PSPACE Reasoning with the Description Logic $\mathcal{ALCF}(\mathcal{D})$

CARSTEN LUTZ, *Institute for Theoretical Computer Science, Technical University Dresden, 01062 Dresden, Germany.*
E-mail: lutz@tcs.inf.tu-dresden.de

Abstract

Description Logics (DLs), a family of formalisms for reasoning about conceptual knowledge, can be extended with concrete domains to allow an adequate representation of "concrete qualities" of real-worlds entities such as their height, temperature, duration, and size. In this paper, we study the complexity of reasoning with the basic DL with concrete domains $\mathcal{ALC}(\mathcal{D})$ and its extension with so-called feature agreements and disagreements $\mathcal{ALCF}(\mathcal{D})$. We show that, for both logics, the standard reasoning tasks concept satisfiability, concept subsumption, and ABox consistency are PSPACE-complete if the concrete domain $\mathcal{D}$ satisfies some natural conditions.

Keywords: Description Logics, Concrete Domains, Feature (Dis)Agreements, Computational Complexity

1 Motivation

Description Logics (DLs) are a popular family of logical formalisms for the representation of and reasoning about conceptual knowledge [8]. The basic entity for knowledge representation with DLs are so-called concepts which can be understood as logical formulas and are constructed from concept names (unary predicates), role names (binary relations), and concept constructors. For example, the following concept is formulated in the basic propositionally closed DL $\mathcal{ALC}$ [39] and describes processes that are supervised by a human operator and involve only workpieces that are not radioactive:

$$\mathsf{Process} \sqcap \exists \mathsf{operator}.\mathsf{Human} \sqcap \forall \mathsf{workpiece}.\neg \mathsf{Radioactive}.$$

In this concept, $\mathsf{Process}$, Human, and $\mathsf{Radioactive}$ are concept names while $\mathsf{operator}$ and $\mathsf{workpiece}$ are role names.

A major limitation of knowledge representation with Description Logics such as $\mathcal{ALC}$ is that "concrete qualities" of real world entities, such as their weight, temperature, and spatial extension, cannot be adequately represented. For example, $\mathcal{ALC}$ does not offer suitable means of expressivity for extending the above description of a process with information about its cost and duration, or about the relationship between the process' cost and the hourly wage of its operator. To allow an adequate representation of concrete qualities of real-world entities, Description Logics are frequently extended by so-called *concrete domains*, which have first been proposed by Baader and Hanschke in [4] and then further developed in several directions, c.f. the survey article [32]. A concrete domain consists of a set such as the natural numbers and a set of predicates such as the unary "$=_{60}$" and the binary "$>$" with the obvious, fixed extension. The integration of concrete domains into the Description Logic $\mathcal{ALC}$ is achieved by adding

1. so-called *abstract features*, which are functional relations;
2. so-called *concrete features*, which are (partial) functions associating values from the concrete domain (e.g., natural numbers) to logical objects;
3. a concrete domain-based concept constructor.

The DL that is obtained by extending $\mathcal{ALC}$ in this way is called $\mathcal{ALC}(\mathcal{D})$, where $\mathcal{D}$ denotes a concrete domain that can be viewed as a parameter to the logic. For example, when using a suitable concrete domain $\mathcal{D}$, we can extend the above process description as desired: the $\mathcal{ALC}(\mathcal{D})$-concept

$$\mathsf{Process} \sqcap \exists \mathsf{duration}.{=_{60}} \sqcap \exists \mathsf{cost}, \mathsf{operator\ wage}.{>}$$

describes a process whose duration is 60 minutes and which costs more than the (hourly) wage of its operator. Here, the second and third conjunct are instances of the concrete domain concept constructor, operator is an abstract feature, and duration, cost, and wage are concrete features.

The representation of concrete qualities has been identified as a crucial task for a vast number of applications such as mechanical engineering [6], temporal and spatial reasoning [16, 27], the semantic web [23, 24], and reasoning about entity relationship (ER) diagrams [31]. Consequently, apart from $\mathcal{ALC}(\mathcal{D})$ many other Description Logics with concrete domains have been proposed [16, 18, 20, 24, 27, 30, 29] and several implemented Description Logic reasoners such as CLASSIC [11] and RACER [17] provide for some kind of concrete domain. However, despite the considerable interest in DLs with concrete domains and the fact that complexity analysis plays an important role in the area of Description Logics, only very recently researchers have begun to investigate the computational complexity of reasoning with such logics [30]. The current paper is devoted to *establishing tight complexity bounds for reasoning with the fundamental Description Logic with concrete domains* $\mathcal{ALC}(\mathcal{D})$. More precisely, we do not only consider the DL $\mathcal{ALC}(\mathcal{D})$, but also its extension with so-called *feature agreements* and *feature disagreements*, two concept constructors that are quite closely related to concrete domains. Using feature (dis)agreements, one can for example describe processes that have two subprocesses, one of which works on the same workpiece as the mother process, and the other on a different one:

$$\mathsf{Process} \sqcap (\mathsf{workpiece} \downarrow \mathsf{subprocess1\ workpiece}) \sqcap (\mathsf{workpiece} \uparrow \mathsf{subprocess2\ workpiece}).$$

In this concept, the second conjunct uses the feature agreement constructor, the third conjunct uses the feature disagreement constructor, and all lowercase names denote abstract features.

There are several motivations for combining concrete domains and feature (dis)agreements in a single DL. First, there exists an obvious syntactic similarity between feature (dis)agreements and the concrete domain concept constructor: both take sequences of features as arguments. As we shall see in this paper, the similarity between concrete domains and feature (dis)agreements is not only syntactical: they are also amenable to similar algorithmic techniques. Second, the Description Logic $\mathcal{ALCF}(\mathcal{D})$ resulting from the extension of $\mathcal{ALC}(\mathcal{D})$ with feature (dis)agreements has already found applications in knowledge representation [25]. And third, the PSPACE-completeness result for reasoning with $\mathcal{ALCF}(\mathcal{D})$ proved in Section 3 allows to show PSPACE-completeness of a well-known temporal Description Logic [3].

Let us now outline the organization of this paper and describe the obtained results in more detail.

In Section 2, we formally introduce concrete domains and the Description Logics $\mathcal{ALC}(\mathcal{D})$ and $\mathcal{ALCF}(\mathcal{D})$. Some example concrete domains are defined.

In Section 3, tight PSPACE complexity bounds for the satisfiability of $\mathcal{ALC}(\mathcal{D})$-concepts and $\mathcal{ALCF}(\mathcal{D})$-concepts are established. More precisely, we devise a tableau algorithm for deciding satisfiability of $\mathcal{ALCF}(\mathcal{D})$-concepts which uses the so-called *tracing* technique. This algorithm yields a PSPACE upper bound for $\mathcal{ALCF}(\mathcal{D})$-concept satisfiability if the following conditions are satisfied:

- deciding the satisfiability of finite conjunctions of predicates from the concrete domain $\mathcal{D}$ (this task is called "$\mathcal{D}$-satisfiability" in what follows) is in PSPACE;
- the concrete domain is "admissible", i.e., it satisfies some weak closure conditions which, in this paper, we will generally assume to hold.

The corresponding PSPACE lower bound is easily obtained since $\mathcal{ALC}$-concept satisfiability is already PSPACE-hard [39]. Hence, both $\mathcal{ALC}(\mathcal{D})$-concept satisfiability and $\mathcal{ALCF}(\mathcal{D})$-concept satisfiability are PSPACE-complete if $\mathcal{D}$-satisfiability is in PSPACE. Since concept subsumption, another important reasoning task for Description Logics, can easily be reduced to concept (un)satisfiability and vice versa, we also obtain that $\mathcal{ALC}(\mathcal{D})$-concept subsumption and $\mathcal{ALCF}(\mathcal{D})$-concept subsumption are PSPACE-complete if $\mathcal{D}$-satisfiability is in PSPACE. Note that adding concrete domains and feature (dis)agreements to $\mathcal{ALC}$ does thus not increase the complexity of reasoning. This is particularly interesting since there exist several seemingly "harmless" means of expressivity like acyclic TBoxes and inverse roles, whose addition to $\mathcal{ALC}(\mathcal{D})$ makes reasoning significantly more difficult—namely NEXPTIME-complete [28, 30, 1]. Thus, the logic $\mathcal{ALCF}(\mathcal{D})$ is situated on the boundary of polynomial space complexity.

Section 4 is devoted to extending the results from Section 3 to another standard reasoning task called ABox consistency. ABoxes are commonly used to describe snapshots of the real world [7, 12, 17, 38, 41]. For example, the following $\mathcal{ALC}(\mathcal{D})$-ABox describes a process a and its subprocess b:

$$a : \mathsf{Process} \quad b : \mathsf{Process} \quad (a, b) : \mathsf{subprocess} \quad (a, x) : \mathsf{duration} \quad x : =_{60}$$

We use the *precompletion* technique from [13, 21] to show that $\mathcal{ALCF}(\mathcal{D})$-ABox consistency is PSPACE-complete if $\mathcal{D}$-satisfiability is in PSPACE. As in the case of concept satisfiability, this implies that the same holds for $\mathcal{ALC}(\mathcal{D})$-ABox consistency.

In Section 5, we demonstrate the relevance of the results obtained in Sections 3 and 4 by considering two example concrete domains: the concrete domain A based on the rational numbers with predicates such as $<_{27}$, $\geq$, and $+$; and the concrete domain S based on the set of regions in two-dimensional space with a binary predicate for each of the well-known RCC8 topological relations [10]. We show that both A-satisfiability and S-satisfiability is in NP and thus obtain that, for $\mathcal{D} \in \{\mathsf{A}, \mathsf{S}\}$, $\mathcal{ALCF}(\mathcal{D})$-concept satisfiability, $\mathcal{ALCF}(\mathcal{D})$-concept subsumption, and $\mathcal{ALCF}(\mathcal{D})$-ABox consistency are PSPACE-complete.

The paper ends with a conclusion in Section 6.

2 Preliminaries

We start this section with introducing concrete domains formally, then define some example concrete domains, and finally describe the Description Logic $\mathcal{ALCF}(\mathcal{D})$ in detail.

DEFINITION 2.1 (Concrete Domain)
A *concrete domain* $\mathcal{D}$ is a pair $(\Delta_\mathcal{D}, \Phi_\mathcal{D})$, where $\Delta_\mathcal{D}$ is a set and $\Phi_\mathcal{D}$ a set of predicate names. Each predicate name $P \in \Phi_\mathcal{D}$ is associated with an arity n and an n-ary predicate $P^\mathcal{D} \subseteq \Delta_\mathcal{D}^n$. Let V be a set of variables. A predicate conjunction of the form

$$c = \bigwedge_{i<k} (x_0^{(i)}, \ldots, x_{n_i}^{(i)}) : P_i,$$

where P_i is an n_i-ary predicate for $i < k$ and the $x_j^{(i)}$ are variables from V, is called *satisfiable* iff there exists a function δ mapping the variables in c to elements of $\Delta_\mathcal{D}$ such that $(\delta(x_0^{(i)}), \ldots, \delta(x_{n_i}^{(i)})) \in P_i^\mathcal{D}$ for each $i < k$. Such a function is called a *solution* for c. A concrete domain $\mathcal{D}$ is called *admissible* if the following conditions are satisfied:

1. $\Phi_\mathcal{D}$ contains a name $\top_\mathcal{D}$ for $\Delta_\mathcal{D}$;
2. $\Phi_\mathcal{D}$ is closed under negation, i.e., for each n-ary predicate $P \in \Phi_\mathcal{D}$, we find another predicate $\overline{P} \in \Phi_\mathcal{D}$ of arity n such that $\overline{P}^\mathcal{D} = \Delta_\mathcal{D}^n \setminus P^\mathcal{D}$;
3. the satisfiability problem for finite conjunctions of predicates is decidable.

When devising algorithms for reasoning with Description Logics that are equipped with a concrete domain $\mathcal{D}$, one important subtask usually is to decide the satisfiability of finite conjunctions of predicates from $\Phi_\mathcal{D}$ as described in Definition 2.1 [4, 30]. For brevity, we refer to this task as $\mathcal{D}$-satisfiability. It is obvious that $\mathcal{D}$-satisfiability should be decidable if the concrete domain $\mathcal{D}$ is to be used in a DL reasoning algorithm. However, usually the slightly stronger requirement that $\mathcal{D}$ should be admissible is adopted. In this article, we follow this tradition and generally assume concrete domains to be admissible.

Before we proceed to defining the Description Logic $\mathcal{ALCF}(\mathcal{D})$ itself, let us introduce two example concrete domains, an arithmetic one and a spatial one. The arithmetic concrete domain A is defined by setting $\Delta_\mathsf{A} := \mathbb{Q}$ (i.e., the set of rational numbers), and defining Φ_A as the (smallest) set containing the following predicates:

- a unary predicate $\top_\mathsf{A}$ with $(\top_\mathsf{A})^\mathsf{A} = \mathbb{Q}$ and a unary predicate $\bot_\mathsf{A}$ with $(\bot_\mathsf{A})^\mathsf{A} = \emptyset$;
- unary predicates int and $\overline{\mathsf{int}}$ with $(\mathsf{int})^\mathsf{A} = \mathbb{Z}$ (where $\mathbb{Z}$ denotes the integers) and $(\overline{\mathsf{int}})^\mathsf{A} = \mathbb{Q} \setminus \mathbb{Z}$;
- unary predicates P_q for each $P \in \{<, \leq, =, \neq, \geq, >\}$ and each $q \in \mathbb{Q}$ with $(P_q)^\mathsf{A} = \{q' \in \mathbb{Q} \mid q' \; P \; q\}$;
- binary predicates $<, \leq, =, \neq, \geq, >$ with the obvious extension;

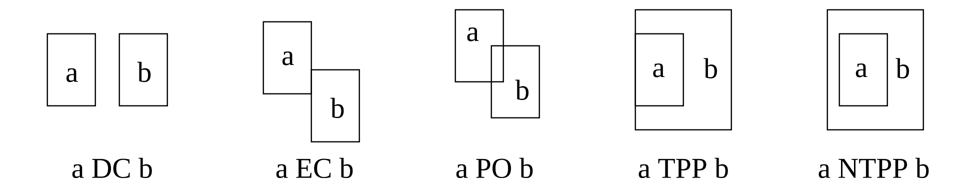

FIG. 1. The RCC8 relations in two-dimensional space.

- ternary predicates $+$ and $\overline{+}$ with $(+)^{\mathsf{A}} = \{(q,q',q'') \in \mathbb{Q}^3 \mid q + q' = q''\}$ and $(\overline{+})^{\mathsf{A}} = \mathbb{Q}^3 \setminus (+)^{\mathsf{A}}$.

As an example for an (unsatisfiable) conjunction of A-predicates, consider the following one:

$$=_3(x) \wedge >_1(y) \wedge \mathsf{int}(y) \wedge +(x,y,z) \wedge *(x,y,z') \wedge \geq(z,z').$$

It is easily checked that the concrete domain A satisfies Conditions 1 and 2 of admissibility (Condition 3 will be treated in Section 5). The other concrete domain considered in this paper is related to the RCC-8 calculus and is called S. RCC-8 provides a set of eight jointly exhaustive and pairwise disjoint relations that describe the possible relationships between any two regular closed regions[1] in a topological space [34, 10, 36]. For 2D space, these relations are illustrated in Figure 1, where the equality relation EQ, the inverse TPPI of TPP, and the inverse NTPPI of NTPP have been omitted. The concrete domain S is defined by setting Δ_{S} to the set $\mathcal{RC}_{\mathbb{R}^2}$ of all regular closed subsets of $\mathbb{R}^2$ and defining Φ_{S} as the (smallest) set containing the following predicates:

- a unary predicate $\top_{\mathsf{S}}$ with $(\top_{\mathsf{S}})^{\mathsf{S}} = \mathcal{RC}_{\mathbb{R}^2}$ and a unary predicate $\bot_{\mathsf{S}}$ with $(\bot_{\mathsf{S}})^{\mathsf{S}} = \emptyset$;
- binary predicates rel and $\overline{\mathsf{rel}}$ for each of the topological relations rel such that $(\mathsf{rel})^{\mathsf{S}} = \{(r_1, r_2) \in \mathcal{RC}_{\mathbb{R}^2} \times \mathcal{RC}_{\mathbb{R}^2} \mid r_1 \ \mathsf{rel} \ r_2\}$.

An example (unsatisfiable) S-conjunction is

$$\top_{\mathsf{S}}(x) \wedge DC(x,y) \wedge EC(y,z) \wedge NTPP(z,x) \wedge \overline{PO}(y,y).$$

It is easily checked that S satisfies Conditions 1 and 2 of admissibility. For Property 3, we again refer to Section 5.

Based on concrete domains, we can now define $\mathcal{ALCF}(\mathcal{D})$-concepts.

DEFINITION 2.2 ($\mathcal{ALCF}(\mathcal{D})$ syntax)
Let $\mathsf{N_C}$, $\mathsf{N_R}$, and $\mathsf{N_{cF}}$ be pairwise disjoint and countably infinite sets of *concept names*, *role names*, and *concrete features*. Furthermore, let $\mathsf{N_{aF}}$ be a countably infinite subset of $\mathsf{N_R}$. The elements of $\mathsf{N_{aF}}$ are called *abstract features*. An *abstract path* p is a composition $f_1 \cdots f_n$ of n abstract features ($n \geq 1$). A *concrete path* u is a composition $f_1 \cdots f_n g$ of n abstract features $f_1, \ldots, f_n$ ($n \geq 0$) and a concrete feature g. Let $\mathcal{D}$ be a concrete domain. The set of $\mathcal{ALCF}(\mathcal{D})$-concepts is the smallest set such that

1. every concept name is a concept

[1] A region r is regular closed if it satisfies $ICr = r$, where C is the topological closure operator and I is the topological interior operator.

2. if C and D are concepts, R is a role name, g is a concrete feature, p_1 and p_2 are abstract paths, $u_1, \ldots, u_n$ are concrete paths, and $P \in \Phi_\mathcal{D}$ is a predicate of arity n, then the following expressions are also concepts:

$$\neg C,\ C \sqcap D,\ C \sqcup D,\ \exists R.C,\ \forall R.C,\ p_1{\uparrow}p_2,\ p_1{\downarrow}p_2,\ \exists u_1, \ldots, u_n.P, \text{ and } g{\uparrow}.$$

We use $\top$ to abbreviate $A \sqcup \neg A$, where A is an arbitrary concept name, and $\bot$ to abbreviate $\neg\top$. Moreover, we write $\forall p.C$ for $\forall f_1.\cdots\forall f_k.C$ if $p = f_1 \cdots f_k$ and $u{\uparrow}$ for $\forall f_1.\cdots\forall f_k.g{\uparrow}$ if $u = f_1 \cdots f_k g$. An $\mathcal{ALCF}(\mathcal{D})$-concept that does not contain subconcepts $p_1{\uparrow}p_2$ and $p_1{\downarrow}p_2$ is called $\mathcal{ALC}(\mathcal{D})$-concept. An $\mathcal{ALC}(\mathcal{D})$-concept that does not use any abstract or concrete features is called $\mathcal{ALC}$-concept.

Throughout this paper, we use the letter A to denote concept names, C, D, and E to denote (possibly complex) concepts, R to denote role names, f to denote abstract features, g to denote concrete features, p to denote abstract paths, u to denote concrete paths, and P to denote predicate names from the concrete domain.

The Description Logic $\mathcal{ALCF}(\mathcal{D})$ is equipped with a Tarski-style set-theoretic semantics that incorporates the concrete domain $\mathcal{D}$.

DEFINITION 2.3 ($\mathcal{ALCF}(\mathcal{D})$ semantics)
An *interpretation* $\mathcal{I}$ is a pair $(\Delta_\mathcal{I}, \cdot^\mathcal{I})$, where $\Delta_\mathcal{I}$ is a set called the *domain* and $\cdot^\mathcal{I}$ the *interpretation function*. 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 $\Delta_\mathcal{I} \times \Delta_\mathcal{I}$,
- each abstract feature f to a partial function $f^\mathcal{I}$ from $\Delta_\mathcal{I}$ to $\Delta_\mathcal{I}$, and
- each concrete feature g to a partial function $g^\mathcal{I}$ from $\Delta_\mathcal{I}$ to $\Delta_\mathcal{D}$.

If $u = f_1 \cdots f_n g$ is a concrete path, then $u^\mathcal{I}(d)$ is defined as $g^\mathcal{I}(f_n^\mathcal{I} \cdots (f_1^\mathcal{I}(d)) \cdots)$, and similarly for abstract paths. The interpretation function is extended to arbitrary concepts as follows:

$$\begin{aligned}
(\neg C)^\mathcal{I} &:= \Delta_\mathcal{I} \setminus C^\mathcal{I} \\
(C \sqcap D)^\mathcal{I} &:= C^\mathcal{I} \cap D^\mathcal{I} \\
(C \sqcup D)^\mathcal{I} &:= C^\mathcal{I} \cup D^\mathcal{I} \\
(\exists R.C)^\mathcal{I} &:= \{d \in \Delta_\mathcal{I} \mid \{e \mid (d,e) \in R^\mathcal{I}\} \cap C^\mathcal{I} \neq \emptyset\} \\
(\forall R.C)^\mathcal{I} &:= \{d \in \Delta_\mathcal{I} \mid \{e \mid (d,e) \in R^\mathcal{I}\} \subseteq C^\mathcal{I}\} \\
(p_1{\uparrow}p_2)^\mathcal{I} &:= \{d \in \Delta_\mathcal{I} \mid \exists e_1, e_2 \in \Delta_\mathcal{I} : p_1^\mathcal{I}(d) = e_1,\ p_2^\mathcal{I}(d) = e_2, \text{ and } e_1 \neq e_2\} \\
(p_1{\downarrow}p_2)^\mathcal{I} &:= \{d \in \Delta_\mathcal{I} \mid \exists e \in \Delta_\mathcal{I} : p_1^\mathcal{I}(d) = p_2^\mathcal{I}(d) = e\} \\
(\exists u_1, \ldots, u_n.P)^\mathcal{I} &:= \{d \in \Delta_\mathcal{I} \mid \exists x_1, \ldots, x_n \in \Delta_\mathcal{D} : u_i^\mathcal{I}(d) = x_i \text{ for } 1 \leq i \leq n \\
&\qquad\qquad \text{and } (x_1, \ldots, x_n) \in P^\mathcal{D}\} \\
(g{\uparrow})^\mathcal{I} &:= \{d \in \Delta_\mathcal{I} \mid g^\mathcal{I}(d) \text{ undefined}\}
\end{aligned}$$

An interpretation $\mathcal{I}$ is a *model* of a concept C iff $C^\mathcal{I} \neq \emptyset$. A concept C is *satisfiable* iff it has a model. C is *subsumed by* a concept D (written $C \sqsubseteq D$) iff $C^\mathcal{I} \subseteq D^\mathcal{I}$ for all interpretations $\mathcal{I}$.

It is well-known that, in Description Logics providing for full negation such as $\mathcal{ALCF}(\mathcal{D})$, subsumption can be reduced to (un)satisfiability and vice versa: $C \sqsubseteq D$ iff $C \sqcap \neg D$ is unsatisfiable and C is satisfiable iff $C \not\sqsubseteq \bot$. This allows us to concentrate on concept satisfiability in the remainder of this paper.

Note that feature (dis)agreements $p_1 \uparrow p_2$ and $p_1 \downarrow p_2$ take abstract paths as arguments and are thus not concerned with elements from the concrete domain. However, if the concrete domain provides for equality and inequality predicates (as both A and S do), it is obvious that we can express (dis)agreement of concrete paths using the concrete domain constructor. Also note that $a \in (p_1 \uparrow p_2)^{\mathcal{I}}$ implies that $p_1^{\mathcal{I}}(a)$ and $p_1^{\mathcal{I}}(a)$ are defined. Thus, $p_1 \uparrow p_2$ is *not* the negation of $p_1 \downarrow p_2$ (also see Section 3.2 and Figure 3).

We should like to comment on a minor difference between our variant of $\mathcal{ALCF}(\mathcal{D})$ and the original version of $\mathcal{ALC}(\mathcal{D})$ as defined by Baader and Hanschke [4]: instead of separating concrete and abstract features, Baader and Hanschke define only one type of feature which is interpreted as a partial function from $\Delta_{\mathcal{I}}$ to $\Delta_{\mathcal{I}} \cup \Delta_{\mathcal{D}}$. We prefer the "typed" approach since, in our opinion, it improves the readability of concepts. Moreover, it is not hard to see that the combined features can be "simulated" using pairs of concrete and abstract features.

3 Concept Satisfiability

In the following, we devise a tableau algorithm for deciding satisfiability of $\mathcal{ALCF}(\mathcal{D})$-concepts that needs at most polynomial space if $\mathcal{D}$ is admissible and $\mathcal{D}$-satisfiability is in PSpace. The algorithm also yields tight complexity bounds if $\mathcal{D}$-satisfiability is NExpTime-complete or ExpSpace-complete.

3.1 Overview

Since there exist rather different variants of tableau algorithms in Modal Logic and First Order Logic, we call the family of tableau algorithms commonly used for Description Logics *completion algorithms*. The reader is referred to [9] for an overview over such algorithms. Completion algorithms are characterized by an underlying data structure, a set of *completion rules* operating on this data structure, and a (possibly trivial) strategy for applying the rules. In principle, a completion algorithm starts with an initial data structure induced by the concept D whose satisfiability is to be decided and repeatedly applies completion rules according to the strategy. Repeated rule application can be thought of as making implicit knowledge explicit or as constructing a canonical model for the input concept (represented in terms of the underlying data structure). The algorithm stops if it encounters a contradiction or if no more completion rules are applicable. It returns satisfiable iff the latter is the case *and* no obvious contradiction was found, i.e., if the algorithm succeeds in constructing a (witness for a) model of the input concept. Otherwise, it returns unsatisfiable.

If a PSpace upper bound is to be proved using a completion algorithm, some additional efforts have to be made. To simplify discussion, let us consider the logic $\mathcal{ALC}$ for the moment [39]. A naive completion algorithm for $\mathcal{ALC}$ does not yield a PSpace upper bound since there exist satisfiable $\mathcal{ALC}$-concepts all of whose models are of size exponential in the concept length [19, 39]. Thus, an algorithm keeping

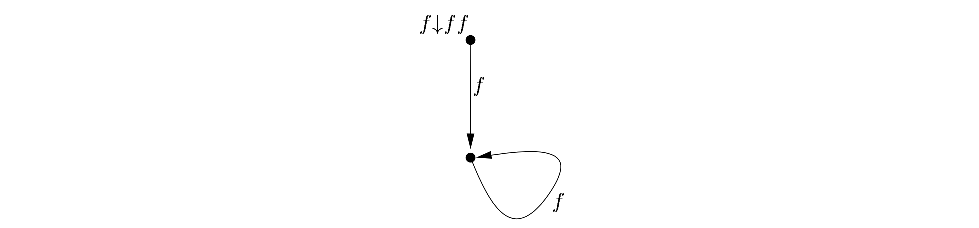

FIG. 2. A model of the $\mathcal{ALCF}(\mathcal{D})$-concept $f{\downarrow}ff$.

the entire (representation of a) model in memory needs exponential space in the worst case. However, there exists a well-known way to overcome this problem: the key observation is that canonical models $\mathcal{I}$ constructed by completion algorithms are *tree models*, i.e., they have the form of a tree if viewed as a graph with $\Delta_{\mathcal{I}}$ the set of vertexes and $\bigcup_{R \in \mathsf{N_R}} R^{\mathcal{I}}$ the set of edges. It is sufficient to consider only such tree models since $\mathcal{ALC}$ has the *tree model property*, which means that each satisfiable concept has a tree model [19]. To check for the existence of tree models for a given concept, we may try to construct one by performing depth-first search over role successors keeping only paths of the tree model in memory. Since, in the case of $\mathcal{ALC}$, the length of paths is at most polynomial in the length of the input concept [19], this technique—which is known as *tracing* [39]—yields an algorithm that needs at most polynomial space in the worst case. Completion algorithms for $\mathcal{ALC}$-concept satisfiability that use tracing are very similar to the well-known K-world algorithm from Modal Logic [26].

The tracing technique has to be modified to deal with $\mathcal{ALCF}(\mathcal{D})$-concepts for two reasons:

(1) Due to the presence of feature (dis)agreements, $\mathcal{ALCF}(\mathcal{D})$ does not enjoy the tree model property. For example, the concept $f{\downarrow}ff$ is satisfiable but, due to the functionality of the abstract feature f, has only non-tree models such as the one depicted in Figure 2.

(2) Due to the presence of the concrete domain constructor, even in tree models the paths of the tree cannot be considered in isolation. For example, the canonical tree model for the concept $\exists(f_1 f_2 g),(f_1' f_2' g').P$ is comprised of two paths with edge labels f_1, f_2, g and f_1', f_2', g', respectively. However, since the final node of the first path and the final node of the second path are elements of the concrete domain that must be related via the predicate P, we have to consider both paths together.

Since only abstract features (but no role names from $\mathsf{N_R} \setminus \mathsf{N_{aF}}$) are admitted in feature (dis)agreements and the concrete domain constructor, it is not hard to see that the described problems are due to substructures of models whose elements are connected by abstract features, only. Based on this observation, we define *generalized tree models*.

DEFINITION 3.1 (Generalized Tree Model)
Let $\mathcal{I}$ be a model of an $\mathcal{ALCF}(\mathcal{D})$-concept C and define a relation $\sim$ on $\Delta_\mathcal{I}$ as follows:

$d \sim e$ iff $d = e$ or there exists an abstract path $f_1 \cdots f_k$ and domain elements $d_0, \ldots, d_k \in \Delta_\mathcal{I}$ such that $d_0 = d$, $d_k = e$, and $d_{i+1} = f_{i+1}^\mathcal{I}(d_i)$ or $d_i = f_{i+1}^\mathcal{I}(d_{i+1})$ for $i < k$.

It is easy to see that $\sim$ is an equivalence relation. By $[d]_\sim$, we denote the equivalence class of $d \in \Delta_\mathcal{I}$ w.r.t. $\sim$. The model $\mathcal{I}$ is a *generalized tree model* of C iff $\mathcal{I}$ is a model of C and the graph $(V_\mathcal{I}, E_\mathcal{I})$ defined as

$$\begin{array}{rcl} V_\mathcal{I} & := & \{[d]_\sim \mid d \in \Delta_\mathcal{I}\} \\ E_\mathcal{I} & := & \{([d]_\sim, [e]_\sim) \mid \exists d' \in [d]_\sim, e' \in [e]_\sim \text{ such that} \\ & & \qquad (d', e') \in R^\mathcal{I} \text{ for some } R \in \mathsf{N_R} \setminus \mathsf{N_{aF}}\} \end{array}$$

is a tree.

It will be a byproduct of the results obtained in this section that $\mathcal{ALCF}(\mathcal{D})$ has the *generalized tree model property*, i.e., that every satisfiable $\mathcal{ALCF}(\mathcal{D})$-concept C has a generalized tree model. Note that the identification of some kind of tree model property is usually very helpful for devising decision procedures [42, 15]. Our completion algorithm for $\mathcal{ALCF}(\mathcal{D})$ uses tracing on generalized tree models: it keeps only fragments of models $\mathcal{I}$ in memory that induce paths in the abstraction $(V_\mathcal{I}, E_\mathcal{I})$. Intuitively, such a fragment consists of a sequence of "clusters" of domain elements, where each cluster is an equivalence class w.r.t. the relation $\sim$, i.e., a set of elements connected by abstract features. Succeeding clusters in the sequence are connected by roles from $\mathsf{N_R} \setminus \mathsf{N_{aF}}$. Fortunately, as we shall see later, there always exists a generalized tree model $\mathcal{I}$ in which the cardinality of clusters and the depth of the tree $(V_\mathcal{I}, E_\mathcal{I})$ is at most polynomial in the length of the input concept. We use these facts to devise a completion algorithm for $\mathcal{ALCF}(\mathcal{D})$-concept satisfiability running in polynomial space.

The polynomial size of object clusters is also exploited for dealing with the concrete domain. Along with constructing the "logical part" of the model for the input concept, our completion algorithm will build up a predicate conjunction describing its "concrete part". This predicate conjunction is required to be satisfiable in order for the constructed data structure to represent a model (see the general description of completion algorithms above). However, if this is done in a straightforward way, the number of conjuncts in the predicate conjunction may become exponential in the length of the input concept—see e.g. the algorithm for $\mathcal{ALC}(\mathcal{D})$ concept satisfiability presented in [4]. In our algorithm, we address this problem as follows: domain elements that are in different clusters of the generalized tree model are not connected through abstract paths. Therefore, it cannot be enforced that concrete successors of domain elements from different clusters are related by a concrete predicate. This, in turn, means that it is sufficient to *separately* check the satisfiability of predicate conjunctions associated with clusters. Since the size of predicate conjunctions associated with a cluster is at most polynomial in the length of the input concept, this separate checking allows to devise a PSPACE algorithm (if $\mathcal{D}$-satisfiability is in PSPACE).

$$
\begin{array}{rclrcl}
\neg(C \sqcap D) & \rightsquigarrow & \neg C \sqcup \neg D & \neg(C \sqcup D) & \rightsquigarrow & \neg C \sqcap \neg D \\
\neg(\exists R.C) & \rightsquigarrow & \forall R.\neg C & \neg(\forall R.C) & \rightsquigarrow & \exists R.\neg C \\
\neg(p_1{\uparrow}p_2) & \rightsquigarrow & p_1{\downarrow}p_2 \sqcup \forall p_1.\bot \sqcup \forall p_2.\bot & \neg(p_1{\downarrow}p_2) & \rightsquigarrow & p_1{\uparrow}p_2 \sqcup \forall p_1.\bot \sqcup \forall p_2.\bot
\end{array}
$$

$$
\begin{array}{rcl}
\neg\neg C & \rightsquigarrow & C \\
\neg(\exists u_1,\ldots,u_n.P) & \rightsquigarrow & \exists u_1,\ldots,u_n.\overline{P} \sqcup u_1{\uparrow} \sqcup \cdots \sqcup u_n{\uparrow} \\
\neg(g{\uparrow}) & \rightsquigarrow & \exists g.\top_{\mathcal{D}}
\end{array}
$$

FIG. 3. The NNF rewrite rules.

3.2 The Completion Algorithm

In the following, we assume that concepts are in negation normal form (NNF), i.e., that negation occurs only in front of concept names. Every $\mathcal{ALCF}(\mathcal{D})$-concept C can be transformed into an equivalent one in NNF by exhaustively applying the rewrite rules displayed in Figure 3 (recall that $\overline{P}$ denotes the negation of the predicate P). Let us start the presentation of the completion algorithm by introducing ABoxes as the underlying data structure.

DEFINITION 3.2 (ABox Syntax)
Let $\mathsf{O_a}$ and $\mathsf{O_c}$ be countably infinite and mutually disjoint sets of *abstract objects* and *concrete objects*. If C is an $\mathcal{ALCF}(\mathcal{D})$-concept, $R \in \mathsf{N_R}$ a role name, g a concrete feature, $a, b \in \mathsf{O_a}$, $x, x_1, \ldots, x_n \in \mathsf{O_c}$, and $P \in \Phi_{\mathcal{D}}$ with arity n, then

$$a : C, \quad (a,b) : R, \quad (a,x) : g, \quad (x_1,\ldots,x_n) : P, \quad \text{and} \quad a \not\approx b$$

are *ABox assertions*. An *ABox* is a finite set of such assertions.

Let $\mathcal{A}$ be an ABox, $a, b \in \mathsf{O_a}$ and $x \in \mathsf{O_c}$. We write $\mathcal{A}(a)$ to denote the set of concepts $\{C \mid a : C \in \mathcal{A}\}$. The abstract object b is called *R-successor of a in $\mathcal{A}$* iff $(a,b) : R$ is in $\mathcal{A}$. The notions *g-successor* (for concrete features g), *p-successor* (for abstract paths p), and *u-successor* (for concrete paths u) are defined analogously. In what follows, we used a and b to denote abstract objects and x to denote concrete objects.

For proving the soundness and completeness of the completion algorithm to be devised, it is convenient to equip ABoxes with a semantics:

DEFINITION 3.3 (ABox Semantics)
In interpretations $\mathcal{I}$, the interpretation function $\cdot^{\mathcal{I}}$ maps, additionally, abstract objects a to elements $a^{\mathcal{I}} \in \Delta_{\mathcal{I}}$ and concrete objects x to elements $x^{\mathcal{I}} \in \Delta_{\mathcal{D}}$. An interpretation $\mathcal{I}$ satisfies an assertion

$$
\begin{array}{rcl}
a : C & \text{iff} & a^{\mathcal{I}} \in C^{\mathcal{I}}; \\
(a,b) : R & \text{iff} & (a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}; \\
(a,x) : g & \text{iff} & g^{\mathcal{I}}(a^{\mathcal{I}}) = x^{\mathcal{I}}; \\
(x_1,\ldots,x_n) : P & \text{iff} & (x_1^{\mathcal{I}},\ldots,x_n^{\mathcal{I}}) \in P^{\mathcal{D}}; \\
a \not\approx b & \text{iff} & a^{\mathcal{I}} \neq b^{\mathcal{I}}.
\end{array}
$$

An interpretation $\mathcal{I}$ is called a *model* of an ABox $\mathcal{A}$ iff it satisfies every assertion in $\mathcal{A}$. An ABox is called *consistent* iff it has a model.

It should be obvious how ABoxes can be used to represent models. If the satisfiability of a concept D is to be decided, the completion algorithm is started with the *initial ABox* for D defined as $\mathcal{A}_D = \{a : D\}$. To keep the presentation of the completion rules succinct, we introduce an operation that allows to introduce new objects on paths and concrete paths.

DEFINITION 3.4 ("+" operation)
An abstract or concrete object is called *fresh* w.r.t. an ABox $\mathcal{A}$ if it does not appear in $\mathcal{A}$. Let $p = f_1 \cdots f_n$ be an abstract path (resp. $u = f_1 \cdots f_n g$ be a concrete path). By $\mathcal{A} + apb$ (resp. $\mathcal{A} + aux$), where $a \in \mathsf{O_a}$ is used in $\mathcal{A}$ and $b \in \mathsf{O_a}$ (resp. $x \in \mathsf{O_c}$), we denote the ABox $\mathcal{A}'$ which can be obtained from $\mathcal{A}$ by choosing distinct objects $b_1, \ldots, b_n \in \mathsf{O_a}$ which are fresh in $\mathcal{A}$ and setting

$$\begin{aligned} \mathcal{A}' \quad &:= \quad \mathcal{A} \cup \{(a, b_1) : f_1, \; \ldots, \; (b_{n-1}, b) : f_n\} \\ (\text{resp. } \mathcal{A}' \quad &:= \quad \mathcal{A} \cup \{(a, b_1) : f_1, \; \ldots, \; (b_{n-1}, b_n) : f_n, \; (b_n, x) : g\}. \end{aligned}$$

When nesting the + operation, we omit brackets writing, e.g., $\mathcal{A} + ap_1b + bp_2c$ for $(\mathcal{A} + ap_1b) + bp_2c$.

The completion rules can be found in Figure 4. Note that the $\mathsf{R}\sqcup$ rule is nondeterministic, i.e., it has more than one possible outcome. Thus, the described completion algorithm is a nondeterministic decision procedure. Such an algorithm accepts its input (i.e. returns satisfiable) iff there is *some* way to make the nondeterministic decisions such that a positive result is obtained. A convenient way to think of nondeterministic rules is that they "guess" the correct outcome, i.e., if there is an outcome which, if chosen, leads to a positive result, then this outcome is in fact considered.

Most completion rules are standard and known from, e.g., [5] and [22]. The $\mathsf{R}\exists\mathsf{f}$ and $\mathsf{R}\forall\mathsf{f}$ rules are special in that they only deal with concepts $\exists f.C$ and $\forall f.C$ where f is an abstract feature. As we will see later, concepts $\exists R.C$ and $\forall R.C$ with $R \in \mathsf{N_R} \setminus \mathsf{N_{aF}}$ are not treated by completion rules but through recursion calls of the algorithm. The Rfe rule also deserves some attention: it ensures that, for any object $a \in \mathsf{O_a}$, there exists at most a single f-successor for each $f \in \mathsf{N_{aF}}$ and at most a single g-successor for each $g \in \mathsf{N_{cF}}$. Redundant successors are eliminated by identification. This process is often referred to as *fork elimination* (hence the name of the rule). In many cases, fork elimination is not explicitly formulated as a completion rule but viewed as an integral part of the other completion rules. In the presence of feature (dis)agreements, this latter approach seems to be less transparent. Consider for example the ABox

$$\{a : \exists f_1.\top, \; a : \exists f_2.\top, \; a : f_1{\downarrow}f_2\}.$$

Assume the $\mathsf{R}\exists\mathsf{f}$ rule is applied twice adding the assertions $(a, b) : f_1$ and $(a, c) : f_2$. Now, the $\mathsf{R}{\downarrow}$ rule is applied adding $(a, b') : f_1$ and $(a, b') : f_2$. Clearly, we may now apply the Rfe rule to the assertions $(a, b) : f_1$ and $(a, b') : f_1$. Say the rule application replaces b' by b, and we obtain the ABox

$$\{a : \exists f_1.\top, \; a : \exists f_2.\top, \; a : f_1{\downarrow}f_2, \; (a, b) : f_1, \; (a, c) : f_2, \; (a, b) : f_2\}.$$

Obviously, we may now apply Rfe to $(a, c) : f_2$ and $(a, b) : f_2$ replacing b by c. Observe that this latter fork elimination does not involve any objects generated by

R$\sqcap$	if $C_1 \sqcap C_2 \in \mathcal{A}(a)$ and $\{C_1, C_2\} \not\subseteq \mathcal{A}(a)$ then $\mathcal{A} := \mathcal{A} \cup \{a : C_1, a : C_2\}$
R$\sqcup$	if $C_1 \sqcup C_2 \in \mathcal{A}(a)$ and $\{C_1, C_2\} \cap \mathcal{A}(a) = \emptyset$ then $\mathcal{A} := \mathcal{A} \cup \{a : C\}$ for some $C \in \{C_1, C_2\}$
R$\exists$f	if $\exists f.C \in \mathcal{A}(a)$ and there is no f-successor b of a with $C \in \mathcal{A}(b)$ then set $\mathcal{A} := \mathcal{A} \cup \{(a, b) : f,\ b : C\}$ for a $b \in \mathsf{O_a}$ fresh in $\mathcal{A}$
R$\forall$f	if $\forall f.C \in \mathcal{A}(a)$, b is an f-successor of a, and $C \notin \mathcal{A}(b)$ then set $\mathcal{A} := \mathcal{A} \cup \{b : C\}$
Rc	if $\exists u_1, \ldots, u_n.P \in \mathcal{A}(a)$ and there exist no $x_1, \ldots, x_n \in \mathsf{O_c}$ such that x_i is u_i-successor of a for $1 \leq i \leq n$ and $(x_1, \ldots, x_n) : P \in \mathcal{A}$ then set $\mathcal{A} := (\mathcal{A} + au_1x_1 + \cdots + au_nx_n) \cup \{(x_1, \ldots, x_n) : P\}$ with $x_1, \ldots, x_n \in \mathsf{O_c}$ fresh in $\mathcal{A}$
R$\downarrow$	if $p_1{\downarrow}p_2 \in \mathcal{A}(a)$ and there is no b that is both a p_1-successor of a and a p_2-successor of a then set $\mathcal{A} := \mathcal{A} + ap_1b + ap_2b$ for a $b \in \mathsf{O_a}$ fresh in $\mathcal{A}$
R$\uparrow$	if $p_1{\uparrow}p_2 \in \mathcal{A}(a)$ and there are no b_1, b_2 with b_1 p_1-successor of a, b_2 p_2-successor of a, and $(b_1 \not\approx b_2) \in \mathcal{A}$ then set $\mathcal{A} := (\mathcal{A} + ap_1b_1 + ap_2b_2) \cup \{(b_1 \not\approx b_2)\}$ for $b_1, b_2 \in \mathsf{O_a}$ fresh in $\mathcal{A}$
Rfe	if $\{(a, b) : f, (a, c) : f\} \subseteq \mathcal{A}$ and $b \neq c$ (resp. $\{(a, x) : g, (a, y) : g\} \subseteq \mathcal{A}$ and $x \neq y$) then replace b by c in $\mathcal{A}$ (resp. x by y)

Fig. 4. Completion rules for $\mathcal{ALCF}(\mathcal{D})$.

the last "non-Rfe" rule application. To make such effects more transparent, we chose to formulate fork elimination as a separate rule.

Let us now formalize what it means for an ABox to be contradictory.

Definition 3.5 (Clash)
With each ABox $\mathcal{A}$, we associate a predicate conjunction

$$\zeta_{\mathcal{A}} = \bigwedge_{(x_1, \ldots, x_n) : P \in \mathcal{A}} P(x_1, \ldots, x_n).$$

The ABox $\mathcal{A}$ is called *concrete domain satisfiable* iff $\zeta_{\mathcal{A}}$ is satisfiable. It is said to contain a *clash* iff one of the following conditions applies:

1. $\{A, \neg A\} \subseteq \mathcal{A}(a)$ for a concept name A and object $a \in \mathsf{O_a}$,
2. $(a \not\approx a) \in \mathcal{A}$ for some object $a \in \mathsf{O_a}$,
3. $g{\uparrow} \in \mathcal{A}(a)$ for some $a \in \mathsf{O_a}$ such that there exists a g-successor of a, or
4. $\mathcal{A}$ is not concrete domain satisfiable.

If $\mathcal{A}$ does not contain a clash, then $\mathcal{A}$ is called *clash-free.*

define procedure $\mathsf{sat}(\mathcal{A})$
  $\mathcal{A} := \mathsf{fcompl}(\mathcal{A})$
  if $\mathcal{A}$ contains a clash **then**
    return unsatisfiable
  forall assertions $\exists R.C \in \mathcal{A}(a)$ with $R \in \mathsf{N_R} \setminus \mathsf{N_{aF}}$ **do**
    Fix $b \in \mathsf{O_a}$
    if $\mathsf{sat}(\{b : C\} \cup \{b : E \mid \forall R.E \in \mathcal{A}(a)\}) =$ unsatisfiable **then**
      return unsatisfiable
  return satisfiable

define procedure $\mathsf{fcompl}(\mathcal{A})$
  while a rule from Figure 4 is applicable to $\mathcal{A}$ **do**
    Choose an applicable rule R s.t. $\mathsf{R} = \mathsf{Rfe}$ if Rfe is applicable
    Apply R to $\mathcal{A}$
  return $\mathcal{A}$

FIG. 5. The $\mathcal{ALCF}(\mathcal{D})$-concept satisfiability algorithm.

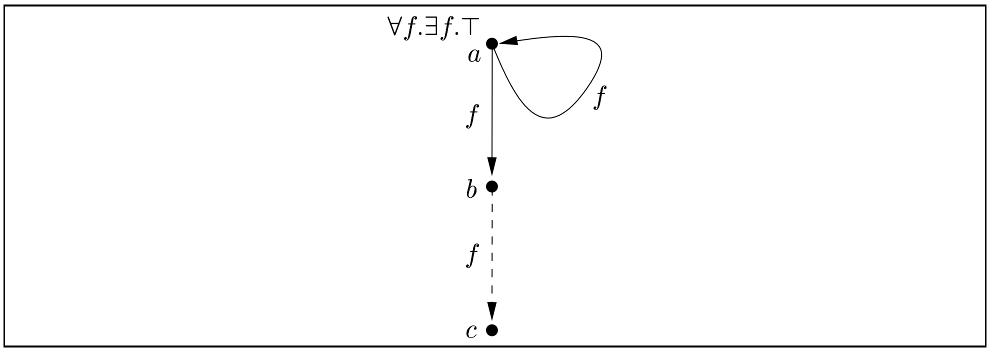

FIG. 6. The "yo-yo" effect.

The completion algorithm itself can be found in Figure 5. We briefly summarize the strategy followed by the algorithm. The argument to sat is an ABox containing exactly one object $a \in \mathsf{O_a}$ and only assertions of the form $a : C$. The algorithm uses the fcompl function to create all feature successors of a, all feature successors of these feature successors and so on. However, fcompl does not generate any R-successors for role names $R \in \mathsf{N_R} \setminus \mathsf{N_{aF}}$. In other words, fcompl generates a cluster of objects as described in Section 3.1. After the call to the fcompl function, the algorithm makes a recursion call for each role successor enforced via an $\exists R.C$ assertion (with $R \in \mathsf{N_R} \setminus \mathsf{N_{aF}}$). A single such recursion call corresponds to moving along a path in a generalized tree model, i.e, to moving to a successor cluster of the cluster under consideration. Each cluster of objects is checked separately for contradictions. Note that, due to Definition 3.5, checking for a clash involves checking whether the predicate conjunction $\zeta_{\mathcal{A}}$ is satisfiable. This, in turn, is a decidable problem since we assume $\mathcal{D}$ to be admissible.

R∃r	if $\exists R.C \in \mathcal{A}(a)$ with $R \in \mathsf{N_R} \setminus \mathsf{N_{aF}}$ and there is no R-successor b of a with $C \in \mathcal{A}(b)$ then set $\mathcal{A} := \mathcal{A} \cup \{(a,b) : R\ b : C\}$ for a $b \in \mathsf{O_a}$ fresh in $\mathcal{A}$
R∀r	if $\forall R.C \in \mathcal{A}(a)$ with $R \in \mathsf{N_R} \setminus \mathsf{N_{aF}}$, b is a R-successor of a, and $C \notin \mathcal{A}(b)$ then set $\mathcal{A} := \mathcal{A} \cup \{b : C\}$

FIG. 7. Virtual completion rules for $\mathcal{ALCF}(\mathcal{D})$.

Observe that fcompl applies the Rfe rule with highest priority. Without this strategy, the algorithm would not terminate: consider the ABox

$$\mathcal{A} = \{a : \forall f.\exists f.\top,\ (a,a) : f,\ (a,b) : f\}.$$

This ABox, which is depicted in the upper part of Figure 6, is encountered if, for example, the algorithm is started on the input concept $f'{\downarrow}f'f \sqcap \exists f'.(\forall f.\exists f.\top \sqcap \exists f.\top)$. Now assume that the completion rules are applied to $\mathcal{A}$ without giving Rfe the highest priority. This means that we can apply the R∀f rule and obtain $b : \exists f.\top$. We can then apply R∃f generating $(b,c) : f,\ c : \top$. Fork elimination may now identify a and b and thus we are back at the initial situation (up to renaming). Clearly, this sequence of rule applications may be repeated indefinitely—the algorithm does not terminate. This "yo-yo" effect was also described, e.g., in [9].

3.3 Correctness and Complexity

In this section, we prove that the completion algorithm is sound, complete, and terminating and can be executed using only polynomial space provided that $\mathcal{D}$-satisfiability is in PSPACE. With D, we denote the input concept to the completion algorithm whose satisfiability is to be decided.

We first prove termination of the algorithm. It is convenient to start with establishing an upper bound for the number of rule applications performed by the fcompl function and, closely related, an upper bound for the size of ABoxes generated by the fcompl function. Before we do this, let us introduce the two additional completion rules displayed in Figure 7, which will play an important role in the termination and correctness proofs. These rules are not applied explicitly by the algorithm, but rather can the recursion calls of the sat function be viewed as a single application of the R∃r rule together with multiple applications of the R∀r rule. Let us now return to the upper bounds for the fcompl function. With foresight to the ABox consistency algorithm to be devised in the next section, we consider the precompl function instead of the fcompl function, where precompl is defined exactly as fcompl except that it also applies the R∀r rule. A formal definition of the precompl function can be found in Figure 9. It is not hard to see that upper bounds for the number of rule applications performed by precompl or the size of ABoxes generated by precompl also apply to the fcompl function: if the fcompl functions perform a computation on an input ABox $\mathcal{A}$, then precompl can perform precisely the same computation on the input ABox $\mathcal{A}'$ obtained from $\mathcal{A}$ by replacing all subconcept $\forall R.C$ appearing in $\mathcal{A}$ with concept names.

In what follows, we use $\mathsf{sub}(C)$ to denote the set of subconcepts of the concept C and $\mathsf{sub}(\mathcal{A})$ to denote the union of the sets of subconcepts of all those concepts C that appear in assertions $a : C$ in the ABox $\mathcal{A}$. Moreover, we use $|C|$ to denote the *length* of a concept C, i.e., the number of symbols used to write it down. The *size* $|\alpha|$ of an ABox assertion α is defined as $|C|$ if $\alpha = a : C$ and 1 otherwise. The size $|\mathcal{A}|$ of an ABox $\mathcal{A}$ is defined as the sum of the sizes of its assertions.

LEMMA 3.6
For any input $\mathcal{A}$, the function $\mathsf{precompl}$ terminates after at most $|\mathcal{A}|^4$ rule applications and constructs an ABox $\mathcal{A}'$ with $|\mathcal{A}'| \leq |\mathcal{A}|^6$.

PROOF. In the following, we call assertions of the form $a : C$ *concept assertions*, assertions of the form $(a, b) : f$ or $(a, x) : g$ *feature assertions*, and assertions of the form $(a, b) : R$ with $R \in \mathsf{N_R} \setminus \mathsf{N_{aF}}$ *role assertions.*

The main task is to show that

$$\mathsf{precompl} \text{ terminates after at most } |\mathcal{A}|^4 \text{ rule applications.} \qquad (*)$$

For suppose that $(*)$ has been shown. We can then prove the lemma by making the following two observations, which clearly imply that the size of the ABox $\mathcal{A}'$ generated by $\mathsf{precompl}$ is bounded by $|\mathcal{A}|^6$.

(i) We have $|\alpha| < |\mathcal{A}|$ for each new assertion α added by rule application: concept assertions are the only kind of assertions that may have a size greater than one and, if a concept assertion $a : C$ is added by rule application, then $C \in \mathsf{sub}(\mathcal{A})$;

(ii) Each rule application adds at most $|\mathcal{A}|$ new assertions: each application adds either no new assertions (the Rfe rule) or at most $|C|$ new assertions, where $a : C$ is the concept assertion appearing in the (instantiated) rule premise. In the latter case, we have $|C| \leq |\mathcal{A}|$ since C is in $\mathsf{sub}(\mathcal{A})$.

Hence, let us prove $(*)$. Let $\mathcal{A}_0, \mathcal{A}_1, \ldots$ be the sequence of ABoxes computed by $\mathsf{precompl}$. More precisely, $\mathcal{A}_0 = \mathcal{A}$ and $\mathcal{A}_{i+1}$ is obtained from $\mathcal{A}_i$ by the i-th rule application performed by $\mathsf{precompl}$.

We first introduce some notions. For $i \geq 0$ and $a \in \mathsf{O_a} \cup \mathsf{O_c}$, we use $\mathsf{nm}_i(a)$ to denote the set of names that a had "until $\mathcal{A}_i$". More precisely, $\mathsf{nm}_0(a) = \{a\}$ for all $a \in \mathsf{O_a} \cup \mathsf{O_c}$. If the Rfe rule is applied to an ABox $\mathcal{A}_i$ renaming an object a to b, then $\mathsf{nm}_{i+1}(b) = \mathsf{nm}_i(a) \cup \mathsf{nm}_i(b)$ and $\mathsf{nm}_{i+1}(c) = \mathsf{nm}_i(c)$ for all $c \neq b$. For all other rule applications, we simply have $\mathsf{nm}_{i+1}(a) = \mathsf{nm}_i(a)$ for all $a \in \mathsf{O_a} \cup \mathsf{O_c}$. The following properties, which we summarize under the notion *persistence*, are easily proved using the fact that assertions are never deleted:

- If $a : C \in \mathcal{A}_i$ and $a \in \mathsf{nm}_j(a')$ for some $j > i$ and $a' \in \mathsf{O_a}$, then $a' : C \in \mathcal{A}_j$.
- if $(a, b) : R \in \mathcal{A}_i$, $a \in \mathsf{nm}_j(a')$, and $b \in \mathsf{nm}_j(b')$ for some $j > i$ and $a', b' \in \mathsf{O_a}$, then $(a', b') : R \in \mathcal{A}_j$.
- If $(a, x) : g \in \mathcal{A}_i$, $a \in \mathsf{nm}_j(a')$, and $x' \in \mathsf{nm}_j(x)$ for some $j > i$, $a' \in \mathsf{O_a}$, and $x' \in \mathsf{O_c}$, then $(a', x') : g \in \mathcal{A}_j$.
- If $(x_1, \ldots, x_n) : P \in \mathcal{A}_i$ and $x'_i \in \mathsf{nm}_j(x_i)$ for $1 \leq i \leq n$, then $(x'_1, \ldots, x'_n) : P \in \mathcal{A}_j$.

A concept assertion $a : C$ is called *touched* in $\mathcal{A}_i$ if there exists an $a' \in \mathsf{nm}_i(a)$ such that one of the first i rule applications involved $a' : C$ in the (instantiated) rule premise and *untouched* otherwise. By $\sharp_{\mathsf{feat}}(\mathcal{A})$, we denote the number of feature assertions in $\mathcal{A}$. For role assertions $(a,b):R$ with $R \in \mathsf{N_R} \setminus \mathsf{N_{aF}}$, we use $\lambda_{\mathcal{A}_i}(a, b : R)$ to denote the number of concepts $\forall R.C$ in $\mathsf{sub}(\mathcal{A})$ for which there exist no $a' \in \mathsf{nm}_i(a)$ and $b' \in \mathsf{nm}_i(b)$ such that one of the first i rule applications involved both $a' : \forall R.C$ and $(a', b') : R$ in the (instantiated) rule premise.

For $i \geq 0$, define

$$w(\mathcal{A}_i) := \sum_{a:C \text{ is untouched in } \mathcal{A}_i} |a:C| \quad + \quad \sharp_{\mathsf{feat}}(\mathcal{A}_i) \quad + \quad |\mathcal{A}| \cdot \sum_{(a,b):R \in \mathcal{A}_i} \lambda_{\mathcal{A}_i}(a, b : R).$$

We show that $w(\mathcal{A}_{i+1}) < w(\mathcal{A}_i)$ for $i \geq 0$, which implies that the length of the sequence $\mathcal{A}_0, \mathcal{A}_1, \ldots$ is bounded by $|\mathcal{A}|^4$ since it is readily checked that $w(\mathcal{A}_0) \leq |\mathcal{A}|^4$. A case distinction is made according to the completion rule applied.

- Assume that $\mathcal{A}_{i+1}$ is obtained from $\mathcal{A}_i$ by an application of the $\mathsf{R}\sqcap$ rule. By definition of this rule and due to persistence, it is applied to an untouched assertion $a : C_1 \sqcap C_2$ in $\mathcal{A}_i$: for suppose that $a : C_1 \sqcap C_2$ is touched in $\mathcal{A}_i$. By definition of "touched", this implies that there exists an $a' \in \mathsf{nm}_i(a)$ such that $\mathsf{R}\sqcap$ has been applied to $a' : C_1 \sqcap C_2$ in the j-th rule application for some $j < i$. By definition of $\mathsf{R}\sqcap$, this implies $\{a' : C_1, a' : C_2\} \subseteq \mathcal{A}_j$. By persistence, we have $\{a : C_1, a : C_2\} \subseteq \mathcal{A}_i$ and, thus, the $\mathsf{R}\sqcap$ rule is not applicable to $a : C_1 \sqcap C_2$ in $\mathcal{A}_i$ which is a contradiction. Hence, we have shown that $a : C_1 \sqcap C_2$ is untouched in $\mathcal{A}_i$. Moreover, this assertion is clearly touched in $\mathcal{A}_{i+1}$. The rule application generates new concept assertions $a : C_1$ and $a : C_2$ which may both be untouched in $\mathcal{A}_{i+1}$. Moreover, it generates no new feature and role assertions. By definition of the size of assertions and the length of concepts, we have $|a : C_1 \sqcap C_2| > |a : C_1| + |a : C_2|$. Thus $w(\mathcal{A}_{i+1}) < w(\mathcal{A}_i)$.
- The $\mathsf{R}\sqcup$ case is analogous to the previous case.
- Assume that $\mathcal{A}_{i+1}$ is obtained from $\mathcal{A}_i$ by an application of the $\mathsf{R}\forall\mathsf{f}$ rule. The rule is applied to assertions $a : \forall f.C$ and $(a,b) : f$. Suppose that $a : \forall f.C$ is touched in $\mathcal{A}_i$, i.e., that the $\mathsf{R}\forall\mathsf{f}$ rule has been applied in a previous step to an assertion $a' : \forall f.C$ with $a' \in \mathsf{nm}_i(a)$. It then added $c : C$ for an f-successor c of a'. The facts that (i) Rfe is applied with highest priority, (ii) b is an f-successor of a in $\mathcal{A}_{i+1}$, and (iii) the $\mathsf{R}\forall\mathsf{f}$ rule is applicable imply that we have $c \in \mathsf{nm}_i(b)$. This, in turn, implies $b : C \in \mathcal{A}_i$ by persistence and we have obtained a contradiction to the assumption that $\mathsf{R}\forall\mathsf{f}$ is applicable. Hence, we have shown that $a : \forall f.C$ is untouched in $\mathcal{A}_i$. The assertion is touched in $\mathcal{A}_{i+1}$. Rule application generates a new assertion $b : C$ that is untouched in $\mathcal{A}_{i+1}$. However, $|a : \forall f.C| > |b : C|$. No new feature or role assertions are generated.
- Assume that $\mathcal{A}_{i+1}$ is obtained from $\mathcal{A}_i$ by an application of the $\mathsf{R}\forall\mathsf{r}$ rule. The rule is applied to assertions $a : \forall R.C$ and $(a,b) : R$ in $\mathcal{A}_i$. Due to persistence, there do not exist $a' \in \mathsf{nm}_i(a)$ and $b' \in \mathsf{nm}_i(b)$ such that the $\mathsf{R}\forall\mathsf{r}$ rule has previously been applied to $a' : \forall R.C$ and $(a', b') : R$. Hence, $\lambda_{\mathcal{A}_{i+1}}(a, b : R) = \lambda_{\mathcal{A}_i}(a, b : R) - 1$ and the third summand of $w(\mathcal{A}_i)$ exceeds the third summand of $w(\mathcal{A}_{i+1})$ by $|\mathcal{A}|$. The rule application adds no feature or role assertions and a single concept assertion $b : C$. Since $\forall R.C \in \mathsf{sub}(\mathcal{A})$, we have $|b : C| < |\mathcal{A}|$ and hence $w(\mathcal{A}_{i+1}) < w(\mathcal{A}_i)$.
- Assume that $\mathcal{A}_{i+1}$ is obtained from $\mathcal{A}_i$ by an application of the $\mathsf{R}\exists\mathsf{f}$ rule. As in the $\mathsf{R}\sqcap$ case, it is easy to show that the rule is applied to an untouched assertion

$a : \exists f.C$. It generates new assertions $(a,b) : f$ and $b : C$ (and no new role assertions). The assertion $b : C$ is untouched in $\mathcal{A}_{i+1}$ and $a : \exists f.C$ is touched in $\mathcal{A}_{i+1}$. The new feature assertion $(a,b) : f$ yields $\sharp_{\mathsf{feat}}(\mathcal{A}_{i+1}) = \sharp_{\mathsf{feat}}(\mathcal{A}_i)+1$. On the other hand, no role assertion is added and we clearly have $|a : \exists f.C| > |b : C| + 1$.

- The Rc, $\mathsf{R}\!\downarrow$, and $\mathsf{R}\!\uparrow$ rules touch a (due to persistence) previously untouched concept assertion $a : C$ appearing in the instantiated premise and do not add new concept or role assertions. It is readily checked that the number of feature assertions added by rule application is smaller than $|a : C|$.
- Assume that the Rfe rule is applied to an ABox $\mathcal{A}_i$. This obviously implies $\sharp_{\mathsf{feat}}(\mathcal{A}_{i+1}) < \sharp_{\mathsf{feat}}(\mathcal{A}_i)$, i.e., the second summand of $w(\mathcal{A}_{i+1})$ is strictly smaller than the second summand of $w(\mathcal{A}_i)$. If the rule application renames a concrete object, these are the only changes and we are done. If an abstract object is renamed, some work is necessary to show that the first and third summand of $w(\mathcal{A}_{i+1})$ are not greater than the corresponding summands of $w(\mathcal{A}_i)$. Assume that $a \in \mathsf{O_a}$ is renamed to b. We then have $\mathsf{nm}_{i+1}(b) = \mathsf{nm}_i(a) \cup \mathsf{nm}_i(b)$.
  - First summand. Let us first consider concept assertions $c : C \in \mathcal{A}_{i+1} \cap \mathcal{A}_i$. Such an assertion is untouched in $\mathcal{A}_{i+1}$ only if it is untouched in $\mathcal{A}_i$ since (i) $\mathsf{nm}_{i+1}(c) = \mathsf{nm}_i(c)$ if $c \neq b$ and (ii) $\mathsf{nm}_i(b) \subseteq \mathsf{nm}_{i+1}(b)$ if $c = b$. Moreover, if there exists an assertion $b : C \in \mathcal{A}_{i+1} \setminus \mathcal{A}_i$ due to variable renaming, then $a : C \in \mathcal{A}_i \setminus \mathcal{A}_{i+1}$, and $b : C$ being untouched in $\mathcal{A}_{i+1}$ implies $a : C$ being untouched in $\mathcal{A}_i$ since $\mathsf{nm}_i(a) \subseteq \mathsf{nm}_{i+1}(b)$. Hence, the first summand does not increase.
  - Third summand. Let $(c,d) : R \in \mathcal{A}_{i+1} \cap \mathcal{A}_i$ (implying $c \neq a$ and $d \neq a$). We distinguish several subcases:
    1. $c \neq b$ and $d \neq b$. Then, clearly, $\lambda_i(c, d : R) = \lambda_{i+1}(c, d : R)$.
    2. $c = b$ and $d \neq b$. By definition of λ_i, $\mathsf{nm}_i(b) \subseteq \mathsf{nm}_{i+1}(b)$ implies $\lambda_i(b, d : R) \geq \lambda_{i+1}(b, d : R)$.
    3. $c \neq b$ and $d = b$. As previous case.
    4. $c = d = b$. As previous case.

    Now let $(c,d) : R \in \mathcal{A}_{i+1} \setminus \mathcal{A}_i$ (implying $c = b$ or $d = b$). We can distinguish the cases (i) $c = b$, $d \neq b$, (ii) $d = b$, $c \neq b$, and (iii) $c = d = b$. Since all cases are similar, we concentrate on (i). In this case, $(a,d) : R \in \mathcal{A}_i \setminus \mathcal{A}_{i+1}$. Moreover, $\mathsf{nm}_i(a) \subseteq \mathsf{nm}_{i+1}(b)$ implies $\lambda_{\mathcal{A}_{i+1}}(b, d : R) \leq \lambda_{\mathcal{A}_i}(a, d : R)$.
    Summing up, the third summand may only decrease but not increase.

∎

The role depth of concepts is defined inductively as follows, where $|p|$ denotes the length of the abstract path p and $|u|$ denotes the length of the concrete path u (including the trailing concrete feature):

- $\mathsf{rd}(A) = \mathsf{rd}(g\!\uparrow) = 0$;
- $\mathsf{rd}(\exists u_1, \ldots, u_n.P) = \max(|u_1|, \ldots, |u_n|)$;
- $\mathsf{rd}(p_1\!\downarrow\! p_2) = \mathsf{rd}(p_1\!\uparrow\! p_2) = \max(|p_1|, |p_2|)$;
- $\mathsf{rd}(\neg C) = \mathsf{rd}(C)$;
- $\mathsf{rd}(C \sqcap D) = \mathsf{rd}(C \sqcup D) = \max(\mathsf{rd}(C), \mathsf{rd}(D))$;
- $\mathsf{rd}(\exists R.C) = \mathsf{rd}(\forall R.C) = \mathsf{rd}(C) + 1$;

We now prove a technical lemma that, together with Lemma 3.6, immediately yields termination.

LEMMA 3.7
Assume that the completion algorithm was started with input D. Then

1. in each recursion call, the size $|\mathcal{A}|$ of the argument $\mathcal{A}$ passed to sat is bounded by $|D|^2$;
2. in each recursion step of sat, at most $p(|D|)$ recursion calls are made, where p is a polynomial; and
3. the recursion depth of sat is bounded by $|D|$.

PROOF. Let us first prove Point 1. ABoxes passed to sat contain assertions of the form $a : C$ for a single object a. Since only concepts from $\mathsf{sub}(D)$ are generated during rule application, the number of distinct assertions of this form is bounded by $|\mathsf{sub}(D)| \leq |D|$. Obviously, the size of each such assertion is also bounded by $|D|$ which yields an upper bound of $|D|^2$ for the size of arguments to sat.

For Point 2, note that in each recursion step, the number of recursion calls made is bounded by the number of assertions $a : \exists R.C$ in the ABox $\mathcal{A}$ obtained by application of fcompl. By Point 1, the size of argument ABoxes to sat is bounded by $|D|^2$. Hence, by Lemma 3.6, the size of $\mathcal{A}$ is bounded by $p(|D|)$ where p is a polynomial and the same bound applies to the number of recursion calls made in each recursion step.

We now turn to Point 3. As a consequence of (i) the fact that rule application performed by fcompl may not introduce concepts with a role depth greater than the role depth of concepts that have already been in the ABox and (ii) the way in which the argument ABoxes for recursion calls to sat are constructed, we have that the role depth of concepts in the argument ABoxes passed to sat strictly decreases with recursion depth. It follows that the role depth of D is an upper bound for the recursion depth, i.e., the recursion depth is bounded by $|D|$. ∎

PROPOSITION 3.8
The completion algorithm terminates on any input $\mathcal{A}_D$.

PROOF. Immediate consequence of Lemma 3.6 and Points 2 and 3 from Lemma 3.7. ∎

We now come to proving soundness and completeness of the completion algorithm. Recall that, intuitively, the completion algorithm traverses a generalized tree model in a depth-first manner without keeping the entire model in memory. For the proofs, it is convenient to make the model traversed by the algorithm explicit—or more precisely the ABox representing it. To do this, we define an extended version of the completion algorithm. This extended algorithm is identical to the original one but additionally constructs a sequence of ABoxes $\mathcal{A}^0_\cup, \mathcal{A}^1_\cup, \dots$ collecting all assertions that the algorithm generates. Hence, it returns $\mathsf{satisfiable}$ if and only if the original algorithm does. We will show that, if the extended algorithm is started on an initial ABox $\mathcal{A}_D$ and terminates after n steps returning $\mathsf{satisfiable}$, then the ABox $\mathcal{A}^n_\cup$ defines a canonical model for $\mathcal{A}_D$. Since the extended algorithm returns $\mathsf{satisfiable}$ if the original one does, this yields soundness. Completeness can also be shown using the correspondence between the two algorithms. Note that the extended version of the algorithm is defined just to prove soundness and completeness of the original version

```
*  Initialization:
*  rc := sc := 0
*  A⁰_∪ := {a_0 : D} if A_D = {a : D}

   define procedure sat(A)
      A := fcompl(A)
      if A contains a clash then
         return unsatisfiable
      forall assertions ∃R.C ∈ A(a) with R ∈ N_R \ N_aF do
*        sc := sc + 1
*        rc := rc + 1
         Fix b ∈ O_a
*        A^rc_∪ := A^(rc−1)_∪ ∪ {(a_(sc−1), b_sc) : R} ∪ {b_sc : C} ∪
*                                    {b_sc : E | a : ∀R.E ∈ A(a)}
         if sat({b : C} ∪ {b : E | ∀R.E ∈ A(a)}) = unsatisfiable then
            return unsatisfiable
      return satisfiable

   define procedure fcompl(A)
*     A_0 := A
      while a rule R from Figure 4 is applicable to A do
         Choose an applicable rule R s.t. R = Rfe if Rfe is applicable
         Apply R to A
*     rc := rc + 1
*     N := A \ A_0
*     Replace each a ∈ O_a (resp. x ∈ O_c) in N with a_sc (resp. x_sc)
*     A^rc_∪ := A^(rc−1)_∪ ∪ N
      return A
```

FIG. 8. The extended satisfiability algorithm.

and we do not claim that the extended version itself can be executed in polynomial space.

The extended algorithm can be found in Figure 8. The extensions are marked with asterisks. If the algorithm is started on the initial ABox $\mathcal{A}_D = \{a : D\}$, we set $\mathcal{A}^0_\cup := \{a_0 : D\}$. The algorithm uses two *global* variables sc and rc, which are both 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 ABoxes $\mathcal{A}^i_\cup$ that have already been generated. The introduction of the global variable sc is necessary due to the following technical problem: the object names created by the algorithm are unique only within the ABox considered in a single recursion step. For the accumulating ABoxes $\mathcal{A}^i_\cup$ that collect assertions from many recursion steps, we have to ensure that an object a from one recursion step can be distinguished from a in a different step since these two objects do clearly not represent the same domain element in the constructed model. To achieve this, objects are renamed before new assertions are added to an ABox $\mathcal{A}^i_\cup$ by indexing with the value of the counter sc.

Observe that, for $i > 0$, the ABox $\mathcal{A}^i_\cup$ is obtained either

1. by multiple applications of completion rules from Figure 4 to the ABox $\mathcal{A}^{i-1}_\cup$ or
2. by a recursion call made while the counter rc has value $i - 1$.

Let us be a little bit more precise about the second point. W.r.t. the sequence of ABoxes $\mathcal{A}^0_\cup, \mathcal{A}^1_\cup, \ldots$, recursion calls can be viewed as applications of the completion rules displayed in Figure 7: if $\mathcal{A}^i_\cup$ is obtained from $\mathcal{A}^{i-1}_\cup$ by a recursion call, then this is equivalent to a single application of the R∃r rule together with exhaustive application of the R∀r rule.

Non-applicability of all completion rules to an ABox will be an important property in what follows.

DEFINITION 3.9 (Complete ABox)
An ABox $\mathcal{A}$ is *complete* iff no completion rule from Figures 4 and 7 is applicable to $\mathcal{A}$.

The following two lemmas are central for proving soundness and completeness.

LEMMA 3.10
Let $\mathcal{A}$ be an ABox and R a completion rule from Figure 4 or Figure 7 such that R is applicable to $\mathcal{A}$. Then $\mathcal{A}$ is consistent iff R can be applied such that the resulting ABox $\mathcal{A}'$ is consistent.

PROOF. Let us first deal with the "if" direction. This is trivial if R $\neq$ Rfe since this implies $\mathcal{A} \subseteq \mathcal{A}'$ and, hence, every model of $\mathcal{A}'$ is also a model of $\mathcal{A}$. Assume that the Rfe rule is applied to assertions $\{(a, b) : f, (a, c) : f\} \in \mathcal{A}$ and replaces c with b. Let $\mathcal{I}$ be a model of $\mathcal{A}'$. Construct an interpretation $\mathcal{I}'$ from $\mathcal{I}$ by setting $c^{\mathcal{I}'} := b^{\mathcal{I}}$. It is straightforward to check that $\mathcal{I}'$ is a model of $\mathcal{A}$. The case that Rfe is applied to assertions $\{(a, x) : g, (a, y) : g\} \in \mathcal{A}$ is analogous.

Now for the "only if" direction. We make a case distinction according to the completion rule R.

- The R⊓ rule is applied to an assertion $a : C_1 \sqcap C_2$ and $\mathcal{A}' = \mathcal{A} \cup \{a : C_1, a : C_2\}$. Let $\mathcal{I}$ be a model of $\mathcal{A}$. Since $a^{\mathcal{I}} \in (C_1 \sqcap C_2)^{\mathcal{I}}$, we have $a^{\mathcal{I}} \in C_1^{\mathcal{I}}$ and $a^{\mathcal{I}} \in C_2^{\mathcal{I}}$ by the semantics of $\mathcal{ALCF}(\mathcal{D})$, which implies that $\mathcal{I}$ is also a model of $\mathcal{A}'$.
- The R⊔ rule is applied to an assertion $a : C_1 \sqcup C_2$. The rule can be applied such that either $\mathcal{A}' = \mathcal{A} \cup \{a : C_1\}$ or $\mathcal{A}' = \mathcal{A} \cup \{a : C_2\}$. Let $\mathcal{I}$ be a model of $\mathcal{A}$. Since $a^{\mathcal{I}} \in (C_1 \sqcup C_2)^{\mathcal{I}}$, we have either $a^{\mathcal{I}} \in C_1^{\mathcal{I}}$ or $a^{\mathcal{I}} \in C_2^{\mathcal{I}}$ by the semantics of $\mathcal{ALCF}(\mathcal{D})$. Hence, we can apply the rule such that $\mathcal{I}$ is a model of $\mathcal{A}'$.
- The R∃f rule is applied to an assertion $a : \exists f.C$ yielding the ABox $\mathcal{A}'$. Then $\mathcal{A}' = \mathcal{A} \cup \{(a, b) : f, b : C\}$ where b is fresh in $\mathcal{A}$. Let $\mathcal{I}$ be a model of $\mathcal{A}$. Since $a^{\mathcal{I}} \in (\exists f.C)^{\mathcal{I}}$, there exists a $d \in \Delta_{\mathcal{I}}$ such that $f^{\mathcal{I}}(a^{\mathcal{I}}) = d$ and $d \in C^{\mathcal{I}}$. Let $\mathcal{I}'$ be the interpretation obtained from $\mathcal{I}$ by setting $a^{\mathcal{I}'} := d$. It is easily checked that $\mathcal{I}'$ is a model of $\mathcal{A}'$.
- The R∃r rule is treated analogously to the previous case.
- The R∀f rule is applied to an assertion $a : \forall f.C$ and $\mathcal{A}' = \mathcal{A} \cup \{b : C\}$ where b is an f-successor of a in $\mathcal{A}$ and $\mathcal{A}'$. Let $\mathcal{I}$ be a model of $\mathcal{A}$. Since $a^{\mathcal{I}} \in (\forall f.C)^{\mathcal{I}}$ and $f^{\mathcal{I}}(a^{\mathcal{I}}) = b^{\mathcal{I}}$, we have $b \in C^{\mathcal{I}}$. Hence, $\mathcal{I}$ is also a model of $\mathcal{A}'$.
- The R∀r rule is treated analogously to the previous case.

- The Rc rule is applied to an assertion $a : \exists u_1, \ldots, u_n.P$ with $u_i = f_1^{(i)} \cdots f_{k_i}^{(i)} g_i$ yielding the ABox $\mathcal{A}'$. Then there exist abstract objects $a_j^{(i)}$ with $1 \leq i \leq n$ and $1 \leq j \leq k_i$ which are fresh in $\mathcal{A}$ and concrete objects $x_1, \ldots, x_n$ which are fresh in $\mathcal{A}$ such that, for $1 \leq i \leq n$,
  - $a_1^{(i)}$ is $f_1^{(i)}$-successor of a,
  - $a_j^{(i)}$ is $f_j^{(i)}$-successor of $a_{j-1}^{(i)}$ for $1 < j \leq k_i$,
  - x_i is g_i-successor of $a_{k_i}^{(i)}$, and
  - $(x_1, \ldots, x_n) : P \in \mathcal{A}'$.

  Let $\mathcal{I}$ be a model of $\mathcal{A}$. Since $a^{\mathcal{I}} \in (\exists u_1, \ldots, u_n.P)^{\mathcal{I}}$, there exist domain elements $d_j^{(i)} \in \Delta_{\mathcal{I}}$ with $1 \leq i \leq n$ and $1 \leq j \leq k_i$ and $z_1, \ldots, z_n \in \Delta_{\mathcal{D}}$ such that, for $1 \leq i \leq n$, we have
  - $(a^{\mathcal{I}}, d_1^{(i)}) \in (f_1^{(i)})^{\mathcal{I}}$,
  - $(d_{j-1}^{(i)}, d_j^{(i)}) \in (f_j^{(i)})^{\mathcal{I}}$ for $1 < j \leq k_i$,
  - $g_i^{\mathcal{I}}(d_{k_i}^{(i)}) = z_i$, and
  - $(z_1, \ldots, z_n) \in P^{\mathcal{D}}$.

  Define $\mathcal{I}'$ as the interpretation obtained from $\mathcal{I}$ by setting

  $$(a_j^{(i)})^{\mathcal{I}'} := d_j^{(i)} \text{ for } 1 \leq i \leq n \text{ and } 1 < j \leq k_i$$

  and

  $$x_i^{\mathcal{I}'} := z_i \text{ for all } i \text{ with } 1 \leq i \leq n.$$

  It is straightforward to check that $\mathcal{I}'$ is a model of $\mathcal{A}'$.
- Applications of the $\mathsf{R}{\downarrow}$ rule are treated similar to the previous case.
- Applications of the $\mathsf{R}{\uparrow}$ rule are also treated similar to the Rc case.
- The Rfe rule is applied to assertions $\{(a, b) : f, (a, c) : f\} \in \mathcal{A}$ and replaces c with b. Let $\mathcal{I}$ be a model of $\mathcal{A}$. Due to the presence of the above two assertions and since features are interpreted as partial functions, we have $b^{\mathcal{I}} = c^{\mathcal{I}}$. It is readily checked that this implies that $\mathcal{I}$ is a model of $\mathcal{A}'$. The case that two concrete objects are identified can be treated in the same way.

∎

LEMMA 3.11
Let $\mathcal{A}$ be an ABox. If $\mathcal{A}$ is complete and clash-free, then it is consistent.

PROOF. Based on $\mathcal{A}$, a canonical interpretation $\mathcal{I}$ can be defined as follows. Fix a solution δ for $\zeta_{\mathcal{A}}$ which exists since $\mathcal{A}$ is clash-free.

1. $\Delta_{\mathcal{I}}$ consists of all abstract objects used in $\mathcal{A}$,
2. $A^{\mathcal{I}} := \{a \in \mathsf{O_a} \mid a : A \in \mathcal{A}\}$ for all $A \in \mathsf{N_C}$,
3. $R^{\mathcal{I}} := \{(a, b) \in \mathsf{O_a} \times \mathsf{O_a} \mid (a, b) : R \in \mathcal{A}\}$ for all $R \in \mathsf{N_R}$,
4. $g^{\mathcal{I}} := \{(a, \delta(x)) \in \mathsf{O_a} \times \Delta_{\mathcal{D}} \mid (a, x) : g \in \mathcal{A}\}$ for all $g \in \mathsf{N_{cF}}$,
5. $a^{\mathcal{I}} := a$ for all $a \in \mathsf{O_a}$, and
6. $x^{\mathcal{I}} := \delta(x)$ for all $x \in \mathsf{O_c}$.

Note that $\mathcal{I}$ is well-defined: Since the Rfe rule is not applicable, $f^{\mathcal{I}}$ and $g^{\mathcal{I}}$ are functional for all $f \in \mathsf{N_{aF}}$ and $g \in \mathsf{N_{cF}}$. We prove that $\mathcal{I}$ is a model of $\mathcal{A}$, i.e., that all assertions in $\mathcal{A}$ are satisfied by $\mathcal{I}$. It is an immediate consequence of the definition of $\mathcal{I}$ that $(a,b) : R \in \mathcal{A}$ implies $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$ and $(a,x) : g \in \mathcal{A}$ implies $g^{\mathcal{I}}(a^{\mathcal{I}}) = x^{\mathcal{I}}$. Moreover, if $(a \neg \sim b) \in \mathcal{A}$, then $a \neq b$ since $\mathcal{A}$ is clash-free. Hence, $(a \neg \sim b) \in \mathcal{A}$ implies $a^{\mathcal{I}} \neq b^{\mathcal{I}}$. Since δ is a solution for $\zeta_{\mathcal{A}}$, $(x_1, \ldots, x_n) : P \in \mathcal{A}$ implies $(x_1^{\mathcal{I}}, \ldots, x_n^{\mathcal{I}}) \in P^{\mathcal{D}}$. It thus remains to show that $a : C \in \mathcal{A}$ implies $a \in C^{\mathcal{I}}$. This is done by induction on the structure of C. For the induction start, we make a case distinction according to the form of C:

- If $C \in \mathsf{N_C}$, then the above claim is an immediate consequence of the definition of C.
- $C = \neg E$. Since we assume all concepts to be in negation normal form, E is a concept name. Since $\mathcal{A}$ is clash-free, $a : E \notin \mathcal{A}$ and, by definition of $\mathcal{I}$, $a \notin E^{\mathcal{I}}$. Hence, $a \in (\neg E)^{\mathcal{I}}$.
- $C = \exists u_1, \ldots, u_n.P$. Since the Rc rule is not applicable to $\mathcal{A}$, there exist $x_1, \ldots, x_n \in \mathsf{O_c}$ such that x_i is u_i-successor of a in $\mathcal{A}$ for $1 < i \leq n$. By definition of $\mathcal{I}$, we have $u_i^{\mathcal{I}}(a) = \delta(x_i)$ for $1 < i \leq n$. Furthermore, we have $(x_1, \ldots, x_n) : P \in \mathcal{A}$ and, since δ is a solution for $\zeta_{\mathcal{P}}$, $(\delta(x_1), \ldots, \delta(x_n)) \in P^{\mathcal{D}}$. Summing up, $a \in (\exists u_1, \ldots, u_n.P)^{\mathcal{I}}$.
- $C = p_1{\downarrow}p_2$. Since the R$\downarrow$ rule is not applicable to $\mathcal{A}$, there exists an object $b \in \mathsf{O_a}$ which is both a p_1-successor and a p_2-successor of a in $\mathcal{A}$. By definition of $\mathcal{I}$, we have $p_1^{\mathcal{I}}(a) = p_2^{\mathcal{I}}(a) = b$ and, hence, $a \in (p_1{\downarrow}p_2)^{\mathcal{I}}$.
- $C = p_1{\uparrow}p_2$. Since the R$\uparrow$ rule is not applicable to $\mathcal{A}$, there exist $b_1, b_2 \in \mathsf{O_a}$ such that b_1 is a p_1-successor of a in $\mathcal{A}$, b_2 is a p_2-successor of a in $\mathcal{A}$, and $b_1 \neg \sim b_2 \in \mathcal{A}$. Since $\mathcal{A}$ is clash-free, we have $b_1 \neq b_2$. By definition of $\mathcal{I}$, we have $p_1^{\mathcal{I}}(a) = b_1$ and $p_2^{\mathcal{I}}(a) = b_2$ and, hence, $a \in (p_1{\uparrow}p_2)^{\mathcal{I}}$.
- $C = g{\uparrow}$. Since $\mathcal{A}$ is clash-free, a has no g-successor x in $\mathcal{A}$. By definition of $\mathcal{I}$, $g^{\mathcal{I}}(a)$ is undefined and hence $a \in (g{\uparrow})^{\mathcal{I}}$.

For the induction step, we make a case analysis according to the topmost constructor in C.

- $C = C_1 \sqcap C_2$. Since the R$\sqcap$ rule is not applicable to $\mathcal{A}$, we have $\{C_1, C_2\} \subseteq \mathcal{A}(a)$. By induction, $a \in C_1^{\mathcal{I}}$ and $a \in C_2^{\mathcal{I}}$, which implies $a \in (C_1 \sqcap C_2)^{\mathcal{I}}$.
- $C = C_1 \sqcup C_2$. Similar to the previous case.
- $C = \exists R.E$. Since neither the R$\exists$f nor the R$\exists$r rule is applicable to $\mathcal{A}$, there exists an object $b \in \mathsf{O_a}$ such that b is an R-successor of a in $\mathcal{A}$ and $E \in \mathcal{A}(b)$. By definition of $\mathcal{I}$, b being an R-successor of a implies $(a,b) \in R^{\mathcal{I}}$. By induction, we have $b \in E^{\mathcal{I}}$ and may hence conclude $a \in (\exists R.E)^{\mathcal{I}}$.
- $C = \forall R.E$. Let $b \in \Delta_{\mathcal{I}}$ such that $(a,b) \in R^{\mathcal{I}}$. By definition of $\mathcal{I}$, b is an R-successor of a in $\mathcal{A}$. Since neither the R$\forall$f not the R$\forall$r rule is applicable to $\mathcal{A}$, we have $E \in \mathcal{A}(b)$. By induction, it follows that $b \in E^{\mathcal{I}}$. Since this holds for all b, we can conclude $a \in (\forall R.E)^{\mathcal{I}}$.

∎

In the following, the *i-th recursion step* denotes the recursion step of the extended completion algorithm in which the counter *sc* has value i.

PROPOSITION 3.12 (Soundness)
If the completion algorithm returns satisfiable, then the input concept is satisfiable.

PROOF. Assume that the completion algorithm is started on an input concept D and there exists a way to make the non-deterministic decisions such that the algorithm returns satisfiable. Moreover assume that the extended algorithm constructs the ABox $\mathcal{A}^n_\cup$ if the non-deterministic decisions are made in precisely the same way, i.e., the counter rc has value n upon termination. We first establish the following claim:

Claim: $\mathcal{A}^n_\cup$ is complete and clash-free.

First for completeness. We distinguish several cases. First assume that a rule

$$\mathsf{R} \in \{\mathsf{R}\sqcap, \mathsf{R}\sqcup, \mathsf{R}\exists\mathsf{f}, \mathsf{Rc}, \mathsf{R}\downarrow, \mathsf{R}\uparrow, \mathsf{R}\exists\mathsf{r}\}$$

is applicable to $\mathcal{A}^n_\cup$. This is due to the presence of an assertion $a_i : C$ in $\mathcal{A}^n_\cup$. If, e.g., $\mathsf{R} = \mathsf{R}\sqcap$, then C has the form $C_1 \sqcap C_2$. By construction of $\mathcal{A}^n_\cup$, this implies that $a : C$ is either part of the argument $\mathcal{A}$ to sat in the i-th recursion call or has been added to $\mathcal{A}$ by the fcompl function during the i-th recursion step. In either case, if $\mathsf{R} \neq \mathsf{R}\exists\mathsf{r}$, the rule R has been applied to $a : C$ by the fcompl function during the i-th recursion step, which, again by construction of $\mathcal{A}^n_\cup$, implies that R is not applicable to $a_i : C$ in $\mathcal{A}^n_\cup$: contradiction. If $\mathsf{R} = \mathsf{R}\exists\mathsf{r}$, then $C = \exists R.E$. Clearly, $(a_i, b_j) : R$ and $b_j : C$ (for some $j > i$) is added to $\mathcal{A}^n_\cup$ due to a subsequent recursion call and we obtain a contradiction to the applicability of $\mathsf{R}\exists\mathsf{r}$ to $a_i : C$ in $\mathcal{A}^n_\cup$.

Now assume that the $\mathsf{R}\forall\mathsf{f}$ rule is applicable to $\mathcal{A}^n_\cup$. This is due to the presence of assertions $a_i : \forall f.C$ and $(a_i, b_j) : f$ in $\mathcal{A}^n_\cup$. Since assertions $(a_i, b_j) : f$ are only added to $\mathcal{A}^n_\cup$ because of applications of the rules $\mathsf{R}\exists\mathsf{f}$, Rc, $\mathsf{R}\downarrow$, and $\mathsf{R}\uparrow$ performed by the fcompl function, we have $i = j$. It follows that $a : \forall f.C$ and $(a, b) : f$ are in $\mathcal{A}$ in the i-th recursion step. Hence, the $\mathsf{R}\forall\mathsf{f}$ rule is applied by fcompl to these assertions. This implies that $b : C$ is in $\mathcal{A}$ in the i-th recursion step which allows us to conclude $b_i : C \in \mathcal{A}^n_\cup$, a contradiction.

Assume that $\mathsf{R}\forall\mathsf{r}$ is applicable to $\mathcal{A}^n_\cup$ due to the presence of assertions $a_i : \forall R.C$ and $(a_i, b_j) : R$. By construction of $\mathcal{A}^n_\cup$, $a_i : \forall R.C$ is in $\mathcal{A}$ in the i-th recursion step and $(a_i, b_j) : R$ has been added to $\mathcal{A}^n_\cup$ due to a recursion call made during the i-th recursion step. By definition of the annotated algorithm, these two facts imply that $b_j : C$ has also been added to $\mathcal{A}^n_\cup$ in the i-th recursion step. Again a contradiction.

To finish the proof that $\mathcal{A}^n_\cup$ is complete, assume that Rfe is applicable to $\mathcal{A}^n_\cup$ due to the presence of assertions $(a_i, b_j) : f$ and $(a_i, c_\ell) : f$. Since assertions $(a_i, b_j) : f$ are only added to $\mathcal{A}^n_\cup$ because of applications of the rules $\mathsf{R}\exists\mathsf{f}$, Rc, $\mathsf{R}\downarrow$, and $\mathsf{R}\uparrow$ performed by the fcompl function, we have $i = j = \ell$. It follows that $(a, b) : f$ and $(a, c) : f$ are in $\mathcal{A}$ in the i-th recursion step. Hence, the Rfe rule is applied by fcompl. This, however, implies that either $(a_i, b_j) : f$ or $(a_i, c_\ell) : f$ is not in $\mathcal{A}^n_\cup$.

We now prove that $\mathcal{A}^n_\cup(a_i)$ is clash-free. Assume $\{A, \neg A\} \subseteq \mathcal{A}^n_\cup(a_i)$. Then $\{A, \neg A\} \subseteq \mathcal{A}(a)$ in the i-th recursion step. Since $\mathcal{A}$ is clash-free in every recursion step (the algorithm returned satisfiable), we obtain a contradiction. Clashes of the form $a_i \neg \sim a_i \in \mathcal{A}^n_\cup$ are treated analogously. Now assume $a_i : g\uparrow$ and $(a_i, x_j) : g$ are in $\mathcal{A}^n_\cup$. Since assertions $(a_i, x_j) : g$ are only added due to applications of the Rc rule by fcompl, we have $i = j$. It is again straightforward to derive a contradiction.

It remains to show that $\mathcal{A}^n_\cup$ is concrete domain satisfiable. For every $i \leq n$, let $\mathcal{A}_i$ be the ABox $\mathcal{A}$ in the i-th recursion step after the application of fcompl and let

δ_i be a solution for $\zeta_{\mathcal{A}_i}$, which exists since $\mathcal{A}_i$ is clash-free. Define $\delta(x_i) := \delta_i(x_i)$ for all x_i occurring in $\mathcal{A}^n_\cup$. It is readily checked that δ is a solution for $\zeta_{\mathcal{A}^n_\cup}$: fix an assertion $((x_1)_{h_1}, \ldots, (x_k)_{h_k}) : P \in \mathcal{A}^n_\cup$. Since such assertions are only added due to applications of the Rc rule by fcompl, there exists an $i \leq n$ such that $h_j = i$ for all j with $1 \leq j \leq k$. Hence, $(x_1, \ldots, x_k) : P \in \mathcal{A}_i$ and $(\delta_i(x_1), \ldots, \delta_i(x_k)) \in P^{\mathcal{D}}$. By definition of δ, it follows that $(\delta((x_1)_{i_1}), \ldots, \delta((x_k)_{i_k})) \in P^{\mathcal{D}}$, as was to be shown.

The proof of the claim is now finished and we return to the proof of soundness. By Lemma 3.11, the claim implies that $\mathcal{A}^n_\cup$ is consistent. By construction, we have $a_0 : D \in \mathcal{A}^n_\cup$. It immediately follows that D is satisfiable. ∎

Proposition 3.13 (Completeness)
If the completion algorithm is started on a satisfiable input concept, then it returns satisfiable.

Proof. Since the completion algorithm returns satisfiable iff the extended algorithm does, it suffices to concentrate on the extended algorithm. Let the extended completion algorithm be started on an input concept D that is satisfiable. Then, the initial ABox $\mathcal{A}_D = \{a : D\}$ is obviously consistent. By Lemma 3.10 and due to the fact that performing a recursion step corresponds to the application of rules from Figure 7, we can make the non-deterministic decisions of the extended algorithm such that every ABox in the sequence $\mathcal{A}^0_\cup, \mathcal{A}^1_\cup, \ldots$ is consistent. By Proposition 3.8 and since the extended algorithm terminates iff the original one does, this sequence is comprised of a finite number n of ABoxes. Moreover, the extended algorithm does not detect a clash: if a clash is detected in an ABox $\mathcal{A}$, then we have $\mathcal{A} \subseteq \mathcal{A}^n_\cup$ up to variable renaming which clearly contradicts the consistency of $\mathcal{A}^n_\cup$. Because of this and again due to Proposition 3.8, the algorithm terminates returning satisfiable. ∎

It may be viewed as a byproduct of the soundness and completeness proof that $\mathcal{ALCF}(\mathcal{D})$ has the generalized tree model property defined in Section 3.1: assume that the extended algorithm is started with initial ABox $\mathcal{A}_D = \{a : D\}$ and that D is satisfiable. By Proposition 3.13 and the correspondence of the original and the extended algorithm, the extended algorithm returns satisfiable. From the proof of Proposition 3.12, we learn that in this case the ABox $\mathcal{A}^n_\cup$ (where n is the value of the counter *sc* upon termination) is complete and clash-free. In the proof of Lemma 3.11, a canonical model $\mathcal{I}$ of $\mathcal{A}^n_\cup$ is constructed where $\Delta_\mathcal{I}$ is the set of abstract objects used in $\mathcal{A}^n_\cup$. It is straightforward to check that this model is a generalized tree model for D since

1. $a_0 : D$ is in $\mathcal{A}^n_\cup$,
2. the sets $X_i := \{a_i \mid a_i \in \Delta_\mathcal{I}\}$ for $0 \leq i \leq n$ are equivalence classes w.r.t. $\mathcal{I}$ and $\sim$ as in Definition 3.1, and
3. due to the recursive nature of the completion algorithm, the graph $(V_\mathcal{I}, E_\mathcal{I})$ (see Definition 3.1) is a tree.

We now analyze the time and space requirements of our algorithm.

PROPOSITION 3.14

1. If $\mathcal{D}$-satisfiability is in PSPACE, then the completion algorithm can be executed in polynomial space.
2. If $\mathcal{D}$-satisfiability is in NEXPTIME, then the completion algorithm can be executed in nondeterministic exponential time.
3. If $\mathcal{D}$-satisfiability is in EXPSPACE, then the completion algorithm can be executed in exponential space.

PROOF. By Point 1 of Lemma 3.7 and Lemma 3.6, the maximum size of ABoxes $\mathcal{A}$ encountered in recursion steps is bounded by $p(|D|)$, where p is a polynomial. Since, by Point 3 of Lemma 3.7, the recursion depth is bounded by $|D|$, sat can be executed in polynomial space if the check for concrete domain satisfiability is not taken into account.

Assume that $\mathcal{D}$-satisfiability is in PSPACE. Since the maximum size of ABoxes $\mathcal{A}$ encountered in recursion steps is bounded by $p(|D|)$, the maximum number of conjuncts in predicate conjunctions $\zeta_{\mathcal{A}}$ checked for concrete domain satisfiability is also bounded by $p(|D|)$. Together with the fact that the complexity class PSPACE is oblivious for polynomial blowups of the input, it follows that the completion algorithm can be executed in polynomial space. Along the same lines, it can be shown that the algorithm can be executed in exponential space if $\mathcal{D}$-satisfiability is in EXPSPACE.

Now assume that $\mathcal{D}$-satisfiability is in NEXPTIME. From Lemma 3.6, we know that fcompl terminates after at most $|\mathcal{A}|^4$ rule applications if started on input $\mathcal{A}$. Since, by Point 1 of Lemma 3.7, the size of its input is bounded by $|D|^2$, it terminates after at most $|D|^8$ rule applications. Since the recursion depth is bounded by $|D|$, and, by Point 2 of Lemma 3.7, at most $q(|D|)$ recursion calls are made per recursion step for some polynomial q, sat can be executed in nondeterministic exponential time if the check for concrete domain satisfiability is not taken into account. By the bounds on the recursion depth and the number of recursion calls per recursion steps, the number of concrete domain satisfiability checks performed is at most exponential in $|D|$. Since the size of predicate conjunctions passed in each step is bounded by $p(D)$ and $\mathcal{D}$-satisfiability is in NEXPTIME, we can perform each check in (non-deterministic) time exponential in $|D|$. Summing up, the sat algorithm an be executed in nondeterministic exponential time. ∎

Combining this result with the PSPACE lower bound of $\mathcal{ALC}$-concept satisfiability [39] and using Savitch's Theorem which implies that PSPACE = NPSPACE and EXPSPACE = NEXPSPACE [37], we obtain the following theorem.

THEOREM 3.15

Let $\mathcal{D}$ be an admissible concrete domain.

1. If $\mathcal{D}$-satisfiability is in PSPACE, then $\mathcal{ALC}(\mathcal{D})$-concept satisfiability and $\mathcal{ALCF}(\mathcal{D})$-concept satisfiability are PSPACE-complete.
2. If $\mathcal{D}$-satisfiability is in C $\in$ {NEXPTIME, EXPSPACE}, then $\mathcal{ALC}(\mathcal{D})$-concept satisfiability and $\mathcal{ALCF}(\mathcal{D})$-concept satisfiability are also in C.

Since lower complexity bounds obviously transfer from $\mathcal{D}$-satisfiability to $\mathcal{ALCF}(\mathcal{D})$-concept satisfiability, Point 2 of this theorem yields tight complexity bounds if $\mathcal{D}$-satisfiability is NEXPTIME-complete or EXPSPACE-complete (instead of just *in* the respective class). Moreover, since subsumption can be reduced to (un)satisfiability and vice versa, we obtain corresponding complexity bounds for subsumption:

COROLLARY 3.16
Let $\mathcal{D}$ be an admissible concrete domain.

1. If $\mathcal{D}$-satisfiability is in PSPACE, then $\mathcal{ALC}(\mathcal{D})$-concept subsumption and $\mathcal{ALCF}(\mathcal{D})$-concept subsumption are PSPACE-complete.
2. If $\mathcal{D}$-satisfiability is in NEXPTIME, then $\mathcal{ALC}(\mathcal{D})$-concept subsumption and $\mathcal{ALCF}(\mathcal{D})$-concept subsumption are in CO-NEXPTIME.
3. If $\mathcal{D}$-satisfiability is in EXPSPACE then $\mathcal{ALC}(\mathcal{D})$-concept subsumption and $\mathcal{ALCF}(\mathcal{D})$-concept subsumption are in EXPSPACE.

4 ABox Consistency

In the preceding section, we used ABoxes merely as a data structure. However, ABoxes are interesting in their own right since they are frequently used to represent assertional knowledge about the state of affairs in a particular "world". In this section, we extend the complexity results obtained in the previous section from concept satisfiability to ABox consistency by devising a *precompletion algorithm* in the style of [13, 21]. Most importantly, the extended algorithm yields a tight PSPACE complexity bound for $\mathcal{ALCF}(\mathcal{D})$-ABox consistency if $\mathcal{D}$-satisfiability is in PSPACE.

4.1 The Algorithm

The algorithm works by reducing ABox consistency to concept satisfiability. First, a set of precompletion rules is exhaustively applied to the input ABox $\mathcal{A}$ yielding a *precompletion* of $\mathcal{A}$. Intuitively, rule application makes all implicit knowledge in the ABox explicit except that it does *not* generate new R-successors for roles $R \in \mathsf{N_R} \setminus \mathsf{N_{aF}}$. Then, several *reduction concepts* are generated from the precompletion and passed to the concept satisfiability algorithm devised in the previous section. The input ABox is satisfiable iff the precompletion contains no obvious contradiction and all reduction concepts are satisfiable.

The precise formulation of the algorithm can be found in Figure 9. We assume all concepts in the input ABox to be in NNF. As already mentioned in Section 3.3, the $\mathsf{precompl}$ function is identical to the fcompl function in Figure 5 except that it additionally applies the $\mathsf{R\forall r}$ rule. This is necessary since, in contrast to ABoxes processed by the sat algorithm, the input ABox to cons may contain assertions of the form $(a, b) : R$ with $R \in \mathsf{N_R} \setminus \mathsf{N_{aF}}$. Although not generating new R-successors for roles $R \in \mathsf{N_R} \setminus \mathsf{N_{aF}}$, the precompletion algorithm *does* generate new f-successors and new g-successors for features $f \in \mathsf{N_{aF}}$ and $g \in \mathsf{N_{cF}}$. Intuitively, the input ABox induces a set of clusters of objects as discussed in Section 3.1 and these clusters are constructed by the $\mathsf{precompl}$ function.

Note that the construction of a reduction concept corresponds to a single application of the $\mathsf{R\exists r}$ rule together with exhaustive application of the $\mathsf{R\forall r}$ rule very similar to recursion calls of the sat functions in Figure 5.

```
define procedure cons(𝒜)
  𝒜 := precompl(𝒜)
  if 𝒜 contains a clash then
    return inconsistent
  forall assertions ∃R.C ∈ 𝒜(a) with R ∈ N_R \ N_aF do
    Fix b ∈ O_a
    if sat({b : C ⊓ ⊓_{∀R.E∈𝒜(a)} b : E) = unsatisfiable then
      return inconsistent
  return consistent

define procedure precompl(𝒜)
  while a rule from {R⊓, R⊔, R∀r, R∀f, R∃f, Rc, R↓, R↑, Rfe}
                    is applicable to 𝒜 do
    Choose an applicable rule R s.t. R = Rfe if Rfe is applicable
    Apply R to 𝒜
  return 𝒜
```

FIG. 9. The $\mathcal{ALCF}(\mathcal{D})$-ABox consistency algorithm.

4.2 Correctness and Complexity

Termination of the precompletion algorithm is easily obtained.

PROPOSITION 4.1
The precompletion algorithm terminates on any input.

PROOF. By Lemma 3.6, the precompl function terminates, and, by Proposition 3.8, the sat function also terminates. ∎

We now prove soundness and completeness. In the following, an ABox $\mathcal{A}'$ is called a *precompletion* of an ABox $\mathcal{A}$ iff $\mathcal{A}'$ can be obtained by applying the precompl function to $\mathcal{A}$. Note that precompl is non-deterministic (due to the use of the $\mathsf{R}\sqcup$ rule) and hence there may exist more than a single precompletion for a given ABox $\mathcal{A}$.

PROPOSITION 4.2 (Soundness)
If the precompletion algorithm returns consistent, then the input ABox is consistent.

PROOF. If the algorithm is started on input ABox $\mathcal{A}$ returning consistent, then there exists a precompletion $\mathcal{A}_p$ for $\mathcal{A}$ that does not contain a clash and all reduction concepts $C_1, \ldots, C_n$ of $\mathcal{A}_p$ that are passed as arguments to the sat algorithm are satisfiable. We show that this implies that $\mathcal{A}_p$ has a model, which, by Lemma 3.10 and the definition of precompletion, proves the proposition.

Let $\mathcal{I}_1, \ldots, \mathcal{I}_n$ be the models of the reduction concepts $C_1, \ldots, C_n$ and $a_i : \exists R_i.E_i$ be the assertion in $\mathcal{A}_p$ that triggered the construction of the reduction concept C_i. W.l.o.g., we assume that

- $\Delta_{\mathcal{I}_i} \cap \Delta_{\mathcal{I}_j} = \emptyset$ for $1 \leq i < j \leq n$ and
- $\Delta_{\mathcal{I}_i} \cap \mathsf{O_a} = \emptyset$ for $1 \leq i \leq n$.

For each i with $1 \leq i \leq n$, we fix an element $d_i \in \Delta_{\mathcal{I}_i}$ with $d_i \in C_i^{\mathcal{I}_i}$. Moreover, we fix a solution δ for $\zeta_{\mathcal{A}_p}$, which exists since $\mathcal{A}_p$ is clash-free. Define an interpretation $\mathcal{I}$ as follows:

1. $\Delta_{\mathcal{I}} := \mathsf{O_a} \uplus \Delta_{\mathcal{I}_1} \uplus \cdots \uplus \Delta_{\mathcal{I}_n}$,
2. $A^{\mathcal{I}} := \{a \in \mathsf{O_a} \mid a : A \in \mathcal{A}_p\} \cup \bigcup_{1 \leq i \leq n} A^{\mathcal{I}_i}$ for all $A \in \mathsf{N_C}$,
3. $R^{\mathcal{I}} := \{(a, b) \in \mathsf{O_a} \times \mathsf{O_a} \mid (a, b) : R \in \mathcal{A}\} \cup \{(a_i, d_i) \mid 1 \leq i \leq n \text{ and } R = R_i\}$ $\cup \bigcup_{1 \leq i \leq n} R^{\mathcal{I}_i}$ for all $R \in \mathsf{N_R}$,
4. $g^{\mathcal{I}} := \{(a, \delta(x)) \in \mathsf{O_a} \times \Delta_{\mathcal{D}} \mid (a, x) : g \in \mathcal{A}\} \cup \bigcup_{1 \leq i \leq n} g^{\mathcal{I}_i}$ for all $g \in \mathsf{N_{cF}}$,
5. $a^{\mathcal{I}} := a$ for all $a \in \mathsf{O_a}$, and
6. $x^{\mathcal{I}} := \delta(x)$ for all $x \in \mathsf{O_c}$.

$\mathcal{I}$ is well-defined: due to the non-applicability of the Rfe rule to $\mathcal{A}_p$, $f^{\mathcal{I}}$ and $g^{\mathcal{I}}$ are functional for all $f \in \mathsf{N_{aF}}$ and $g \in \mathsf{N_{cF}}$. The following claim is an easy consequence of the construction of $\mathcal{I}$:

Claim: Let $1 \leq i \leq n$. For all $d \in \Delta_{\mathcal{I}_i}$ and $C \in \mathsf{sub}(\mathcal{A}_p)$, $d \in C^{\mathcal{I}_i}$ implies $d \in C^{\mathcal{I}}$.

It remains to show that $\mathcal{I}$ is a model of $\mathcal{A}_p$, i.e., that all assertions in $\mathcal{A}_p$ are satisfied by $\mathcal{I}$. For assertions of the form $(a, b) : R$ and $(a, x) : g$, this is an immediate consequence of the definition of $\mathcal{I}$. Assertions $a \neg \sim b$ are satisfied since $\mathcal{A}_p$ is clash-free and assertions $(x_1, \ldots, x_n) : P$ are satisfied since δ is a solution for $\zeta_{\mathcal{A}_p}$. It thus remains to show that $a : C \in \mathcal{A}_p$ implies $a \in C^{\mathcal{I}}$. This is done by induction over the structure of C as in the proof of Lemma 3.11. The only differences are in the following cases of the induction step:

- $a : \exists R.E \in \mathcal{A}_p$. Then there is an i with $1 \leq i \leq n$ such that $a = a_i$, $R = R_i$, and $E = E_i$ appears as a conjunct in the reduction concept C_i. By definition of $\mathcal{I}$, we have $(a, d_i) \in R^{\mathcal{I}}$. By the above claim together with $d_i \in C_i^{\mathcal{I}_i}$, we have $d_i \in C_i^{\mathcal{I}}$. Since E is a conjunct in C_i, this clearly implies $d_i \in E^{\mathcal{I}}$ and thus $a \in (\exists R.E)^{\mathcal{I}}$.
- $a : \forall R.E \in \mathcal{A}_p$. Fix a $b \in \Delta_{\mathcal{I}}$ such that $(a, b) \in R^{\mathcal{I}}$. Then either b is an R-successor of a in $\mathcal{A}_p$ or $a = a_i$, $R = R_i$, and $b = d_i$ for some $1 \leq i \leq n$. The first case was already treated in the proof of Lemma 3.11. Hence, let us stick to the second case. By construction of C_i, E appears as a conjunct in C_i. By the claim, we have $d_i \in C_i^{\mathcal{I}}$ and hence $d_i \in E^{\mathcal{I}}$.

■

PROPOSITION 4.3 (Completeness)
If the precompletion algorithm is started on a consistent input ABox, then it returns consistent.

PROOF. Suppose that the algorithm is started on a consistent ABox $\mathcal{A}$. By Lemma 3.10, the precompl function can apply the completion rules such that only consistent ABoxes are obtained. Hence, by Lemma 3.6, the precompl function generates a consistent precompletion $\mathcal{A}_p$ of $\mathcal{A}$. Consistency of $\mathcal{A}_p$ clearly implies that the reduction concepts constructed from $\mathcal{A}_p$ are satisfiable. Since, by Proposition 3.8, the sat function terminates, the precompletion algorithm also terminates and returns consistent. ■

It remains to analyze the time and space requirements of our algorithm.

PROPOSITION 4.4

1. If $\mathcal{D}$-satisfiability is in PSPACE, then the precompletion algorithm can be executed in polynomial space.
2. If $\mathcal{D}$-satisfiability is in NEXPTIME, then the precompletion algorithm can be executed in nondeterministic exponential time.
3. If $\mathcal{D}$-satisfiability is in EXPSPACE, then the precompletion algorithm can be executed in exponential space.

PROOF. Let $\mathcal{A}$ be the input ABox to the precompletion algorithm. By Lemma 3.6, the precompl function terminates after at most $|\mathcal{A}|^4$ steps generating an ABox $\mathcal{A}'$ of size at most $|\mathcal{A}|^6$. Since all complexity classes mentioned in the proposition are oblivious for polynomial blowups of the input, the concrete domain satisfiability check does not spoil the upper bound on the time/space requirements. Concerning the calls to the sat function, it suffices to refer to Proposition 3.14. ∎

As in the previous section, we use the PSPACE lower bound of $\mathcal{ALC}$-concept satisfiability and the fact that PSPACE =NPSPACE and EXPSPACE =NEXPSPACE to obtain the following theorem.

THEOREM 4.5
Let $\mathcal{D}$ be an admissible concrete domain.

1. If $\mathcal{D}$-satisfiability is in PSPACE, then $\mathcal{ALC}(\mathcal{D})$-ABox consistency and $\mathcal{ALCF}(\mathcal{D})$-ABox consistency are PSPACE-complete.
2. If $\mathcal{D}$-satisfiability is in C $\in$ {NEXPTIME, EXPSPACE}, then $\mathcal{ALC}(\mathcal{D})$-ABox consistency and $\mathcal{ALCF}(\mathcal{D})$-ABox consistency are also in C.

5 Applying the Results

We give some example applications of the results just obtained by reconsidering the concrete domains A and S introduced in Section 2. In order to apply Theorems 3.15 and 4.5, we need to determine the complexity of A-satisfiability and S-satisfiability. More precisely, we show that both problems are in NP.

Let us start with the concrete domain A. The proof is by a reduction to mixed integer programming (MIP), i.e., to linear programming where some of the variables must take integer values. More precisely, a *mixed integer programming problem* has the form $Ax = b$, where A is an $m \times n$-matrix of rational numbers, x is an n-vector of variables, each of them being either an integer variable or a rational variable, and b is an m-vector of rational numbers (see, e.g. [40]). A *solution* of $Ax = b$ is a mapping δ that assigns an integer to each integer variable in x and a rational number to each rational variable in x such that the equality $Ax = b$ holds. Deciding the *satisfiability* of a MIP problem means to decide whether such a problem has a solution.

PROPOSITION 5.1
A-satisfiability is in NP.

PROOF. We sketch a non-deterministic polynomial time algorithm for A-satisfiability. The algorithm is based on several normalization steps, simple inconsistency checks, and a final call to an algorithm which is capable of deciding the satisfiability of MIP problems.

Let c be a finite conjunction of A predicates. The following steps are executed sequentially to decide the satisfiability of c:

1. Return unsatisfiable if c contains the $\bot_{\mathsf{A}}$ predicate.
2. Eliminate all occurrences of the $\top_{\mathsf{A}}$ predicate from c and call the result c_1.
3. Eliminate each occurrence of predicates $\overline{\mathsf{int}}$, P_q, and $+$:
   - replace each conjunct $\overline{\mathsf{int}}(x)$ with the conjuncts
     $$>(x, f),\ \mathsf{int}(f),\ =_1(o),\ +(f, o, f'),\ <(x, f'),$$
     where f, f', o are fresh (i.e. previously unused) variables.
   - replace each conjunct $P_q(x)$ (where $P \in \{<, \leq, \neq, \geq, >\}$ and $q \in \mathbb{Q}$) with the two conjuncts $=_q(f)$ and $P(x, f)$, where f is a fresh variable.
   - replace each conjunct $\overline{+}(x, y, z)$ with $+(x, y, f)$ and $\neq(f, z)$, where f is a fresh variable.

   Call the result c_2
4. Eliminate each occurrence of the predicates $\leq$, $\neq$, $\geq$, and $>$ in c_2: conjuncts $\leq(x, y)$ are non deterministically replaced with either $<(x, y)$ or $=(x, y)$. The other predicates can be treated similarly. Call the result c_3. Note that c_3 does only contain the predicates int, $=_q$, $<$, $=$, and $+$.
5. Transform c_3 into a MIP problem in the obvious way:
   - every variable x used in c_3 such that $\mathsf{int}(x)$ is a conjunct of c_3 becomes an integer variable in the MIP problem. All other variables appearing in c_3 become rational variables;
   - every conjunct $=_q(x)$ is translated into an equation $x = q$;
   - every conjunct $=(x, y)$ is translated into an equation $x - y = 0$;
   - every conjunct $<(x, y)$ is translated into an equation $x + s - y = 0$, where s is a fresh rational variable (also known as slack variable);
   - every conjunct $+(x, y, z)$ is translated into an equation $x + y - z = 0$.

   Use a standard NP algorithm to decide the satisfiability of this problem and return the result.

It is straightforward to prove the correctness of the sketched algorithm by showing that (i) each of the normalization steps preserves (un)satisfiability, and (ii) the reduction to MIP is correct. Moreover, it is not hard to see that the algorithm can be executed in nondeterministic polynomial time: each of the normalization steps leads to at most a polynomial blowup of the size of the predicate conjunction. Finally, deciding the satisfiability of MIP problems can be done in NP [14]. ∎

An application of Theorems 3.15 and 4.5 immediately yields the complexity of reasoning with the Description Logic $\mathcal{ALCF}(\mathsf{A})$.

COROLLARY 5.2
$\mathcal{ALCF}(\mathsf{A})$-concept satisfiability and $\mathcal{ALCF}(\mathsf{A})$-ABox consistency are PSPACE-complete.

Now for the concrete domain S. It is straightforward to reduce S-satisfiability to the satisfiability problem of so-called RCC8 networks [10, 36]. Such a network is simply a finite set of assertions $\mathsf{rd}(X, Y)$, where rd is a disjunction $\mathsf{rel}_0 \vee \cdots \vee \mathsf{rel}_k$ of RCC8 relations and X and Y are region variables from some fixed set of variables $\mathcal{V}$. A triple

$\langle U, T, \delta\rangle$, where (U,T) is a topology and δ maps each region variable from $\mathcal{V}$ to an element of T, is a *model* of an RCC8 network N iff, for each $\mathsf{rel}_0 \vee \cdots \vee \mathsf{rel}_k(X,Y) \in N$, there exists an $i \leq k$ such that $\delta(X)\ \mathsf{rel}_i\ \delta(Y)$. N is satisfiable iff it has a model.

Proposition 5.3
S-satisfiability is in NP.

Proof. It is easy to reduce S-satisfiability to the satisfiability of RCC8 networks: given a finite conjunction c of predicates from Φ_{S}, first eliminate any occurrences of the $\top_{\mathsf{s}}$ predicate and return unsatisfiable if c contains the $\bot_{\mathsf{s}}$ predicate; then replace all predicates $\overline{\mathsf{rel}}$ by the disjunction of all elements of RCC8 $\setminus \{\mathsf{rel}\}$, where RCC8 denotes the set of all eight RCC8 relations; finally, translate each conjunct in c into an RCC8 assertion $\mathsf{rd}(X,Y)$ in the obvious way. As shown by Renz and Nebel in [36], the satisfiability of the resulting RCC8 network can be decided in nondeterministic polynomial time. Moreover, every satisfiable RCC8 network has a model in the topological space $\mathcal{RC}_{\mathbb{R}^2}$ [35]. ∎

Again, we obtain the desired corollary by applying Theorems 3.15 and 4.5.

Corollary 5.4
$\mathcal{ALCF}(\mathsf{S})$-concept satisfiability and $\mathcal{ALCF}(\mathsf{S})$-ABox consistency are PSpace-complete.

6 Discussion and Related Work

In this paper, we have established tight complexity bounds for concept- and ABox-reasoning with the basic Description Logic with concrete domains $\mathcal{ALC}(\mathcal{D})$ and its extensions with feature (dis)agreements $\mathcal{ALCF}(\mathcal{D})$. The upper bound for concept satisfiability has been obtained by a completion algorithm that uses the tracing technique while the upper bound for ABox consistency has been established by a precompletion-style reduction to concept satisfiability. We have strictly separated the algorithms for these two reasoning problems since this makes more explicit the additional means necessary for dealing with ABoxes instead of with concepts. However, for the implementation of DL reasoners that can decide ABox consistency, it may be more appropriate to use a "direct" ABox consistency algorithm instead of reducing this reasoning task to concept satisfiability. Considering the two algorithms developed in this paper, it should be straightforward to devise such a direct algorithm.

Using an arithmetic concrete domain A and a spatial concrete domain S, we have demonstrated the relevance of the obtained complexity results: since A-satisfiability and S-satisfiability are in NP, it follows from the established complexity bounds that concept- and ABox-reasoning with both $\mathcal{ALCF}(\mathsf{A})$ and $\mathcal{ALCF}(\mathsf{S})$ is PSpace-complete. We have also established upper bounds for the case that $\mathcal{D}$-satisfiability is in NExpTime or ExpSpace. A rather expressive concrete domain R based on Tarski algebra (also known as real closed fields), for which R-satisfiability is ExpSpace-complete, can be found in [30, 5]. Using the results from this paper and the obvious fact that $\mathcal{D}$-satisfiability can be polynomially reduced to $\mathcal{ALC}(\mathcal{D})$-concept satisfiability, we immediately obtain ExpSpace-completeness of concept- and ABox-reasoning with the Description Logic $\mathcal{ALC}(\mathsf{R})$. Other important concrete domains that are captured by the presented results are the temporal ones that can be found in [33, 30, 27].

The results presented in this paper have stimulated interesting further research. For example, in [3] the PSpace upper bound for $\mathcal{ALCF}(\mathcal{D})$-concept satisfiability has been used to obtain a PSpace upper bound for reasoning with the interval-based temporal Description Logic $\mathcal{TL}$-$\mathcal{ALCF}$, which was first described in [2]. Perhaps most interesting, it has been found that the PSpace upper bounds established in this paper are fragile in the following sense: there exist several standard means of expressivity whose addition to $\mathcal{ALC}(\mathcal{D})$ leads to the complexity of reasoning leaping from PSpace-completeness to NExpTime-completeness—at least for so-called arithmetic concrete domains [28, 30, 1]. Examples for such means of expressivity include acyclic TBoxes, inverse roles, nominals, and role conjunction. This is particularly surprising since (i) the mentioned means of expressivity are usually considered "harmless" w.r.t. the complexity of reasoning, i.e., for most standard DLs, their addition does *not* change the complexity of reasoning; (ii) many concrete domains suggested in the literature are arithmetic; and (iii) there exist rather simple arithmetic concrete domains $\mathcal{D}$—in particular some for which $\mathcal{D}$-satisfiability is in PTime.

References

[1] Carlos Areces and Carsten Lutz. Concrete domains and nominals united. In Carlos Areces, Patrick Blackburn, Maarten Marx, and Ulrike Sattler, editors, *Proceedings of the fourth Workshop on Hybrid Logics (HyLo'02)*, 2002.

[2] Alessandro Artale and Enrico Franconi. A temporal description logic for reasoning about actions and plans. *Journal of Artificial Intelligence Research (JAIR)*, 9:463–506, 1998.

[3] Alessandro Artale and Carsten Lutz. A correspondence between temporal description logics. In Patrick Lambrix, Alex Borgida, Maurizio Lenzerini, Ralf Möller, and Peter Patel-Schneider, editors, *Proceedings of the International Workshop on Description Logics (DL'99)*, number 22 in CEUR-WS (http://ceur-ws.org/), pages 145–149, 1999.

[4] Franz Baader and Philipp Hanschke. A scheme for integrating concrete domains into concept languages. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 452–457, Sydney, Australia, 1991.

[5] Franz Baader and Philipp Hanschke. A scheme for integrating concrete domains into concept languages. DFKI Research Report RR-91-10, German Research Center for Artificial Intelligence (DFKI), 1991.

[6] Franz Baader and Philipp 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. Springer-Verlag, 1992.

[7] Franz Baader and Bernhard Hollunder. A terminological knowledge representation system with complete inference algorithm. In *Proceedings of the Workshop on Processing Declarative Knowledge (PDK-91)*, volume 567 of *Lecture Notes in Artificial Intelligence*, pages 67–86. Springer-Verlag, 1991.

[8] Franz Baader, Deborah L. McGuiness, Daniele Nardi, and Peter Patel-Schneider. *The Description Logic Handbook: Theory, implementation and applications.* Cambridge University Press, 2002. To appear.

[9] Franz Baader and Ulrike Sattler. Tableau algorithms for description logics. In R. Dyckhoff, editor, *Proceedings of the International Conference on Automated Reasoning with Tableaux and Related Methods (Tableaux 2000)*, volume 1847 of *Lecture Notes in Artificial Intelligence*, pages 1–18. Springer-Verlag, 2000.

[10] Brandon Bennett. Modal logics for qualitative spatial reasoning. *Journal of the Interest Group in Pure and Applied Logic*, 4(1), 1997.

[11] Ronald J. Brachman, Deborah L. McGuinness, Peter F. Patel-Schneider, Lori Alperin Resnick, and Alexander Borgida. Living with classic: When and how to use a KL-ONE-like language. In John F. Sowa, editor, *Principles of Semantic Networks – Explorations in the Representation of Knowledge*, chapter 14, pages 401–456. Morgan Kaufmann, 1991.

[12] Giuseppe De Giacomo and Maurizio Lenzerini. TBox and ABox reasoning in expressive description logics. In *Proceedings of the Fifth International Conference on the Principles of Knowledge Representation and Reasoning (KR'96)*, pages 316–327. Morgan Kaufmann Publishers, 1996.

[13] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Andrea Schaerf. Deduction in concept languages: from subsumption to instance checking. *Journal of Logic and Computation*, 4(4):423–452, 1994.

[14] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, San Francisco, CA, USA, 1979.

[15] Erich Grädel. On the restraining power of guards. *Journal of Symbolic Logic*, 64:1719–1742, 1999.

[16] Volker Haarslev, Carsten Lutz, and Ralf Möller. A description logic with concrete domains and role-forming predicates. *Journal of Logic and Computation*, 9(3):351–384, 1999.

[17] Volker Haarslev and Ralf Möller. RACER system description. In Rajeev Goré, Alexander Leitsch, and Tobias Nipkow, editors, *Proceedings of the First International Joint Conference on Automated Reasoning (IJCAR'01)*, number 2083 in Lecture Notes in Artifical Intelligence, pages 701–705. Springer-Verlag, 2001.

[18] Volker Haarslev, Ralf Möller, and Michael Wessel. The description logic $\mathcal{ALCNH}_{R+}$ extended with concrete domains: A practically motivated approach. In Rajeev Goré, Alexander Leitsch, and Tobias Nipkow, editors, *Proceedings of the First International Joint Conference on Automated Reasoning IJCAR'01*, number 2083 in Lecture Notes in Artifical Intelligence, pages 29–44. Springer-Verlag, 2001.

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

[20] Philipp Hanschke. Specifying role interaction in concept languages. In William Nebel, Bernhard; Rich, Charles; Swartout, editor, *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning (KR'92)*, pages 318–329. Morgan Kaufmann, 1992.

[21] Bernhard Hollunder. Consistency checking reduced to satisfiability of concepts in terminological systems. *Annals of Mathematics and Artificial Intelligence*, 18:133–157, 1996.

[22] Bernhard Hollunder and Werner Nutt. Subsumption algorithms for concept languages. DFKI Research Report RR-90-04, German Research Center for Artificial Intelligence (DFKI), Kaiserslautern, Germany, 1990.

[23] Ian Horrocks and Peter Patel-Schneider. The generation of DAML+OIL. In Carole Goble, Deborah L. McGuinness, Ralf Möller, and Peter F. Patel-Schneider, editors, *Proceedings of the International Workshop in Description Logics 2001 (DL2001)*, number 49 in CEUR-WS (http://ceur-ws.org/), pages 30–35, 2001.

[24] Ian Horrocks and Ulrike Sattler. Ontology reasoning in the $\mathcal{SHOQ}(D)$ description logic. In Bernhard Nebel, editor, *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI'01)*, pages 199–204. Morgan-Kaufmann, 2001.

[25] Martina Kullmann, François de Bertrand de Beuvron, and François Rousselot. A description logic model for reacting in a dynamic environment. In F. Baader and U. Sattler, editors, *Proceedings of the 2000 International Workshop in Description Logics (DL2000)*, number 33 in CEUR-WS (http://ceur-ws.org/), pages 203–212, 2000.

[26] Richard E. Ladner. The computational complexity of provability in systems of modal propositional logic. *SIAM Journal on Computing*, 6(3):467–480, 1977.

[27] Carsten Lutz. Interval-based temporal reasoning with general TBoxes. In Bernhard Nebel, editor, *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI'01)*, pages 89–94. Morgan-Kaufmann, 2001.

[28] Carsten Lutz. NExpTime-complete description logics with concrete domains. In Rajeev Goré, Alexander Leitsch, and Tobias Nipkow, editors, *Proceedings of the First International Joint Conference on Automated Reasoning (IJCAR'01)*, number 2083 in Lecture Notes in Artifical Intelligence, pages 45–60. Springer-Verlag, 2001.

[29] Carsten Lutz. Adding numbers to the $\mathcal{SHIQ}$ description logic—First results. In *Proceedings of the Eighth International Conference on Principles of Knowledge Representation and Reasoning (KR2002)*. Morgan Kaufman, 2002.

[30] Carsten Lutz. *The Complexity of Reasoning with Concrete Domains.* PhD thesis, LuFG Theoretical Computer Science, RWTH Aachen, Germany, 2002.

[31] Carsten Lutz. Reasoning about entity relationship diagrams with complex attribute dependencies. In Ian Horrocks and Sergio Tessaris, editors, *Proceedings of the International Workshop in Description Logics 2002 (DL2002)*, number 53 in CEUR-WS (http://ceur-ws.org/), pages 185–194, 2002.

[32] Carsten Lutz. Description logics with concrete domains—a survey. In *Advances in Modal Logics (AiML) 2002*, To appear.

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

[34] David A. Randell, Zhan Cui, and Anthony G. Cohn. A spatial logic based on regions and connection. In Bernhard Nebel, Charles Rich, and William Swartout, editors, *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning (KR'92)*, pages 165–176. Morgan Kaufman, 1992.

[35] Jochen Renz. A canonical model of the region connection calculus. In Anthony G. Cohn, Lenhart Schubert, and Stuart C. Shapiro, editors, *KR'98: Principles of Knowledge Representation and Reasoning*, pages 330–341. Morgan Kaufmann, San Francisco, California, 1998.

[36] Jochen Renz and Bernhard Nebel. On the complexity of qualitative spatial reasoning: A maximal tractable fragment of the region connection calculus. *Artificial Intelligence*, 108(1–2):69–123, 1999.

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

[38] Andrea Schaerf. On the complexity of the instance checking problem in concept languages with existential quantification. *Journal of Intelligent Information Systems*, 2:265–278, 1993.

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

[40] Alexander Schrijver. *Theory of Linear and Integer Programming*. Wiley, Chichester, UK, 1986.

[41] Sergio Tessaris, Ian Horrocks, and Graham Gough. Evaluating a modular abox algorithm. In *Proceedings of the Eighth International Conference on Principles of Knowledge Representation and Reasoning (KR2002)*, pages 227–239. Morgan Kaufman, 2002.

[42] Moshe Y. Vardi. Why is modal logic so robustly decidable? In Neil Immerman and Phokion G. Kolaitis, editors, *Descriptive Complexity and Finite Models*, volume 31 of *DIMACS: Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, 1997.

Received 27 September 2002