Reasoning with Concrete Domains

Carsten Lutz
RWTH Aachen, LuFg Theoretical Computer Science
Ahornstr. 55, 52074 Aachen, Germany
clu@cantor.informatik.rwth-aachen.de

Abstract

Description logics are formalisms for the representation of and reasoning about conceptual knowledge on an abstract level. Concrete domains allow the integration of description logic reasoning with reasoning about concrete objects such as numbers, time intervals, or spatial regions. The importance of this combined approach, especially for building real-world applications, is widely accepted. However, the complexity of reasoning with concrete domains has never been formally analyzed and efficient algorithms have not been developed. This paper closes the gap by providing a tight bound for the complexity of reasoning with concrete domains and presenting optimal algorithms.

1 Introduction

Description logics are knowledge representation and reasoning formalisms dealing with conceptual knowledge on an abstract logical 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 [Baader and Hanschke, 1993], reasoning about aggregation in databases [Baader and Sattler, 1998], as well as temporal and spatial reasoning (see [Haarslev *et al.*, 1998] and [Lutz, 1998]). Many description logic systems such as e.g. CLASSIC and $\mathcal{KRIS}$ (see [Borgida *et al.*, 1989], [Baader and Hollunder, 1991], resp.), provide interfaces that allow the attachment of external reasoning facilities which deal 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 [Horrocks, 1998]) 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. An important reason why these systems fail to integrate concrete knowledge is that no complexity results and no efficient algorithms are available.

Baader and Hanschke [1991] extend description logics by concrete domains, a theoretically well-founded approach to integrated reasoning with abstract and concrete knowledge. On basis of the well-known description logic $\mathcal{ALC}$, they define the description logic $\mathcal{ALC}(\mathcal{D})$, which can be parameterized by a concrete domain $\mathcal{D}$. In this paper, we extend $\mathcal{ALC}(\mathcal{D})$ by the operators feature agreement and feature disagreement. This leads to the new logic $\mathcal{ALCF}(\mathcal{D})$, which combines $\mathcal{ALC}(\mathcal{D})$ with the logic $\mathcal{ALCF}$ [Hollunder and Nutt, 1990]. Algorithms for deciding the concept satisfiability and ABox consistency problems for the logic $\mathcal{ALCF}(\mathcal{D})$ are given. Furthermore, the complexity of reasoning with $\mathcal{ALCF}(\mathcal{D})$ is formally analyzed. Since reasoning with $\mathcal{ALCF}(\mathcal{D})$ involves a satisfiability check for the concrete domain, the complexity of the combined formalism depends on the complexity of reasoning in the concrete domain. The proposed algorithms are proved to need polynomial space which implies that, first, reasoning with $\mathcal{ALCF}(\mathcal{D})$ is PSPACE-complete provided that reasoning with the concrete domain is in PSPACE, and, second, the devised algorithms are optimal. The obtained complexity results carry over to the description logic $\mathcal{ALC}(\mathcal{D})$. The algorithmic techniques introduced in this paper are vital for efficient implementations of both $\mathcal{ALCF}(\mathcal{D})$ and $\mathcal{ALC}(\mathcal{D})$.

As a simple example illustrating the expressivity of $\mathcal{ALCF}(\mathcal{D})$, consider the concept $\mathit{Man} \sqcap \mathit{wife}{\downarrow}\mathit{boss} \sqcap \exists \mathit{wage}, (\mathit{wife}\ \mathit{wage}).\ >$. In this example, *Man* is a primitive concept, *wife* and *wage* are features (i.e., single valued roles), and $>$ is a concrete predicate. The given concept describes the set of men whose boss coincides with their wife and who, furthermore, have a higher wage than their wife. In this example, the wage of a person is knowledge of a concrete type while being a man is knowledge of a more abstract nature. The coincidence of wife and boss is described using the feature agreement operator $\downarrow$ and cannot be expressed in $\mathcal{ALC}(\mathcal{D})$. The syntax used is defined in the next section.

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

In this section, the description logic $\mathcal{ALCF}(\mathcal{D})$ is introduced. We start the formal specification by recalling the definition of a concrete domain given in [Baader and Hanschke, 1991].

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 in $\Phi_\mathcal{D}$ 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 problem for finite conjunctions of predicates is decidable.

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

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

- $\neg C$ (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).

A simple feature is a feature chain of length one. 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. As usual, a set theoretic semantics is given.

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 $f_1^\mathcal{I} \circ \cdots \circ f_k^\mathcal{I}$ 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:

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

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

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

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 $\mathcal{I}$.

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 $\Delta_\mathcal{D}$. Agreement and disagreement over concrete objects can be expressed by using a concrete domain which includes an equality predicate. Using disjunction, "global" agreement and disagreement over both the concrete and the abstract domain can then also be expressed (see [Lutz, 1998]). This approach was chosen since global agreement and disagreement are not considered to be very "natural" operators. We will now introduce the assertional formalism of $\mathcal{ALCF}(\mathcal{D})$.

Definition 4. Let O_D and O_A be disjoint sets of object names. Elements of O_D are called *concrete objects* and elements of O_A are called *abstract objects*. If C, R, f, and P are defined as in Definition 2, a and b are elements of O_A and $x, x_1, \ldots, x_n$ 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.

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. In the next section, an algorithm for deciding the consistency of $\mathcal{ALCF}(\mathcal{D})$ ABoxes is presented.

3 Algorithms

Completion algorithms, also known as tableau algorithms, are frequently used to decide concept satisfiability and ABox

consistency for various description logics. Completion algorithms work on (possibly generalized) ABoxes and 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. If the satisfiability of a concept C is to be decided, the ABox $\{a:C\}$ is considered. The algorithm repeatedly applies completion rules adding new axioms, and, by doing so, makes all knowledge implicitly contained in the ABox explicit. If the algorithm succeeds to construct an ABox which is complete (i.e., to which no more completion rules are applicable) and which does not contain an obvious contradiction, then $\mathcal{A}_0$ has a model. Otherwise, $\mathcal{A}_0$ does not have a model.

In [Hollunder and Nutt, 1990], a completion algorithm for deciding the satisfiability of $\mathcal{ALCF}$ concepts is given which can be executed in polynomial space. In [Baader and Hanschke, 1991], an algorithm for deciding the consistency of $\mathcal{ALC}(\mathcal{D})$ (i.e., $\mathcal{ALCF}(\mathcal{D})$ without feature agreement and disagreement) ABoxes is given. However, this algorithm needs exponential space in the worst case. This is due to the fact that the algorithm collects *all* axioms of the form $(x_1,\ldots,x_n):P$ (*concrete domain axioms*) obtained during rule application, conjoins them into one big conjunction c, and finally tests c for satisfiability w.r.t. the concrete domain. Unfortunately, the size of this conjunction may be exponential in the size of $\mathcal{A}_0$ (see [Lutz, 1998] for an example). To obtain a polynomial space algorithm for deciding the consistency of $\mathcal{ALCF}(\mathcal{D})$ ABoxes, the concrete domain satisfiability test has to be broken up into independent "chunks" of polynomial size.

The completion algorithm for deciding the consistency of $\mathcal{ALCF}(\mathcal{D})$ ABoxes is developed in two steps: First, an algorithm for deciding the satisfiability of $\mathcal{ALCF}(\mathcal{D})$ concepts is devised. Second, an algorithm is given which reduces ABox consistency to concept satisfiability by constructing a number of "reduction concepts" for a given ABox $\mathcal{A}_0$. A similar reduction can be found in [Hollunder, 1994].

Before giving a formal description of the completion algorithms themselves, the completion rules are defined. To define the rules in a succinct way, the functions $succ_{\mathcal{A}}$ and $chain_{\mathcal{A}}$ are introduced. Let $\mathcal{A}$ be an ABox. 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 $\mathcal{A}$. 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 $\mathcal{A}$ 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:

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

Now, the set of completion rules can be formulated. Please note that the completion rule R$\sqcup$ is nondeterministic, i.e., there is more than one possible outcome of a rule application.

Definition 5. The following *completion rules* replace a given ABox $\mathcal{A}$ nondeterministically by an ABox $\mathcal{A}'$. An ABox $\mathcal{A}$ is said to contain a *fork* (for a feature f) iff it contains the two axioms $(a,b):f$ and $(a,c):f$ or the two axioms $(a,x):f$ and $(a,y):f$, where $b,c \in \mathsf{O}_A$ and $x,y \in \mathsf{O}_D$. A fork can be eliminated by replacing all occurrences of c in $\mathcal{A}$ with b, or of x with y, resp. It is assumed that forks are eliminated as soon as they appear (as part of the rule application) with the proviso that newly generated objects are replaced by older ones and not vice versa. In the following, C and D denote concepts, $\hat{R}$ 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 objects from O_A, and $x_1,\ldots,x_n$ objects from O_D.

R$\sqcap$ The conjunction rule.
If $a:C \sqcap D \in \mathcal{A}$, $\{a:C,\ a:D\} \not\subseteq \mathcal{A}$
then $\mathcal{A}' = \mathcal{A} \cup \{a:C,\ a:D\}$

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

Rr$\exists$C, Rf$\exists$C The role/feature exists restriction rule.
If $a:\exists\hat{R}.C \in \mathcal{A}$, $\nexists b \in \mathsf{O}_A:\ \{(a,b):\hat{R},\ b:C\} \subseteq \mathcal{A}$
then $\mathcal{A}' = \mathcal{A} \cup \{(a,b):\hat{R},\ b:C\}$ where $b \in \mathsf{O}_A$ is fresh in $\mathcal{A}$.
This is the Rr$\exists$C rule. To obtain Rf$\exists$C, replace $\hat{R}$ by f.

Rr$\forall$C, Rf$\forall$C The role/feature value restriction rule.
If $a:\forall\hat{R}.C \in \mathcal{A}$, $\exists b \in O_A:\ (a,b):\hat{R} \in \mathcal{A} \wedge b:C \notin \mathcal{A}$
then $\mathcal{A}' = \mathcal{A} \cup \{b:C\}$
This is the Rr$\forall$C rule. To obtain Rf$\forall$C, replace $\hat{R}$ by f.

R$\exists$P The predicate exists restriction rule (may create forks).
If $a:\exists u_1,\ldots,u_n.P \in \mathcal{A}$, $\nexists x_1,\ldots,x_n \in \mathsf{O}_D:$
$(succ_{\mathcal{A}}(a,u_1) = x_1 \wedge \ \ldots\ \wedge succ_{\mathcal{A}}(a,u_n) = x_n \wedge$
$(x_1,\ldots,x_n):P \in \mathcal{A})$
then $\mathcal{C}_0 := \mathcal{A} \cup \{(x_1,\ldots,x_n):P\}$
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$\downarrow$ The agreement rule (may create forks).
If $a:u_1{\downarrow}u_2 \in \mathcal{A}$, $\nexists b \in \mathsf{O}_A:\ succ_{\mathcal{A}}(a,u_1) = succ_{\mathcal{A}}(a,u_2) = b$
then $\mathcal{C} = \mathcal{A} \cup chain_{\mathcal{A}}(a,b,u_1)$ where $b \in \mathsf{O}_A$ is fresh in $\mathcal{A}$.
$\mathcal{A}' = \mathcal{C} \cup chain_{\mathcal{C}}(a,b,u_2)$

R$\uparrow$ The disagreement rule (may create forks).
If $a:u_1{\uparrow}u_2 \in \mathcal{A}$, $\nexists b_1,b_2 \in \mathsf{O}_A:$
$(succ_{\mathcal{A}}(a,u_1) = b_1 \wedge succ_{\mathcal{A}}(a,u_2) = b_2 \wedge b_1 \neq b_2 \in \mathcal{A})$
then $\mathcal{C} = \mathcal{A} \cup chain_{\mathcal{A}}(a,b_1,u_1)$
$\mathcal{A}' = \mathcal{C} \cup chain_{\mathcal{C}}(a,b_2,u_2) \ \cup\ \{b_1 \neq b_2\}$
where $b_1,b_2 \in \mathsf{O}_A$ 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$\exists$C rule are generating. Application of the rules Rf$\exists$C, R$\exists$P, R$\downarrow$, R$\uparrow$ are usually generating but may be non-generating if fork elimination takes place.

A formalized notion of contradictory and of complete ABoxes is introduced in the following.

Definition 6. Let the same naming conventions be given as in Definition 5. An ABox $\mathcal{A}$ is called *concrete domain satisfiable* iff 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.

- *Primitive clash*: $\{a:C,\ a:\neg C\} \subseteq \mathcal{A}$
- *Feature domain clash*: $\{(a,x):f,\ (a,b):f\} \subseteq \mathcal{A}$

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($\{b : D\} \cup \{b : E \mid a : \forall \hat{R}.E \in \mathcal{A}'\}$)
      returns *inconsistent* **then**
      return *inconsistent*
  return *consistent*

define procedure feature-complete($\mathcal{A}$)
  while a rule r from the set {R⊓, R⊔, Rf∃C, Rf∀C, R∃P, R↓, R↑} is applicable to $\mathcal{A}$, **do**
    $\mathcal{A} := \mathcal{A} \cup$ *apply*$(\mathcal{A}, r)$
  return $\mathcal{A}$

Figure 1: The sat algorithm.

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

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

We are now ready to define the completion algorithm sat for deciding the satisfiability of $\mathcal{ALCF}(\mathcal{D})$ concepts. Sat takes an ABox $\{a : C\}$ as input, where C has to be in *negation normal form*, i.e., negation is allowed only in front of concept names. Conversion to NNF can be done by exhaustively applying appropriate rewrite rules to push negation inwards. We only give the conversion rules needed for the new constructors feature agreement and feature disagreement, and refer to [Lutz, 1998] for the $\mathcal{ALC}(\mathcal{D})$ rule set. For a feature chain $u = f_1 \cdots f_k$, set $\lambda(u) := \exists f_1.\top_{\mathcal{D}} \sqcup \exists f_1 f_2.\top_{\mathcal{D}} \sqcup \ldots \sqcup \exists f_1 \cdots f_{k-1}.\top_{\mathcal{D}}$.

$$\neg(u_1{\downarrow}u_2) \Rightarrow u_1{\uparrow}u_2 \sqcup \exists u_1.\top_{\mathcal{D}} \sqcup \exists u_2.\top_{\mathcal{D}} \sqcup \forall u_1.\bot \sqcup \forall u_2.\bot \sqcup \lambda(u_1) \sqcup \lambda(u_2)$$

$$\neg(u_1{\uparrow}u_2) \Rightarrow u_1{\downarrow}u_2 \sqcup \exists u_1.\top_{\mathcal{D}} \sqcup \exists u_2.\top_{\mathcal{D}} \sqcup \forall u_1.\bot \sqcup \forall u_2.\bot \sqcup \lambda(u_1) \sqcup \lambda(u_2)$$

Any $\mathcal{ALCF}(\mathcal{D})$ concept can be converted into an equivalent concept in NNF in linear time. Some comments about the application of nondeterministic completion rules are in order. The application of the nondeterministic rule R⊔ yields more than one possible outcome. It is not specified which possibility is chosen in a given run of a completion algorithm. This means that the algorithms to be specified are nondeterministic algorithms. Such algorithms returns a positive result if there is *any* way to make the nondeterministic decisions such that a positive result is obtained.

The satisfiability algorithm makes use of two auxiliary functions which will be described only informally. The function *apply* takes two arguments, an ABox $\mathcal{A}$ and a completion rule r. It applies r once to arbitrary axioms from $\mathcal{A}$ matching r's premise and (nondeterministically) returns the new axioms generated by the rule application. The function satisfiable? takes as arguments a concrete domain $\mathcal{D}$ and a set

define procedure ABox-cons($\mathcal{A}$)
  eliminate forks in $\mathcal{A}$ (see Definition 5)
  $\mathcal{A} :=$ 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 $a : \exists \hat{R}.D \in \mathcal{A}$, where $\hat{R}$ is a role, **do**
    Let b be an object name from O_A.
    if sat($\{b : (D \sqcap \bigsqcap_{a:\forall R.E \in \mathcal{A}} E)\}$)
      returns *inconsistent* **then**
      return *inconsistent*
  return *consistent*

define procedure preprocess($\mathcal{A}$)
  while a rule r from the set {R⊓, R⊔, Rr∀C, Rf∃C, Rf∀C, R∃P, R↓, R↑} is applicable to $\mathcal{A}$, **do**
    $\mathcal{A} := \mathcal{A} \cup$ *apply*$(\mathcal{A}, r)$
  return $\mathcal{A}$

Figure 2: The ABox-cons algorithm.

$\mathcal{C}$ 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. The sat algorithm is given in figure 1. Based on sat, we define the ABox-cons algorithm for deciding ABox consistency. This algorithm can be found in figure 2.

A formal correctness proof for the algorithms is omitted for the sake of brevity and can be found in [Lutz, 1998]. A short, informal discussion of the employed strategies is given instead. The sat algorithm performs depth-first search over role successors. This technique, first introduced by Schmidt-Schauß and Smolka [1991] for the logic $\mathcal{ALC}$, allows to keep only a polynomial fragment (called "trace") of the model in memory, although the total size of the model may be exponential. Tracing algorithms usually expand the axioms belonging to a single object, only, and make a recursive call for each role successor of this object. This is not feasible in the case of $\mathcal{ALCF}(\mathcal{D})$ since more than a single object may have to be considered when checking concrete domain satisfiability. The central idea to overcome this problem is to expand axioms not for single objects but for "clusters" of objects which are connected by features. This is done by the feature-complete function. During cluster expansion, chunks of concrete domain axioms are collected. Any such chunk can separately be checked for satisfiability. To see this, it is important to note that roles are not allowed inside the predicate operator, and thus concrete domain axioms cannot involve objects from different clusters (which are connected by roles). A similar strategy is employed for $\mathcal{ALCF}$ in [Hollunder and Nutt, 1990]. The ABox-cons reduces ABox consistency to satisfiability by performing preprocessing on the initial ABox and then constructing a reduction concept for each role successor of any object in the resulting ABox. In the next section, the complexity of both algorithms is analyzed.

4 Complexity of Reasoning

To characterize space requirements, a formal notion for the size of an ABox is introduced.

Definition 7. The *size* $||C||$ of a concept C is defined inductively. Let C and D be concepts, A a concept name, 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.

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

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 $\mathcal{A}$.

For the analysis of the space needed by sat, two lemmata are needed.

Lemma 8. *For any input $\mathcal{A}$, the function* feature-complete *constructs an ABox $\mathcal{A}'$ with $||\mathcal{A}'|| \leq ||\mathcal{A}||^2 + ||\mathcal{A}||$.*

Proof: The upper bound for the size of $\mathcal{A}'$ 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 $||\alpha|| \leq ||\mathcal{A}||$.

The second point is obvious, but the first one needs to be proven. The rules Rr∃C and Rr∀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∀C rule since there is at most one axiom $(a, b) : f$ per feature f and object a. We make the simplifying assumption that the premise of the Rf∀C rule does *only* contain the axiom $a : \forall f.C$, i.e., that it is applied to *every* axiom of this form regardless if there is an axiom $(a, b) : f$ or not. This may result in too high an estimation of the number of generated axioms but not in one that is too low. We now prove 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.

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 proved 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 a concept name. 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⊓ 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 cases for the remaining operators $D \sqcup E$, $\exists f.C$, and $\forall f.C$ are analogous. Because of the simplifying assumptions made, the $\forall f.C$ case does not need a special treatment. ∎

Lemma 9. *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. 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. ∎

The space requirements of sat can now be settled.

Proposition 10. *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: We will first analyze the maximum size of the arguments passed to sat in recursive calls. The argument to sat is an ABox which contains axioms $a : C$ for a single object a. It is obvious that there can be at most as many such axioms per object as there are distinct (sub)concepts appearing in $\mathcal{A}_0$. This number is bounded by $||\mathcal{A}_0||$. Furthermore, the size of any axiom is at most $||\mathcal{A}_0||$. It follows that the maximum size of arguments given in a recursive call is $||\mathcal{A}_0||^2$. Using feature-complete, the argument ABox is extended by new axioms. Combining the argument size with the result from Lemma 8, we find that the maximum size of ABoxes constructed during recursive calls is $||\mathcal{A}_0||^4 + ||\mathcal{A}_0||^2$. Together with Lemma 9, it follows that sat can be executed in $||\mathcal{A}_0||^5 + ||\mathcal{A}_0||^3$ space. ∎

This result completes the analysis of the sat algorithm. The ABox-cons algorithm performs some preprocessing on the input ABox and then repeatedly calls sat. Its space requirements are investigated in the next Proposition.

Proposition 11. *Started on input $\mathcal{A}$,* ABox-cons *can be executed in space polynomial in $||\mathcal{A}||$, provided that this also holds for the function* satisfiable?.

Proof: It was already proven that sat can be executed in polynomial space if this also holds for satisfiable?. Thus, it remains to be shown that, for an ABox $\mathcal{A}$, the size of $\mathcal{A}' := \text{preprocess}(\mathcal{A})$ is polynomial in $||\mathcal{A}||$. We will only give a sketch of the proof, for the full version see [Lutz, 1998]. Objects are called *old* if they are used in $\mathcal{A}$ and *new* if they are used in $\mathcal{A}'$ but not in $\mathcal{A}$. The proof relies on the fact that the preprocess function is identical to the feature-complete function except that preprocess does also apply the Rr∀C rule. An upper bound for the number of Rr∀C applications performed by preprocess can be given as follows: If Rr∀C is applied to axioms $a : \forall \hat{R}.C$ and $(a, b) : \hat{R}$, then both a and b are old objects. This is the case since preprocess does not apply Rr∃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 which means that the number of Rr∀C applications is bounded by $||\mathcal{A}||^2$. Together with Lemma 8, it can be shown that $||\mathcal{A}'||$ is of order $\mathcal{O}(||\mathcal{A}||^k)$. ∎

The results just obtained allows us to determine the formal complexity of reasoning with concrete domains.

Theorem 12. *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. *Satisfiability and subsumption of $\mathcal{ALCF}(\mathcal{D})$ concepts.*
3. *Satisfiability and subsumption of $\mathcal{ALC}(\mathcal{D})$ concepts.*
4. *Consistency of $\mathcal{ALC}(\mathcal{D})$ and $\mathcal{ALCF}$ ABoxes.*

If the satisfiability test of $\mathcal{D}$ is in a complexity class X *with* PSPACE $\subseteq$ X*, then all of the above problems are* PSPACE*-hard.*

Proof: (1) Since $\mathcal{ALC}$ is a proper subset of $\mathcal{ALCF}(\mathcal{D})$ and the satisfiability problem for $\mathcal{ALC}$ is PSPACE-complete [Schmidt-Schauß and Smolka, 1991], deciding the consistency of $\mathcal{ALCF}(\mathcal{D})$ ABoxes is PSPACE-hard. It remains to be shown that it is in PSPACE if this is also the case for the concrete domain satisfiability test. This follows from Proposition 11 together with the well-known fact that PSPACE = NPSPACE [Savitch, 1970]. (2) is true since satisfiability as well as subsumption can be reduced to ABox consistency, cf. Section 2. (3) and (4) hold since $\mathcal{ALC}$ is a proper subset of both logics $\mathcal{ALC}(\mathcal{D})$ and $\mathcal{ALCF}$ which are in turn proper subsets of $\mathcal{ALCF}(\mathcal{D})$. ∎

Examples of useful concrete domains for which the satisfiability test is in PSPACE are given in [Lutz, 1998].

5 Conclusions and Future Work

We have presented optimal algorithms for deciding the concept satisfiability and the ABox consistency problems for the logic $\mathcal{ALCF}(\mathcal{D})$. In contrast to existing decision procedures, the devised algorithms can be executed in polynomial space provided that this does also hold for the concrete domain satisfiability test. Based on this result, it was proven that reasoning with $\mathcal{ALCF}(\mathcal{D})$ is a PSPACE-complete problem. The rule application strategy used by the proposed algorithm is vital for efficient implementations of description logics with concrete domains. An interesting new result in this context is that in the case of $\mathcal{ALC}(\mathcal{D})$ and $\mathcal{ALCF}$, satisfiability w.r.t. TBoxes is a NEXPTIME-hard problem [Lutz, 1998]. As future work, we will consider the combination of concrete domains with more expressive description logics. Furthermore, the logic $\mathcal{ALCF}(\mathcal{D})$ seems to be a promising candidate for the reduction of some temporal description logics in order to obtain complexity results for them.

Acknowledgments I would like to thank Ulrike Sattler 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

[Baader and Hanschke, 1991] F. Baader and P. Hanschke. A scheme for integrating concrete domains into concept languages. In John Mylopoulos and Ray Reiter, editors, *Proc. of IJCAI-91*, pages 452–457, Sydney, Australia, August 24–30, 1991.

[Baader and Hanschke, 1993] F. Baader and P. Hanschke. Extensions of concept languages for a mechanical engineering application. In *Proc. of GWAI-92*, volume 671 of *LNCS*, pages 132–143, Bonn, Germany, 1993. Springer–Verlag.

[Baader and Hollunder, 1991] 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.

[Baader and Sattler, 1998] F. Baader and U. Sattler. Description logics with concrete domains and aggregation. In Henri Prade, editor, *Proc. of ECAI-98*, Brighton, August 23–28, 1998. John Wiley & Sons, New York, 1998.

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

[Haarslev *et al.*, 1998] V. Haarslev, C. Lutz, and R. Möller. A description logic with concrete domains and role-forming predicates. *Journal of Logic and Computation*, 1998. To appear.

[Hollunder and Nutt, 1990] B. Hollunder and W. Nutt. Subsumption algorithms for concept languages. DFKI Research Report RR-90-04, German Research Center for Artificial Intelligence, Kaiserslautern, 1990.

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

[Horrocks, 1998] I. Horrocks. Using an expressive description logic: Fact or fiction? In A.G. Cohn, L.K. Schubert, and S.C. Shapiro, editors, *Proc. of KR'98*, pages 636–647, Trento, Italy, 1998. Morgan Kaufmann Publ. Inc., San Francicso, CA, 1998.

[Lutz, 1998] C. Lutz. The complexity of reasoning with concrete domains. LTCS-Report 99-01, LuFG Theoretical Computer Science, RWTH Aachen, Germany, 1999.

[Lutz, 1998] C. Lutz. On the Complexity of Terminological Reasoning. LTCS-Report 99-04, LuFG Theoretical Computer Science, RWTH Aachen, Germany, 1999. To appear.

[Sattler, 1996] U. Sattler. A concept language extended with different kinds of transitive roles. In G. Görz and S. Hölldobler, editors, *20. Deutsche Jahrestagung für KI*, volume 1137 of *Lecture Notes in Artificial Intelligence*, 1996.

[Savitch, 1970] W. J. Savitch. Relationsship between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4:177–192, 1970.

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