal{A}$ iff it satisfies all assertional axioms in $\mathcal{A}$. An ABox is *consistent* iff it has a model.
An object b is called a *successor* of an object a in an ABox $\mathcal{A}$ iff $\mathcal{A}$ contains an assertional axiom $(a,b):R$, where R is a role or feature.

An $\mathcal{ALC}(\mathcal{D})$ ABox is an $\mathcal{ALCF}(\mathcal{D})$ ABox in which only $\mathcal{ALC}(\mathcal{D})$ concepts are used. Analogously, an $\mathcal{ALCF}$ ABox is an $\mathcal{ALCF}(\mathcal{D})$ ABox in which only $\mathcal{ALCF}$ concepts are used.

Satisfiability of concepts, as introduced in Definition 3, can be reduced to ABox consistency since a concept C is satisfiable iff the ABox $\{a:C\}$ is consistent.

3 A Completion Algorithm

In this section, a completion algorithm is devised which can be used to decide the consistency of $\mathcal{ALCF}(\mathcal{D})$ ABoxes. Completion algorithms, which are also known as tableau algorithms, are characterized by a set of completion rules and a strategy to apply these rules to the assertional axioms of an ABox. The algorithm starts with an initial ABox $\mathcal{A}_0$ whose consistency is to be

decided. As noted before, if only the satisfiability of a single concept C is to be checked, the special ABox $\{a : C\}$ is considered. The algorithm repeatedly applies completion rules that add new axioms, and, by doing so, it makes all knowledge that is implicitly contained in the ABox explicit. If the algorithm succeeds to construct an ABox $\mathcal{A}_c$ which is complete (i.e., to which no more completion rules are applicable) and which does not contain an obvious contradiction, then $\mathcal{A}_c$ defines a canonical model for $\mathcal{A}_0$. Otherwise, $\mathcal{A}_0$ does not have a model. In fact, things are a little more difficult due to the presence of so-called branching rules. The application of a completion rule to an ABox $\mathcal{A}$ yields one or more succeeding ABoxes (descendants of $\mathcal{A}$). The rules for which more than one descendant per application is obtained are called branching rules. In the presence of branching rules, a completion algorithm creates a tree of ABoxes. Again, if a complete ABox is found which does not contain a contradiction, then this ABox defines a canonical model for $\mathcal{A}_0$.

In [19], it is proved that reasoning with $\mathcal{ALCF}$ is PSPACE-complete. A completion algorithm is employed for the proof. To the contrary, the complexity of reasoning with $\mathcal{ALC}(\mathcal{D})$ is yet unknown. Baader and Hanschke propose a completion algorithm for deciding the consistency of $\mathcal{ALC}(\mathcal{D})$ ABoxes without analyzing its complexity [6]. This algorithm is discussed in the following section.

3.1 Analyzing Baader and Hanschke's algorithm

In [6], Baader and Hanschke define a completion algorithm (from now on called "BHA") in order to demonstrate decidability of the consistency problem for $\mathcal{ALC}(\mathcal{D})$ ABoxes. The algorithm applies completion rules until a complete ABox is found (keeping the whole ABox in memory) and then performs a concrete domain satisfiability check on all axioms of the form $(x_1, \ldots, x_n) : P$ (*concrete domain axioms*) found during the rule application process. As will be discussed in this section, BHA may in the worst case generate ABoxes which are exponential in the size of the initial ABox.

A formal notion of "size" will be introduced later. For now, consider the size of a concept C to be the number of operators in C, and the size of an ABox $\mathcal{A}$ to be the sum of the sizes of all concepts used in assertional axioms in $\mathcal{A}$. When used with ABoxes that contain concepts according to the following schema, BHA generates a complete ABox that is exponential in the size of

the original ABox.

$$\begin{aligned}
&\exists R.C \sqcap \exists R.D \\
&\quad \sqcap \forall R.(\exists R.C \sqcap \exists R.D) \\
&\ldots \\
&\quad \sqcap \forall R^n.(\exists R.C \sqcap \exists R.D)
\end{aligned}$$

Here, $\forall R^n$ denotes n nested value restrictions over R. ABoxes can be seen as graphs where role successor relationships are edges and objects are nodes. In this sense, all models (complete ABoxes) that are generated by BHA are trees. Please note that concepts following the given schema are also satisfied by a cyclic (non-tree) model with only 2 domain objects. This is, however, not a general phenomenon. Halpern and Moses [17] show that for the modal logic K, there exist formulae that have models of exponential size only. Since it is well-known that the logic K_n is a notational variant of $\mathcal{ALC}$ ([30]), these formulae can be directly translated into $\mathcal{ALC}(\mathcal{D})$ concepts. On the other hand, Schmidt-Schauß and Smolka show that for the logic $\mathcal{ALC}$ (as for a number of other logics), algorithms can be devised which use only polynomial space for exploring exponentially sized models by performing depth-first search over the role successors and keeping only a "trace" of an ABox in memory [31]. If this technique is to be applied to $\mathcal{ALC}(\mathcal{D})$ or $\mathcal{ALCF}(\mathcal{D})$, an additional problem arises. BHA requires that all concrete domain axioms appearing in a complete ABox $\mathcal{A}$ are conjoined into one big conjunction and then checked for concrete domain satisfiability. The concepts obtained from the translation of the K formulae given by Halpern and Moses can easily be extended such that each abstract domain object in every model is in the extension of a concept of the form $\exists u_1, \ldots, u_n.P$. Hence, there are also exponentially many concrete domain axioms to be collected. It is obvious that any algorithm following this strategy needs exponential space in the worst case.

In the following, it will be shown that it is not necessary to collect all concrete domain axioms at once: The concrete domain satisfiability check can be partitioned into polynomial chunks which do not interact, i.e, which do not share any variables. In the next section, a tableau algorithm is developed that does exactly this. Only a polynomial trace of the ABox is kept in memory, and, furthermore, the concrete domain satisfiability check is broken down into independent, polynomial chunks. This algorithm is then used to prove that deciding $\mathcal{ALCF}(\mathcal{D})$ ABox consistency is PSPACE-complete provided that the

concrete domain satisfiability test is in PSPACE.

3.2 A PSPACE Algorithm

The algorithm for deciding the consistency of $\mathcal{ALCF}(\mathcal{D})$ ABoxes is developed in two steps. First, an algorithm sat for deciding concept satisfiability is devised. Afterwards, an algorithm ABox-cons is developed which is capable of deciding the consistency of $\mathcal{ALCF}(\mathcal{D})$ ABoxes.

Sat takes a concept C as input. C has to be in *negation normal form*, i.e., negation is allowed only in front of atomic concepts. Conversion to NNF can be done by exhaustively applying appropriate rewrite rules to push negation inwards.[3]

Lemma 5. *Let $\mathcal{D}$ be an admissible concrete domain. Let C, D be $\mathcal{ALC}(\mathcal{D})$ concepts, $\hat{R}$ a role, f a feature, P an n-ary predicate in $\Phi_{\mathcal{D}}$, and $u_1, \ldots, u_n$ feature chains. For a feature chain $u = f_1 \cdots f_k$, set*

$$\lambda(u) := \exists f_1.\top_{\mathcal{D}} \sqcup \exists f_1 f_2.\top_{\mathcal{D}} \sqcup \ldots \sqcup \exists f_1 \cdots f_{k-1}.\top_{\mathcal{D}}$$

The following transformations preserve equivalence of concepts:

$$\begin{aligned}
\neg(C \sqcap D) &\Longrightarrow \neg C \sqcup \neg D \qquad \neg(C \sqcup D) \Longrightarrow \neg C \sqcap \neg D \qquad \neg\neg C \Longrightarrow C \\
\neg(\forall \hat{R}.C) &\Longrightarrow \exists \hat{R}.\neg C \qquad \neg(\exists \hat{R}.C) \Longrightarrow \forall \hat{R}.\neg C \\
\neg(\forall f.C) &\Longrightarrow \exists f.\neg C \sqcup \exists f.\top_{\mathcal{D}} \qquad \neg(\exists f.C) \Longrightarrow \forall f.\neg C \sqcup \exists f.\top_{\mathcal{D}} \\
\neg(\exists u_1, \ldots, u_n.P) &\Longrightarrow \exists u_1, \ldots, u_n.\overline{P} \sqcup \forall u_1.\top \sqcup \ldots \sqcup \forall u_n.\top \\
&\quad \sqcup \lambda(u_1) \sqcup \ldots \sqcup \lambda(u_n) \\
\neg(u_1 \downarrow u_2) &\Longrightarrow u_1 \uparrow u_2 \sqcup \exists u_1.\top_{\mathcal{D}} \sqcup \exists u_2.\top_{\mathcal{D}} \sqcup \forall u_1.\bot \sqcup \forall u_2.\bot \\
&\quad \sqcup \lambda(u_1) \sqcup \lambda(u_2) \\
\neg(u_1 \uparrow u_2) &\Longrightarrow u_1 \downarrow u_2 \sqcup \exists u_1.\top_{\mathcal{D}} \sqcup \exists u_2.\top_{\mathcal{D}} \sqcup \forall u_1.\bot \sqcup \forall u_2.\bot \\
&\quad \sqcup \lambda(u_1) \sqcup \lambda(u_2)
\end{aligned}$$

By applying the above rules, any $\mathcal{ALCF}(\mathcal{D})$ concept can be converted into an equivalent concept in NNF in linear time.

In order to keep the further considerations simple, nondeterministic completion rules are used instead of branching rules. This means that in a

[3] In [6], the rewrite rule concerning the predicate operator is erroneous. This observation is due to Anni-Yasmin Turhan.

branching situation, the algorithm does not explore all of the given possibilities but just a single one. It is not specified which possibility is chosen. Thus, the described completion algorithm is a nondeterministic decision procedure. Such an algorithm accepts its input (i.e. returns *consistent*), if there is *any* way to make the nondeterministic decisions such that a positive result is obtained (i.e., a complete and non-contradictory ABox is found). A convenient way to think of nondeterministic rules is that they "guess" the "right" descendant, i.e., if there is a descendant which, if chosen, leads to a complete and non-contradictory ABox, then this descendant is in fact considered.

To decide the satisfiability of the concept C, sat starts with the initial ABox $\mathcal{A}_0 := \{o\!:\!C\}$ and then repeatedly applies completion rules. First, the set of completion rules is defined.

3.2.1 The Ruleset

To define the rules in a succinct way, the two functions $succ_{\mathcal{A}}$ and $chain_{\mathcal{A}}$ are introduced.

For an object $a \in \mathsf{O}_A$ and a feature chain u, $succ_{\mathcal{A}}(a, u)$ denotes the object b that can be found by following u starting from a in the ABox $\mathcal{A}$. If no such object exists, $succ_{\mathcal{A}}(a, u)$ denotes the special object ϵ that cannot be part of any ABox. An object name $a \in \mathsf{O}_A$ is called *fresh* in an ABox $\mathcal{A}$ if a is not used in $\mathcal{A}$. Let a be an object from O_A, x be an object from O_D, and $u = f_1 \cdots f_k$ be a feature chain. The function *chain* is defined as follows:

$$chain_{\mathcal{A}}(a, x, u) := \{(a, c_1)\!:\!f_1, \ldots, (c_{k-1}, x)\!:\!f_k\}$$
$$\text{where the } c_1, \ldots, c_{k-1} \in \mathsf{O}_A \text{ are distinct and fresh in } \mathcal{A}.$$

An ABox $\mathcal{A}$ is said to contain a *fork* (for a feature f) if it contains the two axioms $(a, b)\!:\!f$ and $(a, c)\!:\!f$ or the two axioms $(a, x)\!:\!f$ and $(a, y)\!:\!f$, where b and c are from O_A and x and y are from O_D. A fork can be *eliminated* by replacing all occurrences of c in $\mathcal{A}$ with b, or by replacing all occurrences of x in $\mathcal{A}$ with y, respectively. During rule application, it is assumed that forks are eliminated as soon as they appear (as an integral part of the rule application) with the proviso that newly generated object are replaced by older ones and not vice versa. Now, the set of completion rules can be formulated.

Definition 6. The following *completion rules* replace a given ABox $\mathcal{A}$ nondeterministically by an ABox $\mathcal{A}'$. $\mathcal{A}'$ is called a *descendant* of $\mathcal{A}$. In the following, C and D denote a concept, $\hat{R}$ a role, f a feature, P a predicate

name from $\Phi_{\mathcal{D}}$ with arity n, $u_1, \ldots, u_n$ feature chains, a and b object names from O_A, and $x_1, \ldots, x_n$ object names from O_D.

R⊓ The conjunction rule.
Premise: $a : C \sqcap D \in \mathcal{A}, \quad \{a : C,\ a : D\} \not\subseteq \mathcal{A}$
Consequence: $\mathcal{A}' = \mathcal{A} \cup \{a : C,\ a : D\}$

R⊔ The (nondeterministic) disjunction rule.
Premise: $a : C \sqcup D \in \mathcal{A}, \quad \{a : C,\ a : D\} \cap \mathcal{A} = \emptyset$
Consequence: $\mathcal{A}' = \mathcal{A} \cup \{a : C\} \ \vee \ \mathcal{A}' = \mathcal{A} \cup \{a : D\}$

Rr∃C The role exists restriction rule.
Premise: $a : \exists \hat{R}.C \in \mathcal{A}, \quad \neg \exists b \in \mathsf{O}_A : \{(a, b) : \hat{R},\ b : C\} \subseteq \mathcal{A}$
Consequence: $\mathcal{A}' = \mathcal{A} \cup \{(a, b) : \hat{R}\ ,\ b : C\}$ where $b \in \mathsf{O}_A$ is fresh in $\mathcal{A}$.

Rf∃C The feature exists restriction rule (may create forks).
Premise: $a : \exists f.C \in \mathcal{A}, \quad \neg \exists b \in \mathsf{O}_A : \{(a, b) : f,\ b : C\} \subseteq \mathcal{A}$
Consequence: $\mathcal{A}' = \mathcal{A} \cup \{(a, b) : f\ ,\ b : C\}$ where $b \in \mathsf{O}_A$ is fresh in $\mathcal{A}$.

Rr∀C The role value restriction rule.
Premise: $a : \forall \hat{R}.C \in \mathcal{A},\ \exists b \in O_A : (a, b) : \hat{R} \in \mathcal{A} \wedge b : C \notin \mathcal{A}$
Consequence: $\mathcal{A}' = \mathcal{A} \cup \{b : C\}$

Rf∀C The feature value restriction rule.
Premise: $a : \forall R.C \in \mathcal{A},\ \exists b \in O_A : (a, b) : f \in \mathcal{A} \wedge b : C \notin \mathcal{A}$
Consequence: $\mathcal{A}' = \mathcal{A} \cup \{b : C\}$

R∃P The predicate exists restriction rule (may create forks).
Premise: $a : \exists u_1, \ldots, u_n.P \in \mathcal{A}, \neg \exists x_1, \ldots, x_n \in \mathsf{O}_D :$
$(succ_{\mathcal{A}}(a, u_1) = x_1 \wedge \ \ldots \ \wedge succ_{\mathcal{A}}(a, u_n) = x_n \wedge$
$(x_1, \ldots, x_n) : P \in \mathcal{A})$
Consequence: $\mathcal{C}_0 := \mathcal{A} \cup \{(x_1, \ldots, x_n) : P\}$
where the $x_i \in \mathsf{O}_D$ are distinct and fresh in $\mathcal{A}$.
$\mathcal{C}_1 := chain_{\mathcal{C}_0}(a, x_1, u_1),\ \ldots\ , \mathcal{C}_n := chain_{\mathcal{C}_{n-1}}(a, x_n, u_n)$
$\mathcal{A}' = \bigcup_{i=0\ldots n} \mathcal{C}_i$

R↓ The agreement rule (may create forks).
Premise: $a : u_1 \downarrow u_2 \in \mathcal{A},$
$\neg \exists b \in \mathsf{O}_A : (succ_{\mathcal{A}}(a, u_1) = succ_{\mathcal{A}}(a, u_2) = b)$
Consequence: $\mathcal{C} = \mathcal{A} \cup chain_{\mathcal{A}}(a, b, u_1)$ where $b \in \mathsf{O}_A$ is fresh in $\mathcal{A}$.

$$\mathcal{A}' = \mathcal{C} \cup chain_{\mathcal{C}}(a, b, u_2)$$

R↑ The disagreement rule (may create forks).
Premise: $a : u_1 \uparrow u_2 \in \mathcal{A}, \quad \neg\exists b_1, b_2 \in \mathsf{O}_A : (succ_{\mathcal{A}}(a, u_1) = b_1 \wedge succ_{\mathcal{A}}(a, u_2) = b_2 \wedge b_1 \neq b_2 \in \mathcal{A})$
Consequence: $\mathcal{C} = \mathcal{A} \cup chain_{\mathcal{A}}(a, b_1, u_1)$
$\mathcal{A}' = \mathcal{C} \cup chain_{\mathcal{C}}(a, b_2, u_2) \cup \{b_1 \neq b_2\}$
where the $b_1, b_2 \in \mathsf{O}_A$ are distinct and fresh in $\mathcal{A}$.

Rule applications that generate new objects are called *generating*. All other rule applications are called *non-generating*. All applications of the Rr∃C rule are generating. Application of the rules Rf∃C, R∃P, R↓, R↑ are usually generating but may be non-generating if fork elimination takes place.

The ruleset is identical to the one used for BHA with three exceptions: (i) The rule R⊔ is nondeterministic; this serves the purpose of making our further considerations simpler; (ii) the rules Rr∃C and Rf∃C, as well as Rr∀C and Rf∀C, respectively, are unified in a single rule in Baader and Hanschke's ruleset; in our setting, it is more convenient to separate the rules since in the satisfiability algorithm to be defined, rule application to axioms of the form $(a, x) : f$, where f is a feature, occurrs at a different time than rule applications to axioms $(a, b) : \hat{R}$, where $\hat{R}$ is a role; (iii) there are two new rules for dealing with feature agreement and disagreement, see [19]. A formalized notion of contradictory and complete ABoxes needs to be introduced.

Definition 7. Let the same naming conventions be given as in Definition 6. An ABox $\mathcal{A}$ is called *concrete domain satisfiable* if there exists a mapping δ from O_D to $\Delta_{\mathcal{D}}$, such that $\bigwedge_{(x_1,\ldots,x_n):P \in \mathcal{A}} (\delta(x_1), \ldots, \delta(x_n)) \in P^{\mathcal{D}}$ is true in $\mathcal{D}$. An ABox $\mathcal{A}$ is called *contradictory* if it is not concrete domain satisfiable or one of the following *clash triggers* is applicable. If none of the clash triggers is applicable to an ABox $\mathcal{A}$, then $\mathcal{A}$ is called *clash-free*.

- *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*: $(a, x) : f \in \mathcal{A}, \quad a : \forall f . C \in \mathcal{A}$
- *Agreement clash*: $a \neq a \in \mathcal{A}$

An ABox to which no completion rules are applicable is called *complete*.

In the following section, the satisfiability algorithm is described.

3.2.2 The Satisfiability Algorithm

The satisfiability algorithm makes use of two auxiliary functions which will be described only informally. The function *apply* takes two arguments which are an ABox $\mathcal{A}$ and a completion rule r. It applies r once to arbitrary axioms from $\mathcal{A}$ matching r's premise and returns the new axioms generated by the rule application. The function satisfiable? takes as arguments a concrete domain $\mathcal{D}$ and a set $\mathcal{C}$ of concrete domain axioms. It returns *yes* if the conjunction of all axioms in $\mathcal{C}$ is satisfiable w.r.t. $\mathcal{D}$ and *no* otherwise. Assume that the satisfiability of a concept C is to be decided. Using the two auxiliary functions just defined, the satisfiability algorithm sat can be specified as follows.

Definition 8. The function sat can be used to decide the satisfiability of $\mathcal{ALCF}(\mathcal{D})$ concepts in NNF. To decide the satisfiability of the concept C, sat takes the input $\{o : C\}$.

define procedure sat($\mathcal{A}$)
  $\mathcal{A}' :=$ feature-complete($\mathcal{A}$)
  if $\mathcal{A}'$ contains a clash **then**
    return *inconsistent*
  $\mathcal{C} := \{\alpha \in \mathcal{A}' \mid \alpha$ is of the form $(x_1, \ldots, x_n) : P\}$
  if satisfiable?($\mathcal{D}$,$\mathcal{C}$) $=$ *no* **then**
    return *inconsistent*
  forall $a : \exists \hat{R}.D \in \mathcal{A}'$, where $\hat{R}$ is a role, **do**
    Let b be an object name from O_A.
    if sat($\{b : D\} \cup \{b : E \mid a : \forall \hat{R}.E \in \mathcal{A}'\}$) $=$ *inconsistent* **then**
      return *inconsistent*
  return *consistent*

define procedure feature-complete($\mathcal{A}$)
  while a rule r from $\{\mathrm{R}\sqcap, \mathrm{R}\sqcup, \mathrm{Rf}\exists\mathrm{C}, \mathrm{Rf}\forall\mathrm{C}, \mathrm{R}\exists\mathrm{P}, \mathrm{R}\downarrow, \mathrm{R}\uparrow\}$
      is applicable to $\mathcal{A}$ **do**
    $\mathcal{A} := \mathcal{A} \cup apply(\mathcal{A}, r)$
  return $\mathcal{A}$

We will now informally describe the strategy followed by sat. A formal proof of its soundness and completeness will be given in Section 4. The argument of sat is an ABox containing exactly one object $a \in \mathsf{O}_A$. Sat uses

the feature-complete function to create all feature successors of a, all feature successors of these feature successors and so on. Sat thus considers a cluster of objects which are related by features, only. If the resulting ABox is interpreted as a graph (see Section 3.1), the cluster is a directed acyclic graph with a single root a. Afterwards, a recursive call is made for each role successor of any object in the cluster. This strategy was first employed for $\mathcal{ALCF}$ reasoning algorithms (see [19]). Each cluster is checked separately for contradictions and concrete domain satisfiability. It will later be shown that this is equivalent to the strategy used by Baader and Hanschke's algorithm. For each recursive call, sat generates an ABox which contains all axioms for the respective successor. Please note that the generation of the new ABox corresponds to an application of the Rr$\exists$C rule and finitely many applications of the Rr$\forall$C rule. Sat is called recursively for the newly generated ABox.

To summarize, sat is a recursive function following a "trace" of object clusters. Based on the sat algorithm, an algorithm for deciding ABox consistency can be defined.

3.2.3 The ABox Consistency Algorithm

The algorithm ABox-cons, which is introduced in this section, can be used to decide the consistency of $\mathcal{ALCF}(\mathcal{D})$ ABoxes. The algorithm takes an ABox $\mathcal{A}$ as input, where all concepts appearing in $\mathcal{A}$ have to be in NNF. ABox-cons reduces the ABox consistency to concept satisfiability. It first performs preprocessing on the input ABox, then constructs a set of "reduction concepts" and finally checks their satisfiability using sat.

Definition 9. The algorithm ABox-cons which can be used to decide the consistency of $\mathcal{ALCF}(\mathcal{D})$ ABoxes is defined as follows.

define procedure ABox-cons($\mathcal{A}$)
eliminate forks in $\mathcal{A}$ (see Definition 6)
$\mathcal{A}$:= preprocess($\mathcal{A}$)
$\mathcal{C} := \{\alpha \in \mathcal{A} \mid \alpha$ is of the form $(x_1, \ldots, x_n):P\}$
if $\mathcal{A}$ contains a clash **then**
  return *inconsistent*
if satisfiable?($\mathcal{D}$,$\mathcal{C}$) = *no* **then**
  return *inconsistent*
forall $a:\exists\hat{R}.C \in \mathcal{A}$, where $\hat{R}$ is a role, **do**
  Let b be an object name from O_A.

if sat$(\{b\!:\!(C \sqcap (\sqcap_{a:\forall \hat{R}.D \in \mathcal{A}} D))\}) = \mathit{inconsistent}$ **then**
return *inconsistent*
return *consistent*

define procedure preprocess($\mathcal{A}$)
while a rule r from $\{\mathrm{R}\sqcap, \mathrm{R}\sqcup, \mathrm{Rr}\forall\mathrm{C}, \mathrm{Rf}\exists\mathrm{C}, \mathrm{Rf}\forall\mathrm{C}, \mathrm{R}\exists\mathrm{P}, \mathrm{R}\downarrow, \mathrm{R}\uparrow\}$
is applicable to $\mathcal{A}$, **do**
$\mathcal{A} := \mathcal{A} \cup \mathit{apply}(\mathcal{A}, r)$
return $\mathcal{A}$

Again, an informal description of the algorithm is given. The preprocessing is performed by the function preprocess, which is very similar to the feature-complete function used by sat. The rule set used by preprocess is identical to that used by feature-complete, except that the Rr$\forall$C rule is also applied. This is necessary because in the initial ABox $\mathcal{A}_0$, there may already be axioms of the form $(a, b)\!:\!\hat{R}$, where $\hat{R}$ is a role. After the resulting ABox, which is called preprocessing complete, has been checked for contradictions and concrete domain satisfiability, it is sufficient to separately check all successors of objects in the preprocessing complete ABox for consistency. This is done by constructing a set of "reduction concepts" and using sat to decide their consistency. Please note that concept construction corresponds to the application of the Rr$\exists$C and Rr$\forall$C rules. Regarding preprocessing on ABoxes with the goal to reduce ABox consistency to concept satisfiability, see also [18] and [9].

The correctness of the reduction implemented by ABox-cons is proved in the next section.

4 Correctness and Complexity

In this section, it is proved that both the satisfiability algorithm and the consistency algorithm are sound and complete and that they can be executed using only polynomial space provided that the satisfiable? function can also be executed in polynomial space. Starting from this result, it will be straightforward to prove that deciding the consistency of an $\mathcal{ALCF}(\mathcal{D})$ ABox is a PSPACE-complete problem provided that the satisfiability test for the concrete domain $\mathcal{D}$ is in PSPACE. Sat and ABox-cons are considered separately.

4.1 The sat Algorithm

First, some basic definitions are necessary. To characterize space requirements, a formal notion for the size of an ABox is given.

Definition 10. The *size* $||C||$ of a concept C is defined inductively. Let C and D be concepts, A an atomic concept, R a role or feature, $u = f_1 \cdots f_k$ a feature chain, and let $u_1, \ldots, u_n$ also be feature chains.

$$\begin{aligned} ||A|| &= 1 & ||f_1 \cdots f_k|| &= k \\ ||C\{\sqcap, \sqcup\}D|| &= ||C|| + ||D|| + 2 & ||\neg C|| &= ||C|| \\ ||\exists u_1, \ldots, u_n.P|| &= ||u_1|| + \cdots + ||u_n|| + 1 & ||\{\forall, \exists\}R.C|| &= ||C|| + 1 \\ ||u_1 \downarrow u_2|| &= ||u_1|| + ||u_2|| & ||u_1 \uparrow u_2|| &= ||u_1|| + ||u_2|| + 1 \end{aligned}$$

The size of an axiom α is $||C||$ if α is of the form $x : C$ and 1 otherwise. The size of an ABox $\mathcal{A}$ is the sum of the sizes of all axioms in $\mathcal{A}$.

Please recall that sat is a nondeterministic algorithm, i.e., the sat yields a positive result if there is any way to make the nondeterministic decisions such that a positive result is obtained. A way to make the nondeterministic corresponds to a run of the algorithm. Correctness of the satisfiability algorithm can be proved by showing that (1) whenever there is a sat run returning *consistent*, then the initial ABox $\mathcal{A}_0 = \{a : C\}$ has a model, (2) whenever all possible sat runs are returning *inconsistent*, then $\mathcal{A}_0$ cannot have a model, and (3) sat terminates on any input $\{a : C\}$, i.e., all possible runs are of finite length. For doing so, it is convenient to define a sequence of ABoxes $\mathcal{A}^0_\cup, \mathcal{A}^1_\cup, \ldots$ that is associated with a given run of the satisfiability algorithm. The ABoxes $\mathcal{A}^i_\cup$ collect all axioms that the sat algorithm generates during the run. Each ABox $\mathcal{A}^i_\cup$ is obtained by the application of a single rule to the ABox $\mathcal{A}^{i-1}_\cup$. It will be shown that if sat terminates after n rule applications returning *consistent*, then the ABox $\mathcal{A}^n_\cup$ defines a canonical model for $\mathcal{A}_0$.

When defining $\mathcal{A}^i_\cup$, we must cope with the following technical problem: The object names created by sat are unique only within the ABox considered in a single recursion step. This means that we must ensure that an object x in one recursion step can be distinguished from x in another step since these two objects are not identical. To achieve this, objects used in axioms are renamed before the axioms are added to an ABox $\mathcal{A}^i_\cup$. For this purpose, the object names are indexed with the value of a counter sc, which counts the (recursive) calls to the sat function.

```
* rc := sc := 0

define procedure sat(A)
        A' := feature-complete(A)
        if A' contains a clash then
          return inconsistent
        C := {α ∈ A' | α is of the form (x_1, ..., x_n):P}
        if satisfiable?(D,C) = no then
          return inconsistent
        forall a:∃R̂.D ∈ A', where R̂ is a role, do
*               sc := sc + 1
*               rc := rc + 1
                Let b be an object name from O_A.
*               A_∪^rc := A_∪^(rc-1) ∪ {(a_(sc-1), b_sc):R̂} ∪ {b_sc:D}
*               forall α ∈ A' of the form a:∀R̂.E do
*                       rc := rc + 1
*                       A_∪^rc := A_∪^(rc-1) ∪ {b_sc:E}
                if sat({b:D} ∪ {b:E | a:∀R̂.E ∈ A'}) = inconsistent then
                  return inconsistent
        return consistent

define procedure feature-complete(A)
        while a rule r from {R⊓, R⊔, Rf∃C, Rf∀C, R∃P, R↓, R↑}
                            is applicable to A do
                N := apply(A, r)
                A := A ∪ N
*               rc := rc + 1
*               forall α ∈ N do
*                       forall a ∈ O_A (and all x ∈ O_D) used in α do
*                               Replace each occurrence of a (resp. x)
*                                                   in α with a_sc (resp. x_sc)
*                       A_∪^rc := A_∪^(rc-1) ∪ {α}
        return A
```

Figure 1: The annotated sat algorithm.

Let $\mathcal{A}_0 = \{a : C\}$ be the ABox that is initially passed to sat. We set $\mathcal{A}^0_\cup := \{a_0 : C\}$ for any sat run with $\mathcal{A}_0$ as initial argument. For $i > 0$, $\mathcal{A}^i_\cup$ is defined recursively by the annotated version of the sat algorithm given in Figure 1. The annotations are marked with asterisks. The annotated version introduces the two global variables *sc* and *rc*, which are assumed to be initialized with the value 0. The first one is a counter for the number of calls to the sat function. The second one counts the number of rules that have been applied. Please note that the annotated version of sat is defined just to prove the correctness of the original version. It is by no means intended to be used as an algorithm for deciding the satisfiability of concepts, neither do we claim that the annotated version itself can be executed in polynomial space. The following Lemma is needed for proving the correctness of the sat algorithm.

Lemma 11. *Let $\mathcal{A}_0$ be an input to the* sat *function. Fix a run τ of* sat *on $\mathcal{A}_0$. Let $\mathcal{A}^0_\cup, \mathcal{A}^1_\cup, \ldots$ be the sequence of ABoxes that is associated with τ. If the run τ terminates then this sequence is of finite length n. In this case, let $\mathcal{A}_\cup$ denote the ABox $\mathcal{A}^n_\cup$.*

1. *Let $\mathcal{A}$ be an ABox. For each completion rule r that can be applied to $\mathcal{A}$, we have that $\mathcal{A}$ has a model if and only if one of the descendants of $\mathcal{A}$ has a model.*[4]

2. *If τ terminates and returns inconsistent, then $\mathcal{A}_\cup$ does not have a model.*

3. *If τ terminates and returns consistent, then $\mathcal{A}_\cup$ has a model.*

4. Sat *terminates on any input.*

The four claims of the Lemma are proved separately:

(1) One direction is trivial: Since all descendants $\mathcal{A}'$ generated by rule application are supersets of their ancestor $\mathcal{A}$, it follows immediately that each model of $\mathcal{A}'$ is also a model of $\mathcal{A}$. The other direction depends on the respective rule and is straightforward in most cases. A formal proof is omitted since the rules used by the sat algorithm have been used in [6] and [19] and

[4] In case of the nondeterministic rule R$\sqcup$, there are two possible descendants. For all other rules, there is only a single descendant.

the property in question ("local correctness") follows from the correctness proofs for the algorithms that can be found in the referred papers.[5]

(2) The run τ of sat returns *inconsistent* if and only if an ABox $\mathcal{A}$ is considered which either contains a clash or is not concrete domain satisfiable. In the former case, $\mathcal{A}_\cup$ does also contain a clash; in the latter case, $\mathcal{A}_\cup$ cannot be concrete domain satisfiable. This follows trivially from the fact that we have $\mathcal{A} \subseteq \mathcal{A}_\cup$ modulo object renaming. The rest of the proof is straightforward, again. It suffices to show that no interpretation can satisfy an ABox to which a clash trigger is applicable or which is not concrete domain satisfiable.

(3) First, the following Lemma needs to be established.

Lemma 12. *If a* sat *run* τ *returns consistent, then* $\mathcal{A}_\cup$ *is complete, clash-free, and concrete domain satisfiable.*

Proof: In the following, the *i'th recursion step* of τ means the recursion step in which the counter *sc* has the value i. We say that i *calls* j if in the i'th recursion step of τ, a recursive call is made in which the counter *sc* has the value j. Please note that $j = i + 1$ does not need to hold.

- completeness. Assume that there exists a set $\mathcal{C}$ of axioms from $\mathcal{A}_\cup$ to which a completion rule r is applicable. It will be shown that a contradiction can be derived from this assumption. A case analysis according to the rules has to be made. First assume the rule r to be from $\{\mathrm{R}\sqcap, \mathrm{R}\sqcup, \mathrm{Rf}\exists\mathrm{C}, \mathrm{Rr}\exists\mathrm{C}, \mathrm{R}\exists\mathrm{P}, \mathrm{R}\downarrow, \mathrm{R}\uparrow\}$. In this case, $\mathcal{C}$ has the form $\{a_i : C\}$. From the construction of $\mathcal{A}_\cup$, it follows that there exists an axiom $\alpha = a : C$ that either is created by feature-complete during the i'th recursion step or that is contained in the ABox which is the argument to the i'th recursion step of τ. The latter case happens if either α is the axiom from the initial ABox $\mathcal{A}_0$ and $i = 0$ or if α was introduced by the implicit application of Rr$\exists$C and Rr$\forall$C during a j'th recursion step of τ, where j calls i. In all of the cases mentioned, r is checked for applicability to an ABox $\mathcal{A}$ containing α during the i'th recursion step. If r is Rr$\exists$C, this is done by sat itself; in the other cases, this is done by feature-complete. Assume first that r is not applicable

[5] To be precise, the rules presented in this paper differ in two points from those given in [6], as was discussed in Section 3.2.1. It is obvious that these two differences (nondeterminism of R$\sqcup$ and separateness of Rr$\exists$C/Rf$\exists$C and Rr$\forall$C/Rf$\forall$C) do not influence local correctness.

to α in $\mathcal{A}$. This can only be the case if the axioms appearing in the consequence of the rule are already part of $\mathcal{A}$. But since we have $\mathcal{A} \subseteq \mathcal{A}_\cup$ modulo object renaming, this would clearly contradict the assumption that r is applicable to $\mathcal{C}$ in $\mathcal{A}_\cup$. Now assume that r is applied to α in $\mathcal{A}$ during the i'th recursion step. This means that the axioms appearing in the consequence of the rule are in $\mathcal{A}_\cup$ (modulo object renaming). This again contradicts the assumption.

The remaining cases for Rf$\forall$C and Rr$\forall$C can be proved similarly. $\mathcal{C}$ now contains the additional axiom $(a_i, b_i) : f$ or $(a_i, b_{i+k}) : \hat{R}$, respectively, where f is a feature and $\hat{R}$ a role. Each axiom $(a_i, b_i) : f$ in $\mathcal{A}_\cup$ corresponds to an axiom $(a, b) : f$ that was introduced by feature-complete during the i'th recursion step to an ABox $\mathcal{A}$. Each axiom $(a_i, b_{i+k}) : \hat{R}$ in $\mathcal{A}_\cup$ corresponds to an axiom $(a, b) : \hat{R}$ that was introduced by an implicit application of the Rr$\exists$C rule during recursion step i. Having established these facts, contradiction proofs exactly analogous to the one given above can be employed.

- clash-freeness. Assume that $\mathcal{A}_\cup$ contains a clash. A case distinction according to the clash types need to be made. Since the cases are very similar, only one case will be discussed exemplarily: $\mathcal{A}_\cup$ contains an all domain clash if it contains two axioms $(a_i, x_i) : f$ and $a_i : \forall f.C$. The axiom $(a_i, x_i) : f$ corresponds to an axiom $(a, x) : f$ introduced by feature-complete during the i'th recursion step of τ. The axiom $a_i : \forall f.C$ corresponds to an axiom $a : \forall f.C$ that was introduced (i) during the i'th recursion step by feature-complete or (ii) by the implicit application of the Rr$\forall$C or Rr$\exists$C rule during a recursion step j, where j calls i, or (iii) that is the axiom contained in the initial ABox $\mathcal{A}_0$ and we have $i = 0$. In any case, an ABox $\mathcal{A}$ containing both $(a, x) : f$ and $a : \forall f.C$ was checked for the applicability of clash triggers during the i'th recursion step of τ. But this means that the run τ terminated with the result *inconsistent* which is a contradiction.

- concrete domain satisfiability. Assume that there exists a set $\mathcal{C} \subset \mathcal{A}_\cup$ of axioms of the form $(X_1, \ldots, X_n) : P$, where each X_i denotes a concrete object x_k used in $\mathcal{C}$, such that the corresponding conjunction of concrete domain predicates is unsatisfiable. Axioms of the above form are introduced during rule application by feature-complete, only. From this follows that in any axiom $(X_1, \ldots, X_n) : P$ from $\mathcal{C}$, all objects

$X_1, \ldots, X_n$ have the same index, i.e., if $X_i = x_k$ and $X_j = y_l$, then $k = l$. This means that $\mathcal{C}$ can be divided into subsets $\mathcal{C}_1, \ldots, \mathcal{C}_k$, such that the sets $\mathcal{C}_i$ are mutually disjoint, we have $\mathcal{C} = \mathcal{C}_1 \cup \cdots \cup \mathcal{C}_k$, and, furthermore, all concrete objects appearing in a set $\mathcal{C}_i$ have the same index. From a solution of the predicate conjunctions corresponding to the sets $\mathcal{C}_1, \ldots, \mathcal{C}_k$, a solution for the predicate conjunction corresponding to $\mathcal{C}$ can easily be constructed. Fix a set $\mathcal{C}_i$. Let j be the index of the concrete objects appearing in $\mathcal{C}_i$. Let $\mathcal{X}$ be the set of axioms that is checked for concrete domain satisfiability in the j'th recursion step of τ. By definition of $\mathcal{A}_\cup$, we have $\mathcal{C}_i \subseteq \mathcal{X}$. From the fact that the run τ returned *consistent*, it follows that $\mathcal{X}$ and hence also $\mathcal{C}_i$ is concrete domain satisfiable. Since this argument holds for all $i = 1 \ldots k$, a contradiction to the assumption is obtained.

■

We now return to the proof of the third claim of Lemma 11. Based on $\mathcal{A}_\cup$, an interpretation $\mathcal{I}$ can be defined as follows:

1. $\Delta_\mathcal{I}$ consists of all objects in O_A that occur in $\mathcal{A}_\cup$
2. If A is a concept name then $a \in A^\mathcal{I}$ iff $a\!:\!A \in \mathcal{A}_\cup$.
3. If R is a role or feature name then $(a, b) \in R^\mathcal{I}$ iff $(a, b)\!:\!R \in \mathcal{A}_\cup$.
4. Because there $\mathcal{A}_\cup$ is concrete domain satisfiable (Lemma 12 Point 3), there is a variable assignment α that satisfies the conjunction of all occurring axioms $(x_1, \ldots, x_n)\!:\!P$. So we set $x^\mathcal{I} = \alpha(x)$ for all $x \in \mathsf{O}_D$.

It remains to be proven that $\mathcal{I}$ is a model for $\mathcal{A}_\cup$. The proof is by induction over the size of axioms of the form $a\!:\!C$ in $\mathcal{A}_\cup$ and makes a case distinctions according to the topmost operator in C. A prerequisite for the proof is that the ABox $\mathcal{A}_\cup$, which was used to construct the interpretation $\mathcal{I}$, is complete, clash-free and concrete domain satisfiable, which is assured by Lemma 12. Most cases are already treated in [6], so we only deal with the two remaining cases, which belong to the induction start.

- Let α be $a : u_1{\downarrow}u_1$. Since the R$\downarrow$ rule is not applicable, there is an object $b \in \mathsf{O}_A$ for which we have both $u_1^\mathcal{I}(a^\mathcal{I}) = b^\mathcal{I}$ and $u_2^\mathcal{I}(a^\mathcal{I}) = b^\mathcal{I}$. Hence, $\mathcal{I}$ satisfies α.

- Let α be $a : u_1 \uparrow u_1$. Since the R$\uparrow$ rule is not applicable, there are objects $b_1, b_2 \in \mathsf{O}_A$ for which we have $u_1^{\mathcal{I}}(a^{\mathcal{I}}) = b_1^{\mathcal{I}}$ and $u_2^{\mathcal{I}}(a^{\mathcal{I}}) = b_2^{\mathcal{I}}$. Since the agreement clash is not applicable, $b_1^{\mathcal{I}}$ is distinct from $b_2^{\mathcal{I}}$, and, hence, $\mathcal{I}$ satisfies α.

(4) To finish the proof of Lemma 11, the termination of sat needs to be proven. Termination is a consequence of the following claims:

(i) feature-complete terminates on any input.

(ii) The recursion depth of sat is bounded.

(iii) In each recursion step, only a finite number of recursive calls are made.

Claim (iii) is obvious. Both (i) and (ii) will be proved by a lemma which establish an even stronger result. The stronger result will be helpful in proving the polynomial space-complexity of sat. The following Lemma settles claim (i).

Lemma 13. *For any input ABox $\mathcal{A}$, the function* feature-complete *terminates and constructs an ABox $\mathcal{A}'$ for which we have $||\mathcal{A}'|| \leq ||\mathcal{A}||^2 + ||\mathcal{A}||$.*

Proof: The upper bound for the size of $\mathcal{A}'$ given in the Lemma is a consequence of the following two points:

1. feature-complete generates no more than $||\mathcal{A}||$ new axioms.
2. For each axiom α, we have $||\alpha|| \leq ||\mathcal{A}||$.

Please note that axioms are never deleted which is the reason for the second summand. The second point is obvious, but the first one needs to be proven. Termination of feature-complete directly follows from this first point as well. The rules Rr$\exists$C and Rr$\forall$C will not be considered since they are not applied by feature-complete. For all other completion rules, the most important observation is that they can be applied at most once per axiom $a : C$. This is also true for axioms $a : \forall f.C$ and the Rf$\forall$C rule since there is at most one successor per feature and object. Because of this, we make the simplifying assumption that the premise of the Rf$\forall$C rule does only contain the axiom $a : \forall f.C$ but no axiom $(a, b) : f$, i.e., that it is applied to *every* axiom of the first form regardless if there is an axiom of the second form or not. This may result in too high an estimation of the number of generated axioms but not

in one that is too low. Since in our simplified view, every completion rule is applied to exactly one axiom, we may proof the first point from above by showing that, for each axiom α in $\mathcal{A}$, no more than $||\alpha||$ axioms are generated by feature-complete. This will be done in the following.

No new axioms are generated for axioms of the form $(a, b) : R$, $(a, x) : f$, $a \neq b$, and $(x_1, \ldots, x_n) : P$ since they do not appear in the premise of any completion rule (please recall the simplification we made about Rf∀C). The remaining axioms are of the form $a : C$. For these axioms, the property in question can be proven by induction on the structure of C.

For the induction start, let C be $\exists u_1, \ldots, u_n.P$, $u_1 \downarrow u_2$, $u_1 \uparrow u_2$, $\exists \hat{R}.C$, $\forall \hat{R}.C$, or an atomic concept. In any of these cases, it is trivial to verify that at most $||C||$ new axioms may be generated. For the induction step, we need to make a case distinction according to the form of C. Let C be of the form $D \sqcap E$. The application of the R$\sqcap$ rule generates two axioms $a : D$ and $a : E$. By induction hypothesis, from these two axioms, at most $||D||$ and $||E||$ axioms may be generated, respectively. Hence, from $a : D \sqcap E$, at most $||D|| + ||E|| + 2 = ||D \sqcap E||$ new axioms may be generated. The remaining cases $D \sqcup E$, $\exists f.C$, and $\forall f.C$ are analogous. Because of the simplifying assumptions made, universal quantification over features does not need a special treatment. ∎

Claim (ii) of the above list follows from the next Lemma which concludes the termination proof for sat.

Lemma 14. *For any input $\mathcal{A}_0$, the recursion depth of* sat *is bounded by $||\mathcal{A}_0||$.*

Proof: The *role depth* of a concept C is the maximum nesting depth of exists and value restrictions in C.[6] The role depth of an ABox $\mathcal{A}$ is the maximum role depth of all concepts occurring in $\mathcal{A}$. As an immediate consequence of the way in which the input ABoxes of recursive calls are constructed, we have that the role depth of the arguments ABoxes strictly decreases with recursion depth (note that rule application performed by feature-complete cannot increase the role depth). It follows that the role depth of the initial ABox $\mathcal{A}_0$ is an upper bound for the recursion depth, and, by definition of $||\cdot||$, this number is clearly smaller than $||\mathcal{A}_0||$. ∎

[6] E.g., the concept $\exists R.C \sqcap \forall S.(D \sqcap \exists R.C)$ has role depth 2.

The proof of Lemma 11 is now complete. The Lemma plays a central rôle in the proof of the following proposition.

Proposition 15. *The* sat *algorithm is sound, complete and terminates on any input.*

Proof: By definition of nondeterministic algorithms, the application of the sat algorithm to an ABox $\mathcal{A}_0$ gives the result *consistent* iff there is a run of sat on $\mathcal{A}_0$ which returns *consistent*. As shown in parts 2 and 3 of Lemma 11, a sat run defines a model for the corresponding ABox $\mathcal{A}_\cup$ if and only if the run returns *consistent*. First assume that there is a run of sat which returns *consistent*. By part 1 of Lemma 11, there also exists a model for $\mathcal{A}_0$. Now assume that all runs of sat return *inconsistent*. In this case, it follows from part 1 of Lemma 11 that $\mathcal{A}_0$ cannot have a model since there is a run for every combination of nondeterministic choices that can be made during rule application. ∎

Having proved its correctness, the complexity of the sat algorithm can now be analyzed. Since the most important results have already been established, this can be done straightforwardly.

Proposition 16. *For any input* $\mathcal{A}_0$, sat *can be executed in space polynomial in* $||\mathcal{A}_0||$, *provided that this also holds for the function* satisfiable?.

Proof: Sat is a recursive function. Let us first analyze the maximum size of arguments that are passed to sat in recursion calls. The argument to sat is an ABox which contains axioms of the form $a : C$ for a single object a. Since no new concepts are generated during rule application, there can be at most as many axioms of this form per single object as there are distinct concepts and subconcepts appearing in $\mathcal{A}_0$. Considering the definition of $||\cdot||$, it is easy to see that this number can be at most $||\mathcal{A}_0||$. Furthermore, the size of any axiom is at most $||\mathcal{A}_0||$. It follows that the maximum size of arguments given in a recursion call is $||\mathcal{A}_0||^2$. Using feature-complete, the argument ABox is extended by new axioms. Combining the argument size just obtained with Lemma 13, we find that the maximum size of ABoxes constructed during recursion calls is $||\mathcal{A}_0||^4 + ||\mathcal{A}_0||^2$. Lemma 14 gives us that the recursion depth is bounded by $||\mathcal{A}_0||$, and, hence, sat can be executed in $||\mathcal{A}_0||^5 + ||\mathcal{A}_0||^3$ space. ∎

4.2 The ABox-cons Algorithm

In this section, the correctness of the ABox-cons algorithm will be established and its complexity analyzed. ABox-cons reduces ABox consistency to concept satisfiability as follows. It performs preprocessing on $\mathcal{A}$, then constructs a set of "reduction concepts" and finally calls sat once for each concept in this set.

We start with proving the correctness. First, a basic notion is introduced. An ABox $\mathcal{A}$ that was completed by rule application through the preprocess function is called *preprocessing complete*. If $\mathcal{A}_p$ is preprocessing complete and $\mathcal{A} \subseteq \mathcal{A}_p$, then $\mathcal{A}_p$ is a *preprocessing completion* of $\mathcal{A}$.

Proposition 17. *The* ABox-cons *Algorithm is sound, complete, and terminates on any input.*

Proof: Since preprocess applies the nondeterministic completion rule R$\sqcup$, there may be more than one preprocessing completion for a given ABox $\mathcal{A}$. By definition of nondeterministic algorithms, the ABox-cons algorithm gives the result *consistent* for an ABox $\mathcal{A}_0$ iff one of the possible runs of ABox-cons on $\mathcal{A}_0$ returns *consistent*. Hence, to prove the soundness and completeness of ABox-cons, it has to be shown that (i) an ABox $\mathcal{A}$ has a model iff one of its preprocessing completions has a model and (ii) the reduction to concept satisfiability is sound and complete. This will be done in the following.

The next Lemma establishes point (i). It is an immediate consequence of the local correctness of the completion rules (Lemma 11, part 1).

Lemma 18. *Let $\mathcal{A}$ be an ABox. $\mathcal{A}$ has a model iff one the preprocessing completions $\mathcal{A}_p$ of $\mathcal{A}$ has a model.*

Next, point (ii) is proved. In the "for" loop of ABox-cons, a concept C (*reduction concepts* of $\mathcal{A}_p$) is generated for each axiom of the form $a : \exists \hat{R}.D$ in $\mathcal{A}_p$. These concepts are then one by one passed to the sat function. It needs to be shown that a preprocessing complete ABox has a model if and only if all of the reduction concepts have a model.

Proposition 19 (Soundness). *Let $\mathcal{I}$ be a model for a preprocessing complete ABox $\mathcal{A}_p$. Then, all reduction concepts $C_1, \ldots, C_n$ of $\mathcal{A}_p$ do also have a model.*

Proof: C_i has the form $C \sqcap D_1 \sqcap \ldots \sqcap D_k$ for $1 \leq i \leq n$. This means that in $\mathcal{A}_p$, there are axioms $a : \exists \hat{R}.C$, $a : \forall \hat{R}.D_1$, $\ldots$, $a : \forall \hat{R}.D_k$. By the semantics of

the exists and value restriction, there exists a domain object $x \in \Delta_{\mathcal{I}}$, such that $x \in C^{\mathcal{I}}$ and $x \in D_1^{\mathcal{I}}, \ldots, x \in D_k^{\mathcal{I}}$. $\mathcal{I}$ is obviously an interpretation for C_i and we have $x \in C_i^{\mathcal{I}}$. Hence, $\mathcal{I}$ is a model for C_i. ∎

Let C be a reduction concept. C is created because of the existence of a concept $a : \exists \hat{R}.D$ in $\mathcal{A}_p$ and then passed to sat as part of an ABox $\{b : C\}$. The object b is called the *root object* of C and denoted by $robj(C)$. The object a is denoted by $cobj(C)$ and the role $\hat{R}$ by $role(C)$.

Proposition 20 (Completeness). *Let $\mathcal{A}_p$ be an ABox which is preprocessing complete and concrete domain satisfiable and does not contain a clash. If there are models $\mathcal{I}_1, \ldots, \mathcal{I}_n$ for all reduction concepts $C_1, \ldots, C_n$ of $\mathcal{A}_p$, then $\mathcal{A}_p$ does also have a model.*

Proof: The proposition will be proved by showing that from the preprocessing complete ABox $\mathcal{A}_p$ together with the models $\mathcal{I}_1, \ldots, \mathcal{I}_n$ for the reduction concepts $C_1, \ldots, C_n$, a model $\mathcal{I}$ for $\mathcal{A}_p$ can be constructed. Without loss of generality, it is assumed that $\Delta_{\mathcal{I}_1} \cap \cdots \cap \Delta_{\mathcal{I}_n} \cap \{a \in \mathsf{O}_A \mid a \text{ used in } \mathcal{A}_p\} = \emptyset$. If this is not the case, it can be achieved by consistently renaming the domain objects used in $\mathcal{I}_1, \ldots, \mathcal{I}_n$. The interpretation $\mathcal{I}$ is defined as follows.

- $\Delta_{\mathcal{I}}$ is set to $\Delta_{\mathcal{I}_1} \cup \cdots \cup \Delta_{\mathcal{I}_n} \cup \{a \in \mathsf{O}_A \mid a \text{ used in } \mathcal{A}_p\}$.
- For atomic concepts A, $A^{\mathcal{I}} := \bigcup_{i=1,\ldots,n} A^{\mathcal{I}_i} \cup \bigcup_{a:A \in \mathcal{A}_p} a$.
- For roles $\hat{R}$, $\hat{R}^{\mathcal{I}} := \bigcup_{i=1,\ldots,n \mid \hat{R} = role(C_i)} \{(cobj(C_i), robj(C_i)^{\mathcal{I}_i})\} \cup \bigcup_{i=1,\ldots,n} \hat{R}^{\mathcal{I}_i} \cup \bigcup_{(o_1,o_2):\hat{R} \in \mathcal{A}_p} \{(o_1, o_2)\}$.
- For features f, $f^{\mathcal{I}}(a) := x$ iff $\exists i \in \{1, \ldots, n\} : f^{\mathcal{I}_i}(a) := x$ or $(o, x) : f \in \mathcal{A}_p$.
- For object names $a \in \mathsf{O}_A$ used in $\mathcal{A}_p$, $a^{\mathcal{I}} := a$.
- Since $\mathcal{A}_p$ is concrete domain satisfiable, there is a mapping δ from O_D $\Delta_{\mathcal{D}}$ such that $\bigwedge_{(x_1,\ldots,x_n):P \in \mathcal{A}_p} (\delta(x_1), \ldots, \delta(x_n)) \in P^{\mathcal{D}}$ is true in $\mathcal{D}$. $x^{\mathcal{I}}$ is set to $\delta(x)$.

It remains to prove that $\mathcal{I}$ is a model of $\mathcal{A}_p$. First consider axioms from $\mathcal{A}_p$ which are of the form $a : \exists \hat{R}.C$ (where $\hat{R}$ is a role): By definition of ABox-cons, there is a reduction concept C_i with root object b, such that $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in \hat{R}^{\mathcal{I}}$. Since $\mathcal{I}_i$ is a model for $b : C_i$, by construction of $\mathcal{I}$ we have $b^{\mathcal{I}} \in C^{\mathcal{I}}$.

Now consider axioms of the form $a : \forall \hat{R}.C$ (with $\hat{R}$ role). If there is also an axiom $a : \exists \hat{R}.D$ in $\mathcal{A}_p$, then there is a reduction concept C_i with root object b, such that $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in \hat{R}^{\mathcal{I}}$. By construction of the reduction concepts and of $\mathcal{I}$, we have $b^{\mathcal{I}} \in D^{\mathcal{I}}$. For each axiom $(a, b) : \hat{R}$ in $\mathcal{A}_p$, Rr$\forall$C has been applied. Since there is no primitive clash in $\mathcal{A}_p$, b is in $C^{\mathcal{I}}$.

To show that the remaining axioms in $\mathcal{A}_p$ are also satisfied by $\mathcal{I}$, an inductive proof over the size of the axioms, which is identical to the one employed in Lemma 11, part 3, can be used. ∎

To complete the proof of Proposition 17, termination of ABox-cons has to be proven. This is part of the following Proposition which also treats the space requirements of ABox-cons.

Proposition 21. *Started on an ABox $\mathcal{A}$,* ABox-cons *terminates and needs space polynomial in $||\mathcal{A}||$, provided that the the function* satisfiable? *can also be executed in polynomial space.*

Proof: It was already proven that the sat algorithm terminates and can be executed in polynomial space if the function satisfiable? can be executed in polynomial space (Propositions 15 and 16). Thus, it remains to show that the preprocess function terminates and that for any ABox $\mathcal{A}$, the size $||\mathcal{A}'||$ of $\mathcal{A}' :=$ preprocess$(\mathcal{A})$ is polynomial in $||\mathcal{A}||$. For this proof, it is crucial that the preprocess function is identical to the feature-complete function except that preprocess does also apply the Rr$\forall$C rule. It will be shown that the presence of this additional rule yields only "slightly" more axioms. In the following, objects are called *old* if they are used in $\mathcal{A}$ and *new* if they are used in $\mathcal{A}'$ but not in $\mathcal{A}$.

Consider applications of the Rr$\forall$C rule performed by preprocess. It is an immediate consequence of the following two observations that the number of Rr$\forall$C applications is bounded by $||\mathcal{A}||^2$:

- If Rr$\forall$C is applied to axioms $a : \forall \hat{R}.C$ and $(a, b) : \hat{R}$, then b is an old object. This is the case since preprocess does not apply Rr$\exists$C, and, hence, no new axioms of the form $(a, b) : \hat{R}$, where $\hat{R}$ is a role, are generated. Furthermore, there are at most $||\mathcal{A}||$ old objects.

- Rr$\forall$C creates only axioms of the form $b : C$, which are not already part of the ABox. For any object b, there may exist at most one axiom $b : C$ per concept C used in $\mathcal{A}$ (including all subconcepts). The number of (sub)concepts in $\mathcal{A}$ is bounded by $||\mathcal{A}||$, and, hence, there can be at most $||\mathcal{A}||$ axioms of the form $b : C$ per object b.

The size of each axiom $a : C$ is again bounded by $||\mathcal{A}||$. It follows that the total size of axioms added by Rr∀C applications performed by preprocess is bounded by $||\mathcal{A}||^3$.

Second, consider the application of all other rules. These are exactly the rules applied by feature-complete. Together with the upper bound for the number of Rr∀C applications, this immediately gives us termination. If feature-complete is applied to an ABox $\mathcal{A}$, then the size of the resulting ABox is bounded by $||\mathcal{A}||^2 + ||\mathcal{A}||$ (Lemma 13). To establish an upper bound for the size of the ABox $\mathcal{A}'$ obtained from a call topreprocess, the additional axioms generated by Rr∀C need to be taken into account. It was just proved that the size of these axioms is at most $||\mathcal{A}||^3$. Thus, an upper bound for $||\mathcal{A}'||$ is given by the application of feature-complete to an ABox of size $||\mathcal{A}|| + ||\mathcal{A}||^3$. Summing up, $||\mathcal{A}'||$ is bounded by $||\mathcal{A}||^6 + ||\mathcal{A}||^3 + ||\mathcal{A}||^2 + ||\mathcal{A}||$. ∎

4.3 PSPACE-Completeness

Using the results from the last two sections, proving PSPACE-completeness of $\mathcal{ALCF}(\mathcal{D})$ ABox consistency is straightforward.

Theorem 22. *Provided that the satisfiability test of the concrete domain $\mathcal{D}$ is in* PSPACE*, the following problems are* PSPACE*-complete:*

1. *Consistency of $\mathcal{ALCF}(\mathcal{D})$ ABoxes.*
2. *Consistency of $\mathcal{ALC}(\mathcal{D})$ and $\mathcal{ALCF}$ ABoxes.*
3. *Satisfiability and subsumption of $\mathcal{ALCF}(\mathcal{D})$ concepts.*
4. *Satisfiability and subsumption of $\mathcal{ALC}(\mathcal{D})$ concepts.*

Proof: First, point 1 is proven. In [31] it is proved that satisfiability for $\mathcal{ALC}$ concepts is PSPACE-complete. Since $\mathcal{ALC}$ is a proper subset of $\mathcal{ALCF}(\mathcal{D})$, deciding the consistency of $\mathcal{ALCF}(\mathcal{D})$ ABoxes is PSPACE-hard. It remains to be shown that it is also in PSPACE if the concrete domain satisfiability test is in PSPACE. But this directly follows from Proposition 21 together with the well-known fact that PSPACE = NPSPACE [29].

Point 3 is a consequence of the observation that satisfiability as well as subsumption can be reduced to ABox consistency (see Section 2).

Points 1 and 2 hold since reasoning with $\mathcal{ALC}$ is PSPACE-hard and $\mathcal{ALC}$ is a proper subset of both $\mathcal{ALC}(\mathcal{D})$ and $\mathcal{ALCF}$ which are in turn a proper subsets of $\mathcal{ALCF}(\mathcal{D})$. ∎

The next corollary makes a statement about the complexity of reasoning with concrete domains for which the satisfiability test is not in PSPACE.

Corollary 23. *Let the satisfiability test of the concrete domain $\mathcal{D}$ be in a complexity class X with* PSPACE $\subseteq X$. *Then, subsumption, satisfiability and ABox consistency for the logics $\mathcal{ALC}(\mathcal{D})$ and $\mathcal{ALCF}(\mathcal{D})$ are in X.*

The next section gives some examples of interesting concrete domains, for which the satisfiability check is in PSPACE.

5 Suitable Concrete Domains

In this section, some examples of concrete domains, for which the satisfiability test is in PSPACE, are given. This demonstrates that the results developed in the preceding sections are useful since there are in fact interesting concrete domains which allow to exploit the lower complexity bound found.

As proposed in [24], $\mathcal{ALC}(\mathcal{D})$ (and hence also $\mathcal{ALCF}(\mathcal{D})$) can be used for temporal reasoning when instantiated with an appropriate concrete domain. Since different temporal ontologies can be used as a basis for temporal reasoning, the definition of various temporal concrete domains is possible. Allen's interval algebra (see [3], [23]) provides an appropriate formalization if intervals are used as the basic temporal entity. This formalism employs a set of 13 base relations describing all possible relationships between two (time) intervals. These relations are known as Allen's relations. In the following, a set of concrete domains supporting reasoning with Allen's relations is defined.

Definition 24. Let $(\mathcal{P}, <)$ be a linear, unbounded, and dense temporal structure, where $\mathcal{P}$ is a set of time points and $<$ is a strict linear order on $\mathcal{P}$. The concrete domain $\mathcal{I}_{(\mathcal{P},<)}$ is defined as follows. As the domain $\Delta_{\mathcal{I}_{(\mathcal{P},<)}}$, the set of intervals $\{(l, r) \in \mathcal{P} \times \mathcal{P} \mid l < r\}$ is used. For an interval $i \in \mathcal{I}_{(\mathcal{P},<)}$ with $i = (l, r)$, we define $l(i) = l$ and $r(i) = r$. The set $\Phi_{\mathcal{I}_{(\mathcal{P},<)}}$ contains the following predicates (we will in the following omit the index $(\mathcal{P}, <)$ for brevity):

- a unary predicate *is-interval* which yields true for every element in $\Delta_{\mathcal{I}}$; its negation *is-no-interval*; a predicate *inconsistent-relation* of arity 2 which is always false.

- for each of Allen's 13 relations, a predicate of arity 2 (*basic* predicates). The predicates are defined by specifying relationships between interval endpoints as usual:

$$
\begin{aligned}
(i_1, i_2) \in \mathit{equal}^{\mathcal{I}} &\text{ iff } l(i_1) - l(i_2) = 0 \wedge r(i_1) - r(i_2) = 0 \\
(i_1, i_2) \in \mathit{before}^{\mathcal{I}} &\text{ iff } l(i_2) - r(i_1) > 0 \\
(i_1, i_2) \in \mathit{after}^{\mathcal{I}} &\text{ iff } l(i_1) - r(i_2) > 0 \\
(i_1, i_2) \in \mathit{meets}^{\mathcal{I}} &\text{ iff } l(i_2) - r(i_1) = 0 \\
(i_1, i_2) \in \mathit{overlaps}^{\mathcal{I}} &\text{ iff } l(i_2) - l(i_1) > 0 \wedge r(i_1) - l(i_2) > 0 \wedge r(i_2) - r(i_1) > 0 \\
(i_1, i_2) \in \mathit{overlapped\text{-}by}^{\mathcal{I}} &\text{ iff } l(i_1) - l(i_2) > 0 \wedge r(i_2) - l(i_1) > 0 \wedge r(i_1) - r(i_2) > 0 \\
(i_1, i_2) \in \mathit{during}^{\mathcal{I}} &\text{ iff } l(i_1) - l(i_2) > 0 \wedge r(i_2) - l(i_1) > 0 \wedge \\
&\qquad r(i_1) - l(i_2) > 0 \wedge r(i_2) - r(i_1) > 0 \\
(i_1, i_2) \in \mathit{contains}^{\mathcal{I}} &\text{ iff } l(i_2) - l(i_1) > 0 \wedge r(i_1) - l(i_2) > 0 \wedge \\
&\qquad r(i_2) - l(i_1) > 0 \wedge r(i_1) - r(i_2) > 0 \\
(i_1, i_2) \in \mathit{starts}^{\mathcal{I}} &\text{ iff } l(i_2) - l(i_1) = 0 \wedge r(i_2) - r(i_1) > 0 \\
(i_1, i_2) \in \mathit{started\text{-}by}^{\mathcal{I}} &\text{ iff } l(i_2) - l(i_1) = 0 \wedge r(i_1) - r(i_2) > 0 \\
(i_1, i_2) \in \mathit{finishes}^{\mathcal{I}} &\text{ iff } r(i_2) - r(i_1) = 0 \wedge l(i_1) - l(i_2) > 0 \\
(i_1, i_2) \in \mathit{finished\text{-}by}^{\mathcal{I}} &\text{ iff } r(i_2) - r(i_1) = 0 \wedge l(i_2) - l(i_1) > 0
\end{aligned}
$$

- for each distinct set $\{R_1, \ldots, R_n\}$ of Allen's relations, an additional predicate of arity 2 is defined (*combined* predicates). The predicate has the name $R_1\text{-}\cdots\text{-}R_n$ and we have $(i_1, i_2) \in R_1\text{-}\cdots\text{-}R_n{}^{\mathcal{I}}$ iff $(i_1, i_2) \in R_1{}^{\mathcal{I}}$ or ... or $(i_1, i_2) \in R_n{}^{\mathcal{I}}$. In total, there are $2^{13} - 14$ of these combined relations.

As an example of combined predicates, please consider the predicate *after-before*. A pair of intervals (i_1, i_2) is in *after-before*$^{\mathcal{I}}$ iff $(i_1, i_2) \in \mathit{after}^{\mathcal{I}}$ or $(i_1, i_2) \in \mathit{before}^{\mathcal{I}}$. As an example of the modeling of temporal concepts using

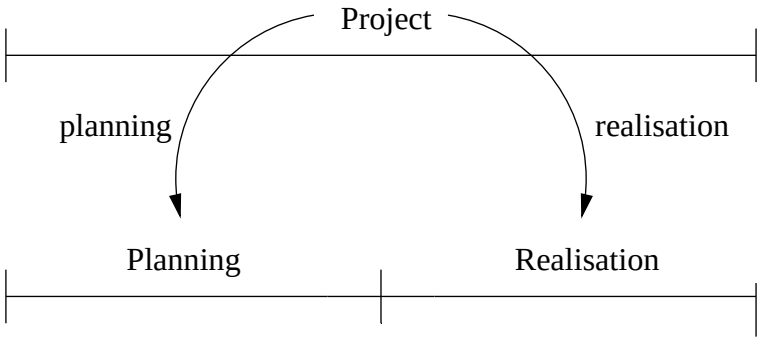

Figure 2: Visualization of the *Project* concept

$\mathcal{ALCF}(\mathcal{I})$, consider the following concepts:

$$\begin{aligned} \mathit{Project} \doteq\ & \mathit{Interval} \sqcap \\ & \exists \mathit{planning}.\mathit{Planning} \sqcap \exists \mathit{realization}.\mathit{Realization} \sqcap \\ & \exists(\mathit{planning} \circ \mathit{time}), (\mathit{time}).\mathit{starts} \sqcap \\ & \exists(\mathit{realization} \circ \mathit{time}), (\mathit{time}).\mathit{ends} \sqcap \\ & \exists(\mathit{planning} \circ \mathit{time}), (\mathit{realization} \circ \mathit{time}).\mathit{meets} \end{aligned}$$

The *Project* concept gives the formalization of a project on a very coarse level. The project can be devided into the two disjunctive phases "planning" and "realization". Please see Figure 2 for a visualization of this concept.

To use $\mathcal{I}$ as a concrete domain for $\mathcal{ALCF}(\mathcal{D})$, the admissibility of $\mathcal{I}$ has to be shown.

Proposition 25. *The concrete domain $\mathcal{I}$ is admissible.*

Proof: It has to be shown that (i) $\Phi_{\mathcal{I}}$ is closed under negation and (ii) the satisfiability of finite conjunctions of predicates from $\mathcal{I}$ is decidable.

(i) Allen's relations are mutually exclusive and exhaustive, i.e. exactly one of Allen's relations holds between any two intervals i_1 and i_2. Given this, it is easy to verify that $\Phi_{\mathcal{I}}$ is closed under negation: Let A be the set of Allen's 13 base relations. For any predicate $R_1\text{-}\cdots\text{-}R_n$ with $1 \leq n < 13$, we have that the negation of $R_1\text{-}\cdots\text{-}R_n$ is $S_1\text{-}\cdots\text{-}S_k$, where $\{S_1, \ldots, S_k\}$ is defined as $A \setminus \{R_1, \ldots, R_n\}$. The predicate $S_1\text{-}\cdots\text{-}S_k$ is obviously in $\Phi_{\mathcal{I}}$. Note that the negation of the predicate representing the disjunctive combination of all 13 base relations is *inconsistent-relation* and vice versa.

(ii) Deciding the satisfiability of a finite conjunction of predicates from $\mathcal{I}$ can be reduced to checking the satisfiability of a temporal constraint network. In the following, a decision procedure will be described and its complexity determined. A *temporal constraint network (TCN)* is defined as a set $\{tc_1(i_1^{(1)}, i_2^{(1)}), \ldots, tc_n(i_1^{(n)}, i_2^{(n)})\}$, where each tc_i is a disjunction $R_1^{(i)} \vee \cdots \vee R_{n_i}^{(i)}$ of Allen's relations and the $i_j^{(i)}$ are *interval variables*. Let a finite conjunction $C = p_1(x_1^{(1)}, \ldots, x_{n_1}^{(1)}) \wedge \cdots \wedge p_k(x_1^{(k)}, \ldots, x_{n_k}^{(k)})$ of predicates from $\Phi_{\mathcal{I}}$ be given. Its satisfiability can be determined as follows:

- If, for any $i = 1, \ldots, k$, p_i is either *is-no-interval* or *inconsistent-relation*, then return *unsatisfiable*.
- For $i \in \{1, \ldots, k\}$, if $p_i =$ *is-interval*, remove the i'th conjunct.
- Translate the remaining conjunction C' (which contains only binary predicates) into a TCN N as follows: The set V of interval variables used in N is the set of variables used in C. For each conjunct $p_i(x_1, x_2)$ from C', do the following. The predicate p_i has the form $R_1\text{-}\cdots\text{-}R_n$.
  1. If there is no constraint $tc(x_1, x_2)$ in N, then add $R_1 \vee \cdots \vee R_n(x_1, x_2)$ to N.
  2. If there is already a constraint $tc(x_1, x_2)$ in N where $tc = R'_1 \vee \cdots \vee R'_m$, then let S be defined as $\{R'_1, \ldots, R'_m\} \cap \{R_1, \ldots, R_n\}$. If $S = \emptyset$, then return *inconsistent*. Otherwise, remove the existing constraint and add $S_1 \vee \cdots \vee S_l(x_1, x_2)$, where $\{S_1, \ldots, S_l\} = S$.
- For all pairs $(x_1, x_2) \in V$, for which there is no constraint $tc(x_1, x_2)$ in N, add a constraint $cov(x_1, x_2)$, where cov is the disjunction of all 13 base relations.
- Check the satisfiability of the TCN N and return the result.

Testing the satisfiability of temporal constraint networks of the given form is an NP-complete problem if the full set of relations is allowed [33]. The correctness of the procedure is easily seen. ∎

Hence, $\mathcal{I}$ is an admissible concrete domain and the standard reasoning problems for $\mathcal{ALCF}(\mathcal{I})$ are PSPACE-complete.

Some comments about the temporal logic $\mathcal{ALCF}(\mathcal{I})$ are in order. We give a brief comparison to a temporal description logic defined by Artale and Franconi. In $\mathcal{ALCF}(\mathcal{I})$, the structure of time has to be modeled in the logic itself. This is called an internal representation of time as opposed to an external representation, where the structure of time is an integral part of the semantics [14]. The latter approach is taken in temporal logic and is also pursued by the description logic $\mathcal{TL}$-$\mathcal{ALCF}$ [4]. $\mathcal{TL}$-$\mathcal{ALCF}$ is a temporal description logic for reasoning about eternal objects that have properties changing over time. Quite to the contrary, $\mathcal{ALCF}(\mathcal{I})$ most adequately supports reasoning about objects which have static properties but a unique temporal extension. Please note that neither $\mathcal{ALCF}(\mathcal{I})$ nor $\mathcal{TL}$-$\mathcal{ALCF}$ allows universal quantification about Allen's relations. See [16] for a description logic which is capable of doing this. Artale and Franconi [5] give an extensive overview over the various approaches to the definition of temporal description logics.

In the remainder of this section, we give a brief discussion of some more concrete domains. An important approach to qualitative spatial reasoning uses a set of 8 topological relations called RCC-8. These relations can be used to describe the relationships of two arbitrary regions in n-dimensional space (see [28] and [13]). A concrete domain $\mathcal{S}_2$ for reasoning with the RCC-8 relations about regions in the plane can be defined exactly analogous to the definition of $\mathcal{I}$. A formal definition can be found in [16]. The satisfiability test for this spatial concrete domain is also in NP.

A further source for concrete domains for which deciding satisfiability is in PSPACE are the areas of linear programming and constraint programming. Linear programming itself is a polynomial problem [32]. This means that linear equalities and inequalities over the rational numbers may be used as predicates of a concrete domain and that the complexity of reasoning with the combined language remains in PSPACE. If the integers are used instead of the rationals, techniques from integer programming can be employed for the concrete domain satisfiability test. In this case, the (concrete domain) satisfiability test is in NP [15, problem MP1], hence reasoning with the combined formalism is in PSPACE.

A concrete domain for which satisfiability is *not* in PSPACE is e.g. $\mathcal{R}$ as defined in [6]. The domain of $\mathcal{R}$ is the set of all real numbers while the predicates of this concrete domain are given by formulae built by first order means from equalities and inequalities between integer polynomials. It was

proved that deciding the satisfiability of such expressions is an ExpSpace-complete problem [25].

6 Conclusion

In this paper, the description logic $\mathcal{ALCF}(\mathcal{D})$ was introduced. $\mathcal{ALCF}(\mathcal{D})$ extends $\mathcal{ALC}(\mathcal{D})$ by feature agreement and disagreement and is hence a conglomeration of the logics $\mathcal{ALC}(\mathcal{D})$ and $\mathcal{ALCF}$. Algorithms for deciding the satisfiability of $\mathcal{ALCF}(\mathcal{D})$ concepts and for deciding the consistency of $\mathcal{ALCF}(\mathcal{D})$ ABoxes were devised and it was shown that these algorithm are sound and complete and can be executed in polynomial space if this is also the case for the concrete domain satisfiability test. From this result, it was derived that the standard reasoning problems concerning concept subsumption, concept satisfiability and ABox consistency are PSpace-complete for the description logics $\mathcal{ALC}(\mathcal{D})$ and $\mathcal{ALCF}(\mathcal{D})$ - provided that the concrete domain satisfiability test is in PSpace. If the satisfiability test is in a higher complexity class X, then the mentioned problems are also in X. As an important contribution, the completion algorithm demonstrates how to split the concrete domain satisfiability check into chunks of polynomial size. This is a vital prerequisite for efficient implementations of languages incorporation concrete domains.

Acknowledgments

I would like to thank Ulrike Sattler for stoicism and countless discussions and Franz Baader for enlightening discussions and helpful comments. The work in this paper was supported by the "Foundations of Data Warehouse Quality" (DWQ) European ESPRIT IV Long Term Research (LTR) Project 22469.

References

[1] *Proceedings of the Ninth National Conference on Artificial Intelligence AAAI-91*, Anaheim, California, July 14 - 19, 1991. AAAI-Press/The MIT Press, Menlo Park – Cambridge – London, 1991.

[2] A. Abecker, D. Drollinger, and P. Hanschke. Taxon: A concept language with concrete domains. In *PDK '91: Proc. of the International Workshop on Processing Declarative Knowledge*, Kaiserslautern, 1991.

[3] J. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11), 1983.

[4] A. Artale and E. Franconi. A temporal description logic for reasoning about actions and plans. *Journal of Artificial Intelligence Research (JAIR)*, (9), 1998.

[5] A. Artale and E. Franconi. Temporal description logics. In L. Vila, P. van Beek, M. Boddy, M. Fisher, D. M. Gabbay, A. Galton, and R. Morris, editors, *Handbook of Time and Temporal Reasoning in Artificial Intelligence*. MIT Press, To appear.

[6] F. Baader and P. Hanschke. A scheme for integrating concrete domains into concept languages. In Mylopoulos and Reiter [27], pages 452–457. Ausführliche Fassung erschienen als DFKI Research Report RR-91-10, Kaiserslautern.

[7] F. Baader and P. Hanschke. Extensions of concept languages for a mechanical engineering application. In *Proceedings of the 16th German AI-Conference, GWAI-92*, volume 671 of *Lecture Notes in Computer Science*, pages 132–143, Bonn (Germany), 1993. Springer–Verlag.

[8] F. Baader and B. Hollunder. KRIS: Knowledge representation and inference system. *SIGART Bulletin*, 2(3):8–14, 1991. Special Issue on Implemented Knowledge Representation and Reasoning Systems.

[9] F. Baader and B. Hollunder. A terminological knowledge representation system with complete inference algorithms. In H. Boley and M. M. Richter, editors, *Processing Declarative Knowledge – Proc. of the International Workshop PDK'91*, volume 567 of *Lecture Notes in Artificial Intelligence*, pages 67–86, Kaiserslautern, Germany, July 1–3, 1991. Springer-Verlag, Berlin – Heidelberg – New York, 1991.

[10] F. Baader and U. Sattler. Description logics with concrete domains and aggregation. In H. Prade, editor, *Proceedings of the Thirteenth European Conference on Artificial Intelligence ECAI-98*, Brighton, August 23–28, 1998. John Wiley & Sons, New York, 1998.

[11] A. Borgida, R. J. Brachman, D. L. McGuiness, and L. Alpern Resnick. CLASSIC: A structural data model for objects. In *Proc. 1989 ACM SIGMOD International Conference on Management of Data*, pages 59–67, Portland, OR, 1989.

[12] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, (49):61–95, 1991.

[13] M. Egenhofer. Reasoning about Binary Topological Relations. In O. Günther and H.-J. Schek, editors, *Advances in Spatial Databases, Second Symposium, SSD'91, Zurich, Aug. 28-30, 1991*, volume 525 of *Lecture Notes in Computer Science*, pages 143–160. Springer-Verlag, Aug. 1991.

[14] M. Finger and D. M. Gabbay. Adding a temporal dimension to a logic system. *Journal of Logic Language and Information*, 1:203–233, 1992.

[15] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, San Francisco, 1979.

[16] V. Haarslev, C. Lutz, and R. Möller. A description logic with concrete domains and role-forming predicates. *Journal of Logic and Computation*, 9(3), 1999.

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

[18] B. Hollunder. *Algorithmic Foundations of Terminological Knowledge Representation Systems*. PhD thesis, Universität des Saarlandes, 1994.

[19] B. Hollunder and W. Nutt. Subsumption algorithms for concept languages. DFKI Research Report RR-90-04, Deutsches Forschungszentrum für Künstliche Intelligenz, Kaiserslautern, 1990.

[20] I. Horrocks. Using an expressive description logic: Fact or fiction? In A. Cohn, L. Schubert, and S.C.Shapiro, editors, *Principles of Knowledge Representation and Reasoning – Proc. of the Sixth International Conference KR'98*, pages 636–647, Trento, Italy, June 2–5, 1998. Morgan Kaufmann Publ. Inc., San Francicso, CA, 1998.

[21] G. Kamp and H. Wache. CTL - a description logic with expressive concrete domains. Technical Report LKI-M-96/01, Labor für Künstliche Intelligenz, Universität Hamburg, Germany, 1996.

[22] H. A. Kautz and P. B. Ladkin. Integrating metric and qualitative temporal reasoning. In AAAI [1], pages 241–246.

[23] P. B. Ladkin and R. D. Maddux. On binary constraint problems. *Journal of the ACM*, 41(3):435–469, 1994.

[24] C. Lutz, V. Haarslev, and R. Möller. A concept language with role-forming predicate restrictions. Technical Report FBI-HH-M-276/97, University of Hamburg, Computer Science Department, Hamburg, 1997.

[25] E. W. Mayr and A. R. Meyer. The complexity of the word problem for commutative semigroups and polynomial ideals. *Advanced Mathematics*, 46:305–329, 1982.

[26] I. Meiri. Combining qualitative and quantitative constraints in temporal reasoning. In AAAI [1], pages 260–267.

[27] J. Mylopoulos and R. Reiter, editors. *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence IJCAI-91*, Sydney, Australia, August 24–30, 1991. Morgan Kaufmann Publ. Inc., San Mateo, CA, 1991.

[28] D. A. Randell, Z. Cui, and A. G. Cohn. A spatial logic based on regions and connection. In B. Nebel, C. Rich, and W. Swartout, editors, *Principles of Knowledge Representation and Reasoning – Proc. of the Third International Conference KR'92*, pages 165–176, Cambridge, Mass., October 25–29, 1992. Morgan Kaufmann Publ. Inc., San Mateo, CA, 1992.

[29] W. J. Savitch. Relationsship between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.*, 4:177–192, 1970.

[30] K. D. Schild. A correspondence theory for terminological logics: Preliminary report. In Mylopoulos and Reiter [27], pages 466–471. Auch erschienen als KIT-Report 91, TU Berlin, Fachbereich Informatik.

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

[32] A. Schrijver. *Theory of Linear and Integer Programming*. Wiley, Chichester, England, 1986.

[33] M. Vilain, H. Kautz, and P. Van Beek. Constraint propagation algorithms for temporal reasoning: A revised report. In D. S. Weld and J. de Kleer, editors, *Readings in Qualitative Reasoning about Physical Systems*, pages 373–381. Kaufmann, San Mateo, CA, 1990.