Complexity of Terminological Reasoning Revisited

Carsten Lutz

RWTH Aachen, LuFG Theoretical Computer Science
Ahornstr. 55, 52074 Aachen

Abstract. TBoxes in their various forms are key components of knowledge representation systems based on description logics (DLs) since they allow for a natural representation of terminological knowledge. Largely due to a classical result given by Nebel [15], complexity analyses for DLs have, until now, mostly failed to take into account the most basic form of TBoxes, so-called *acyclic* TBoxes. In this paper, we concentrate on DLs for which reasoning without TBoxes is PSPACE-complete, and show that there exist logics for which the complexity of reasoning remains in PSPACE if acyclic TBoxes are added and also logics for which the complexity increases. This demonstrates that it is necessary to take acyclic TBoxes into account for complexity analyses.

1 Introduction

A core feature of description logics is their ability to represent and reason about terminological knowledge. Terminological knowledge is stored in so-called TBoxes which mainly come in two flavours. So-called *acyclic* TBoxes are sets of concept definitions that can be thought of as non-recursive macro definitions whereas *general* TBoxes allow to state equivalence of arbitrary, complex concepts. In this paper, we consider the complexity of reasoning with acyclic TBoxes.[1] Surprisingly, although computational complexity of reasoning is a major topic in description logic research, most complexity results available concentrate either on reasoning without TBoxes or on reasoning with general TBoxes (see, e.g., [7], [8], [9], and [10]).

There are two main reasons for this. The first reason is that acyclic TBoxes are a properly subsumed by general TBoxes. However, for many DLs, reasoning with acyclic TBoxes can be expected to be less complex than reasoning with general TBoxes, and, hence, it is interesting to know the exact complexity of reasoning with them. Moreover, there exist description logics for which reasoning with general TBoxes is undecidable but reasoning with acyclic TBoxes is not. In this case, it is obviously desirable to determine the complexity of reasoning with acyclic TBoxes.

[1] Hence, when talking of TBoxes, we generally refer to acyclic TBoxes unless otherwise noted.

The second reason can be understood historically. Early DL systems used unfolding to reduce reasoning with acyclic TBoxes to reasoning with concepts. Unfolding a concept C w.r.t. a TBox $\mathcal{T}$ means iteratively replacing concept names in C by their definitions given in $\mathcal{T}$. For example, the result of unfolding the concept $\mathit{Man} \sqcap \exists \mathit{married\text{-}to}.\mathit{Wife}$ w.r.t. the TBox

$$\{\mathit{Man} \doteq \neg \mathit{Female}, \quad \mathit{Wife} \doteq \mathit{Female} \sqcap \mathit{Married}\}$$

yields $\neg \mathit{Female} \sqcap \exists \mathit{married\text{-}to}.(\mathit{Female} \sqcap \mathit{Married})$. In his seminal paper, Nebel showed that, in the worst case, unfolding may result in an exponential blow-up of the concept size [15]. Since the complexity of reasoning with description logics is usually not ExpSpace-hard, this result shows that unfolding is not an adequate means for treating TBoxes. Nebel also showed that in realistic, practical applications, the worst case is almost never encountered. Largely due to these results (and possibly misunderstandings of these results), complexity analyses of reasoning with acyclic TBoxes have long been neglected: First, one could (wrongly) think that reasoning with acyclic TBoxes is necessarily ExpSpace-hard, and that it is sensible to consider only general TBoxes since this—given the misunderstanding—does not seem to make things harder. Second, since the worst case seems not to occur in most practical applications, one could be tempted to think that unfolding is a proper tool for DL systems and that it is not rewarding to search for better alternatives. Last, if one is only interested in decidability of concepts w.r.t. acyclic TBoxes, unfolding is a technique which is easy to use and always applicable.

For many DLs, reasoning without TBoxes is PSpace-complete (see, e.g., [9], [12], [18]). Although the complexity of reasoning with acyclic TBoxes is rarely addressed formally, it is "common knowledge" in the DL community that, if reasoning without TBoxes is in PSpace, then taking into account TBoxes does "usually" not increase complexity. This knowledge has been exploited for efficient practical reasoning with TBoxes [5], but has, to the best of our knowledge, never been used to obtain theoretical complexity results. This is even more surprising since Nebel showed that there exist DLs for which reasoning w.r.t. TBoxes is harder than reasoning with concepts, only (in Nebel's case, complexity moved from P to NP) [15].

In this paper, we focus on logics for which "pure concept satisfiability" (i.e., concept satisfiability w.r.t. the empty TBox) is PSpace-complete and explore the impact of TBoxes on the complexity of the basic DL reasoning tasks satisfiability and subsumption. It turns out that there exist logics for which reasoning remains in PSpace and also DLs for which reasoning gets significantly harder. In the first part of this paper, we focus on $\mathcal{ALC}$, the basic description logic for which pure concept satisfiability is in PSpace [17]. The "common knowledge" mentioned above is used to demonstrate how a pure $\mathcal{ALC}$ concept satisfiability algorithm using the so-called *trace technique* [17] can be modified to take into account TBoxes such that the resulting algorithm can still be executed in polynomial space. Roughly speaking, TBoxes have to be converted to a normal form which allows the tracing algorithm to operate exclusively on concept

names (instead of concept expressions). Using the presented modification technique, it is proved that satisfiability of $\mathcal{ALC}$ concepts w.r.t. acyclic TBoxes is still PSPACE-complete.

In the second part of this paper, we show that this technique does not always work: there exist description logics for which pure concept satisfiability is PSPACE-complete but the extension by TBoxes makes reasoning harder. We identify $\mathcal{ALCF}$, i.e., the extension of $\mathcal{ALC}$ with features, feature agreement and feature disagreement, to be such a logic. Pure concept satisfiability is known to be PSPACE-complete for this logic [11]. Using a reduction of a constrained version of the domino problem, it is proved that satisfiability of $\mathcal{ALCF}$ concepts w.r.t. TBoxes is NEXPTIME-hard. Applying the modification technique from the first part to an existing algorithm, it is shown that it is also in NEXPTIME and hence NEXPTIME-complete.

2 Description Logics

In this section, the description logic $\mathcal{ALCF}$ is introduced (see also [11]). All logics considered in this paper are fragments of $\mathcal{ALCF}$.

Definition 1. *Let N_C, N_R, and N_F be disjoint sets of concept, role, and feature names. A composition $f_1 \cdots f_n$ of features is called a* feature chain. *The set of $\mathcal{ALCF}$ concepts is the smallest set such that*

1. *every concept name is a concept (*atomic *concepts), and*
2. *if C and D are concepts, R is a role or feature, and u_1 and u_2 are feature chains, then the following expressions are also concepts: $\neg C$, $C \sqcap D$, $C \sqcup D$, $\forall R.C$, $\exists R.C$, $u_1 \downarrow u_2$, and $u_1 \uparrow u_2$.*

Let A be a concept name and C be a concept. Then $A \doteq C$ is a concept definition. *Let $\mathcal{T}$ be a finite set of concept definitions. A concept name A* directly uses *a concept name B in $\mathcal{T}$ if there is a concept definition $A \doteq C$ in $\mathcal{T}$ such that B appears in C. Let* uses *be the transitive closure of "directly uses". $\mathcal{T}$ is called* acyclic *if there is no concept name A such that A uses itself in $\mathcal{T}$. If $\mathcal{T}$ is acyclic, and, furthermore, the left-hand sides of all concept definitions in $\mathcal{T}$ are unique, then $\mathcal{T}$ is called a* TBox.

Let $R_1, \ldots, R_n$ be features or roles. We will use $\forall R_1 \ldots R_n.C$ ($\exists R_1 \ldots R_n.C$) as an abbreviation for $\forall R_1.\forall R_2 \ldots \forall R_n.C$ ($\exists R_1.\exists R_2 \ldots \exists R_n.C$). $\mathcal{ALCF}$ concepts which do not contain features are called $\mathcal{ALC}$ concepts. Next, we define the semantics of the language introduced.

Definition 2. *An* interpretation $\mathcal{I} = (\Delta_\mathcal{I}, \cdot^\mathcal{I})$ *is a pair $(\Delta_\mathcal{I}, \cdot^\mathcal{I})$. $\Delta_\mathcal{I}$ is 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 $\Delta_\mathcal{I} \times \Delta_\mathcal{I}$, and*
- *each feature name f to a partial function $f^\mathcal{I}$ from $\Delta_\mathcal{I}$ to $\Delta_\mathcal{I}$.*

If $u = f_1 \cdots f_k$ is a feature chain, then $u^{\mathcal{I}}$ is defined as the composition $f_1^{\mathcal{I}} \circ \cdots \circ f_k^{\mathcal{I}}$ of the partial functions $f_1^{\mathcal{I}}, \ldots, f_k^{\mathcal{I}}$. Let the symbols C, D, R, u_1, and u_2 be defined as in Definition 1. The interpretation function can inductively be extended to complex concepts as follows:

$$
\begin{aligned}
(C \sqcap D)^{\mathcal{I}} &:= C^{\mathcal{I}} \cap D^{\mathcal{I}} \\
(C \sqcup D)^{\mathcal{I}} &:= C^{\mathcal{I}} \cup D^{\mathcal{I}} \\
(\neg C)^{\mathcal{I}} &:= \Delta_{\mathcal{I}} \setminus C^{\mathcal{I}} \\
(\exists R.C)^{\mathcal{I}} &:= \{a \in \Delta_{\mathcal{I}} \mid \exists b \in \Delta_{\mathcal{I}} \colon (a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\} \\
(\forall R.C)^{\mathcal{I}} &:= \{a \in \Delta_{\mathcal{I}} \mid \forall b \colon (a,b) \in R^{\mathcal{I}} \rightarrow b \in C^{\mathcal{I}}\} \\
(u_1 \downarrow u_2)^{\mathcal{I}} &:= \{a \in \Delta_{\mathcal{I}} \mid \exists b \in \Delta_{\mathcal{I}} \colon u_1^{\mathcal{I}}(a) = b \wedge u_2^{\mathcal{I}}(a) = b\} \\
(u_1 \uparrow u_2)^{\mathcal{I}} &:= \{a \in \Delta_{\mathcal{I}} \mid \exists b_1, b_2 \in \Delta_{\mathcal{I}} \colon b_1 \neq b_2 \wedge \\
&\qquad\qquad u_1^{\mathcal{I}}(a) = b_1 \wedge u_2^{\mathcal{I}}(a) = b_2\}
\end{aligned}
$$

An interpretation $\mathcal{I}$ is a model *of a TBox $\mathcal{T}$ iff it satisfies $A^{\mathcal{I}} = C^{\mathcal{I}}$ for all concept definitions $A \doteq C$ in $\mathcal{T}$. A concept C* subsumes *a concept D* w.r.t. a TBox $\mathcal{T}$ *(written $D \preceq_{\mathcal{T}} C$) iff $D^{\mathcal{I}} \subseteq C^{\mathcal{I}}$ for all models $\mathcal{I}$ of $\mathcal{T}$. A concept C is* satisfiable w.r.t. a TBox $\mathcal{T}$ *iff there exists a model $\mathcal{I}$ of $\mathcal{T}$ such that $C^{\mathcal{I}} \neq \emptyset$.*

Subsumption can be reduced to satisfiability since $D \preceq_{\mathcal{T}} C$ iff the concept $D \sqcap \neg C$ is unsatisfiable w.r.t. $\mathcal{T}$. Satisfiability can be reduced to subsumption since C is unsatisfiable w.r.t. $\mathcal{T}$ iff $C \preceq_{\mathcal{T}} \bot$, where $\bot$ is an abbreviation for $A \sqcap \neg A$.

Sometimes, generalized concept definitions called "general concept inclusions" (GCIs) are considered. A GCI has the form $C \sqsubseteq D$, where both C and D are (possibly complex) concepts. An interpretation $\mathcal{I}$ is a model for a GCI $C \sqsubseteq D$ iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. TBoxes containing GCIs are called *generalized*. In this paper, we will not admit generalized TBoxes unless explicitly mentioned.

2.1 Extending Completion Algorithms

Most satisfiability algorithms for description logics are so-called *completion algorithms*, which check the satisfiability of concepts by trying to explicitly construct a canonical model. Completion algorithms are described by a rule set and a strategy to apply these rules. The rules operate on constraint systems, i.e., partial descriptions of models. Constraints are comprised of objects, concepts and roles. In the following, we will present a completion algorithm for deciding satisfiability of $\mathcal{ALC}$ concepts w.r.t. the empty TBox which was first described in [17]. We will then show how this algorithm can be modified to handle TBoxes. Both the original algorithm and its extension can be executed in polynomial space. The modification scheme presented is also applicable to a variety of other description logics.

The algorithm requires $\mathcal{ALC}$ concepts to be in negation normal form. A concept is in *negation normal form* (*NNF*) iff negation occurs only in front of

atomic concepts. It is easy to see that any $\mathcal{ALC}$ concept can be converted into an equivalent one in NNF in linear time by exhaustively applying the following rewrite rules:

- $\neg(C \sqcap D) \to (\neg C \sqcup \neg D), \quad \neg(C \sqcup D) \to (\neg C \sqcap \neg D), \quad \neg\neg C \to C$
- $\neg(\exists R.C) \to \forall R.\neg C, \quad \neg(\forall R.C) \to \exists R.\neg C$

Definition 3. *Let O_A be a set of object names. For $a, b \in O_A$, an $\mathcal{ALC}$ concept C, and $R \in N_R$, the expressions $a:C$ and $(a,b):R$ are $\mathcal{ALC}$* constraints. *A finite set of constraints S is called an $\mathcal{ALC}$* constraint system. *Interpretations can be extended to constraint systems by mapping every object name to an element of $\Delta_{\mathcal{I}}$. The unique name assumption is not imposed, i.e. $a^{\mathcal{I}} = b^{\mathcal{I}}$ may hold even if a and b are distinct object names. An interpretation $\mathcal{I}$ satisfies a constraint*

$$a:C \quad \textit{iff} \quad a^{\mathcal{I}} \in C^{\mathcal{I}}, \quad \textit{and} \quad (a,b):R \quad \textit{iff} \quad (a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}.$$

An interpretation is a model *of a constraint system S iff it satisfies all constraints in S.*

To decide the satisfiability of an $\mathcal{ALC}$ concept C in NNF (w.r.t. the empty TBox), the algorithm starts with the constraint system $S_0 := \{a : C\}$ and repeatedly applies completion rules. If a constraint system is found which does not contain a contradiction and to which no completion rule is applicable, then this constraint system has a model, which implies the existence of a model for C w.r.t. the empty TBox. If no such constraint system can be found, C is unsatisfiable. One of the completion rules is nondeterministic, i.e., there is more than one possible outcome of a rule application. Hence, the described completion algorithm is a nondeterministic decision procedure, i.e., it returns *satisfiable* iff there is a way to make the nondeterministic decisions such that a positive result is obtained.

Definition 4. *The following* completion rules *replace a given constraint system S nondeterministically by a constraint system S'. S' is called a* descendant *of S. An object $a \in O_A$ is called fresh in S if a is not used in S. In the following, C and D denote concepts, R a role, and a and b object names from O_A.*

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

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

R*$\exists$ *C *The exists restriction rule.*
If $a:\exists R.C \in S$, and there is no $b \in O_A$ such that $\{(a,b):R,\ b:C\} \subseteq S$, then $S' := S \cup \{(a,b):R\ ,\ b:C\}$ where $b \in O_A$ is fresh in S.

R*$\forall$ *C *The value restriction rule.*
If $a:\forall R.C \in S$ and there is a $b \in O_A$ such that $(a,b) : R \in S \wedge b : C \notin S$, then $S' := S \cup \{b:C\}$

A constraint system S is called contradictory *iff $\{a : C, a : \neg C\} \subseteq S$ for some*

```
define procedure sat(S)
    while a rule r from {R⊓, R⊔} is applicable to S
        S := apply(S, r)
        if S is contradictory then
            return unsatisfiable
    forall a:∃R.D ∈ S do
        Let b be an object name from O_A.
        if sat({b:D} ∪ {b:E | a:∀R.E ∈ S}) = unsatisfiable then
            return unsatisfiable
    return satisfiable
```

Fig. 1. The algorithm for deciding satisfiability of $\mathcal{ALC}$ concepts w.r.t. the empty TBox.

$a \in O_A$ *and* $C \in N_C$. *A constraint system to which no completion rules are applicable is called* complete.

Let *apply* be a function which takes a constraint system S and a completion rule r as argument, applies r once to an arbitrary set of constraints in S matching r's premise and returns the resulting constraint system. The algorithm for deciding satisfiability of $\mathcal{ALC}$ concepts is given in Figure 1. It takes a constraint system $\{x : C\}$ as input and returns *satisfiable* if C is satisfiable w.r.t. the empty TBox and *unsatisfiable* otherwise. In order to describe the space requirements of the sat algorithm, a formal notion of the size of concepts is introduced.

Definition 5. *For a concept C, the* size *of C (denoted by $||C||$) is defined as the number of symbols (operators, concept and role names) it contains. For a TBox $\mathcal{T}$, the* size *of $\mathcal{T}$ (denoted by $||\mathcal{T}||$) is defined as the sum of the sizes of the right-hand sides of all concept definitions in $\mathcal{T}$. The* role depth *of a concept C is the nesting depth of exists and value restrictions in C.*

In [17], it is proved that the described algorithm is correct and can be executed in polynomial space.[2] The latter is a consequence of the following facts:

- The recursion depth of sat is bounded by the role depth.
- In each recursion step, the constraints in the constraint system S involve a single object, only. For each object, there can be at most $\mathcal{O}(||C||)$ constraints. The size of each constraint is bounded by $||C||$.

As already argued in the introduction, using unfolding to generalize sat to TBoxes is not a good choice since the space requirements of the resulting algorithm would no longer be polynomial. However, there exists a better strategy for dealing with TBoxes, which is described in the following.

In order to allow for a succinct definition of the extended algorithm, we need to introduce a special form of TBoxes.

[2] Schmidt-Schauß and Smolka present the algorithm in a different form. In the form presented here, the algorithm first appeared in [4].

Definition 6. *A TBox $\mathcal{T}$ is called* simple *iff it satisfies the following requirements:*

- *The right-hand side of each concept definition in $\mathcal{T}$ contains exactly one operator.*
- *If the right hand side of a concept definition in $\mathcal{T}$ is $\neg A$, then A does not occur on the left hand side of any concept definition in $\mathcal{T}$.*

The following lemma shows that restricting ourselves to simple TBoxes is not a limitation.

Lemma 1. *Any TBox $\mathcal{T}$ can be converted into a simple one $\mathcal{T}'$ in linear time, such that $\mathcal{T}'$ is equivalent to $\mathcal{T}$ in the following sense: Any model for $\mathcal{T}'$ can be extended to a model for $\mathcal{T}$ and vice versa.*

Proof: The conversion can be done in three steps as follows.

1. *eliminate non-atomic negation.* (i) convert the right-hand sides of all concept definitions in $\mathcal{T}$ to NNF. (ii) For each definition $A \doteq C$ in $\mathcal{T}$, add a new definition $\overline{A} \doteq \mathit{nnf}(\neg C)$, where $\mathit{nnf}(\neg C)$ denotes the result of converting $\neg C$ to NNF. (iii) For every atomic concept A occurring on the left-hand side of a concept definition in $\mathcal{T}$, replace every occurrence of $\neg A$ in $\mathcal{T}$ with $\overline{A}$.
2. *break up concepts.* Exhaustively apply the following rewrite rules. In the following, C denotes a non-atomic concept and D an arbitrary concept.
   - $A \doteq C \sqcap D \quad \rightarrow \quad A \doteq A' \sqcap D,\ A' \doteq C$ (and analogous for $\sqcup$)
   - $A \doteq D \sqcap C \quad \rightarrow \quad A \doteq D \sqcap A',\ A' \doteq C$ (and analogous for $\sqcup$)
   - $A \doteq \exists R.C \quad \rightarrow \quad A \doteq \exists R.A',\ A' \doteq C$ (and analogous for $\forall$)

   In all cases, A' is a concept name not yet used in $\mathcal{T}$. Please note that if a definition $A \doteq \neg C$ is in $\mathcal{T}$, then, due to the first step, C is atomic and does not occur on the left-hand side of a concept definition.
3. *eliminate redundant names.* For each concept definition $A \doteq A'$, where both A and A' are atomic, replace every occurrence of A' in $\mathcal{T}$ with A. Remove the definition from $\mathcal{T}$.

The correctness of the above procedure is easily seen. The loosened form of equivalence is necessary since $\mathcal{T}'$ contains additional atomic concepts, and, furthermore, some "redundant" atomic concepts from $\mathcal{T}$ may not exist in $\mathcal{T}'$. Let $\mathcal{T}$ be a TBox and $\mathcal{T}'$ be the result of applying the above procedure. The first step can be performed in linear time since NNF conversion needs linear time and the number of concept definitions is at most doubled. Since the number of rewrite rule applications in the second step is bounded by the number of operators in $\mathcal{T}$, this step can also be performed in linear time. This obviously also holds for the third step. □

From the above result, it immediately follows that, for any TBox $\mathcal{T}$, there exists an equivalent simple one $\mathcal{T}'$ such that $||\mathcal{T}'||$ is of order $\mathcal{O}(||\mathcal{T}||)$. We will now modify the sat algorithm to decide the satisfiability of an atomic concept A w.r.t. a simple TBox $\mathcal{T}$. Using the modified algorithm, it is also possible to decide the satisfiability of non-atomic concepts C w.r.t. TBoxes $\mathcal{T}$: Add a

definition $A \doteq C$ to $\mathcal{T}$ (where A is a new concept name in $\mathcal{T}$), convert the resulting TBox to simple form and start the algorithm with $(A, \mathcal{T}')$ where $\mathcal{T}'$ is the newly obtained TBox. The modified algorithm works on constraint systems of a restricted form. In constraints of the form $a : C$, C must be a concept name (which may be the left-hand side of a concept definition in $\mathcal{T}$).

Definition 7. *Let A be an atomic concept and $\mathcal{T}$ be a simple $\mathcal{ALC}$ TBox. Making use of the existing* sat *algorithm, an algorithm* tbsat*, which returns satisfiable if A is satisfiable w.r.t. $\mathcal{T}$ and unsatisfiable otherwise, is given as follows.*

1. *Modify the completion rules of* sat *as follows: In the premise of each completion rule, substitute "$a : C \in S$" by "$a : A \in S$ and $A \doteq C \in \mathcal{T}$". E.g., in the conjunction rule, "$a : C \sqcap D \in S$" is replaced by "$a : A \in S$ and $A \doteq C \sqcap D \in \mathcal{T}$".*
2. *Start the* sat *algorithm with the initial constraint system $\{x : A\}$, where x is an arbitrary object name. Use the modified rules for the* sat *run.*

Unlike unfolding, the described algorithm has the advantage that it can be executed in polynomial space.

Proposition 1. *The* tbsat *algorithm is sound and complete and can be executed in polynomial space.*

Proof: Let $(A, \mathcal{T})$ be an input to tbsat and let C be the result of unfolding A w.r.t. $\mathcal{T}$. Please note that C is in NNF since $\mathcal{T}$ is in simple form. The correctness of tbsat can be proved by showing that a run of tbsat on input $(A, \mathcal{T})$ yields the same result as a run of sat on input C. This, in turn, can be proved by induction over the number of recursion steps. It is important to note that, at every point in the computation where a nondeterministic decision has to be made (deciding which rule to apply or deciding which consequence of the R$\sqcup$ rule to use), the available choices are exactly the same for both algorithms.

It is an immediate consequence of the following facts that the tbsat algorithm can be executed in polynomial space.

- The recursion depth of tbsat is bounded by $||\mathcal{T}||$. This is the case since (i) runs of tbsat on $(A, \mathcal{T})$ are equivalent to runs of sat on C and (ii) the role depth of C is bounded by $||\mathcal{T}||$.[3] The second point can be seen as follows: Assume that the role depth of C exceeds $||\mathcal{T}||$. This means that the right hand side of a concept definition $A' \doteq \exists R.D$ or $A' \doteq \forall R.D$ in $\mathcal{T}$ contributes to the role depth more than once. From this, however, it follows that unfolding D w.r.t. $\mathcal{T}$ yields a concept containing A' which is a contradiction to the acyclicity of $\mathcal{T}$.
- In each recursion step, the constraints in the constraint system S involve a single object, only. The number of constraints per object is bounded by the number of definitions in $\mathcal{T}$ and the maximum size of constraints is constant.

□

[3] I.e., although unfolding may lead to an exponential blow-up in concept size [15], the role depth is "preserved".

The following theorem is an immediate consequence of the above result.

Theorem 1. *Deciding satisfiability of $\mathcal{ALC}$ concepts w.r.t. acyclic TBoxes is* PSPACE*-complete.*

The use of the presented modification scheme is not limited to $\mathcal{ALC}$. In order to give an intuition of when the proposed modification can be applied to yield a PSPACE algorithm, let us summarize why the modification is successful in the case of $\mathcal{ALC}$. As a prerequisite, a completion algorithms is needed which uses tracing, i.e., which performs depth-first search over role successors. In the case of $\mathcal{ALC}$, the recursion depth of this algorithm is bounded by the role depth of the input concept C. As opposed to the concept size, the role depth is "preserved by unfolding", i.e., if a concept C is unfolded w.r.t. a TBox $\mathcal{T}$, then the role depth of the unfolded concept C' is linear in $||C|| + ||\mathcal{T}||$. This fact is used to argue that the recursion depth of the modified algorithm is linear in the size of its input.

The other important point in the proof of Proposition 1 is that the $\mathcal{ALC}$ tracing algorithm considers constraints for only one object per recursion step and so does the modified algorithm. What is important here is, again, that the number of objects considered in a single recursion step is describable by a function which is "preserved by unfolding" (the constant 1 in the case of $\mathcal{ALC}$).

For a formalization of "preservation by unfolding", the notion of a u-stable function (where "u" stands for unfolding) is introduced. A function f mapping concepts to natural numbers is called *u-stable* w.r.t. a description logic $\mathcal{L}$ iff the following holds: There exists an integer k such that, for all atomic concepts A and all $\mathcal{L}$ TBoxes $\mathcal{T}$, if C is the result of unfolding A w.r.t. $\mathcal{T}$, then $f(C)$ is of order $\mathcal{O}(||\mathcal{T}||^k)$. As was shown in the proof of Proposition 1, the role-depth of concepts is an example for a u-stable function. An example for a function which is not u-stable is the size of concepts (as Nebel proved [15]). A rule of thumb can now be formulated as follows:

> *The described modification can be applied to completion algorithms $\mathcal{A}$ which decide satisfiability for a logic $\mathcal{L}$ w.r.t. the empty TBox. Assume that $\mathcal{A}$ performs depth-first search over role-successors and can be executed in polynomial space. If $\mathcal{A}$ expands the constraints of $\alpha(C)$ objects per recursion step and $\mathcal{A}$'s recursion depth is bounded by $\beta(C)$, where C is the input concept and α and β are functions which are u-stable w.r.t. $\mathcal{L}$, then the modified algorithm can be expected to be executable in polynomial space.*

This rule of thumb can, e.g., be applied to the description logic $\mathcal{ALCNR}$ (see [9]). $\mathcal{ALCNR}$ extends $\mathcal{ALC}$ by (unqualified) number restrictions[4] and role conjunction.

Conjecture. *Deciding the satisfiability of $\mathcal{ALCNR}$ concept w.r.t. TBoxes is a* PSPACE*-complete problem.*

[4] We follow Donini et al. and assume unary coding of numbers.

Why is the rule of thumb applicable to $\mathcal{ALCNR}$? Donini et al. [9] give a PSPACE algorithm for deciding satisfiability of $\mathcal{ALCNR}$ concepts w.r.t. empty TBoxes which performs depth-first search over role successors. Its recursion depth is bounded by the role depth of the input concept C. In each recursion step, constraints for at most $ex(C)+1$ objects are expanded where $ex(C)$ is the number of *distinct* existentially quantified subconcepts of C. It is easy to prove that $ex(\cdot)$ is a u-stable function. Assume that C is the result of unfolding an atomic concept A w.r.t. a TBox $\mathcal{T}$ and that $ex(C) \geq ||\mathcal{T}||$. It follows that there exists a concept definition $B_0 \doteq \exists R.B_1$ in $\mathcal{T}$ such that B_0 uses an atomic concept B_2 (where possibly $B_1 = B_2$) and that B_2 can be replaced by *different* concepts during unfolding. This, however, is a contradiction to the definition of TBoxes, since the uniqueness of left-hand sides of concept definitions is mandatory.

3 $\mathcal{ALCF}$ and TBoxes: The Lower Bound

Given the modification scheme for satisfiability algorithms described in the previous section, it is a natural question to ask if there are any relevant description logics for which reasoning w.r.t. the empty TBox is in PSPACE but reasoning w.r.t. TBoxes is not. In the following, we will answer this question to the affirmative by showing that the hardness of reasoning with the logic $\mathcal{ALCF}$ [11] moves from PSPACE to NEXPTIME if TBoxes are admitted.

A domino problem is given by a finite set of *tile types*. All tile types are of the same size, each type has a quadratic shape and colored edges. Of each type, an unlimited number of tiles is available. The problem is to arrange these tiles to cover a torus[5] of exponential size without holes or overlapping, such that adjacent tiles have identical colors on their common edge (rotation of the tiles is not allowed). Please note that this is a restricted version of the (undecidable) general domino problem where a tiling of the first quadrant of the plane is asked for.

Definition 8. *Let $\mathcal{D} = (D, H, V)$ be a* domino system*, where D is a finite set of tile types and $H, V \subseteq D \times D$. Let $U(s,t)$ be the torus $\mathbb{Z}_s \times \mathbb{Z}_t$, where $\mathbb{Z}_n$ denotes the set $\{0, \ldots, n-1\}$. Let $w = w_0, \ldots, w_{n-1}$ be an n-tuple of tiles (with $n \leq s$). We say that $\mathcal{D}$* tiles *$U(s,t)$* with initial condition *w iff there exists a mapping $\tau : U(s,t) \to D$ such that for all $(x,y) \in U(s,t)$:*

- *if $\tau(x,y) = d$ and $\tau(x \oplus_s 1, y) = d'$ then $(d,d') \in H$*
- *if $\tau(x,y) = d$ and $\tau(x, y \oplus_t 1) = d'$ then $(d,d') \in V$*
- *$\tau(i,0) = w_i$ for $0 \leq i < n$.*

where $\oplus_n$ denotes addition modulo n.

Börger et al. show that it is NEXPTIME-complete to decide if, for a given domino system $\mathcal{D}$ and a given n-tuple w, $\mathcal{D}$ tiles $U(2^n, 2^n)$ with initial condition w [6]. In the following, we will reduce this domino problem to satisfiability of $\mathcal{ALCF}$ concepts w.r.t. TBoxes. We will first give an informal explanation of

[5] i.e., a rectangular grid whose edges are "glued" together

$$
\begin{aligned}
\mathit{Tree}_0 &\doteq \beta^n \gamma{\downarrow}\alpha^n \sqcap \\
&\quad \exists\alpha.\mathit{Tree}_1 \sqcap \exists\beta.\mathit{Tree}_1 \\
&\quad \sqcap \alpha\beta^{n-1}\gamma{\downarrow}\beta\alpha^{n-1} \\
\mathit{Tree}_1 &\doteq \exists\alpha.\mathit{Tree}_2 \sqcap \exists\beta.\mathit{Tree}_2 \\
&\quad \sqcap \alpha\beta^{n-2}\gamma{\downarrow}\beta\alpha^{n-2} \\
&\quad \vdots \\
\mathit{Tree}_{n-1} &\doteq \exists\alpha.\mathit{Tree}_n \sqcap \exists\beta.\mathit{Tree}_n \\
&\quad \sqcap \alpha\gamma{\downarrow}\beta \\
\mathit{Tree}_n &\doteq \mathit{Grid}_n
\end{aligned}
$$

Fig. 2. The $\mathcal{ALCF}$ reduction TBox $\mathcal{T}[\mathcal{D}, w, n]$: Tree definition. Substitute (α, β, γ) by (f, g, y) or (u, v, x).

how the reduction works and then formally prove its correctness. For the sake of readability, the reduction TBox $\mathcal{T}[\mathcal{D}, w, n]$ is split into two figures. Models of the reduction TBox represent solutions of instances of the domino problem. To be more precise, models of C w.r.t. $\mathcal{T}[\mathcal{D}, w, n]$ (Figure 3) encode a grid of size 2^n which has the form of a torus and is properly tiled by $\mathcal{D}$. The nodes of the grid are represented by domain objects, horizontal edges are represented by the feature x and vertical edges by the feature y. Please note that the grid may "collapse", i.e., the $2^n \times 2^n$ nodes are not necessarily distinct. Nevertheless, models of C w.r.t. $\mathcal{T}[\mathcal{D}, w, n]$ define a tiling of the full $2^n \times 2^n$ torus.

The first task is to enforce two cyclic feature chains of length 2^n, which will be edges of the grid. This is done by defining a binary tree of depth n whose leaf nodes are connected by a cyclic feature chain. The corresponding concept Tree_0 can be found in Figure 2. Please note that since two trees are needed, the TBox in the Figure has to be instantiated twice, where (α, β, γ) is substituted by (f, g, y) and (u, v, x), respectively. The first instantiation yields a y chain (of length 2^n) and the second one an x chain.

Consider the concept C in Figure 3, which glues together all the necessary building parts. It refers to the Tree_0 concept to build up two trees and it enforces the identification of the "beginning" nodes in the two (cyclic) leaf chains. The next task is to build the remaing grid which is done by the Grid_i concepts in Figure 3. The features $d_1, \ldots, d_n$ are diagonals in the grid (each d_i spans 2^{i-1} "grid cells") and play a central rôle in the grid definition. The use of these diagonals allows the definition of the (exponentially sized) grid by a TBox of polynomial size. First observe that each object on the two cyclic feature chains (row 0 and column 0 of the torus to be defined) is in the extension of Grid_n and hence also of Grid_0. Because of this, each object on the chains has d_1, x, and y fillers such that the d_1 filler coincides with the xy and yx filler. Together with the cyclicity of the initial feature chains, this properly defines row 1 and

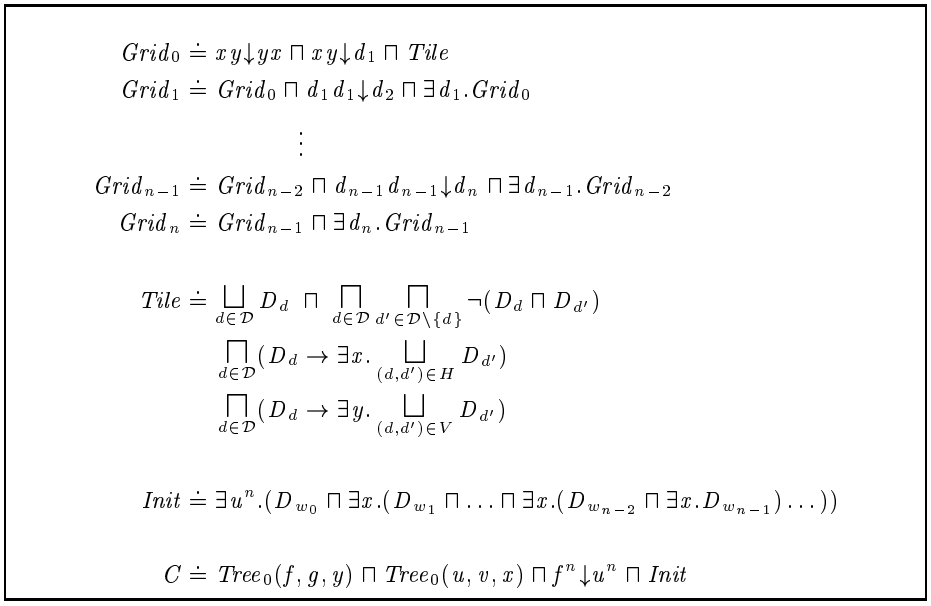

Fig. 3. The $\mathcal{ALCF}$ reduction TBox $\mathcal{T}[\mathcal{D}, w, n]$: Grid definition and tiling.

column 1 of the torus. Since the objects on the initial chains are in the extension of $Grid_1$, the objects on row 1 and column 1, which are d_1 fillers of objects on the initial chains, are in the extension of $Grid_0$. Hence, we can repeat the argument for row/column 1 and conclude the proper definition of row/column 2. Now observe that the objects on row/column 2 are d_2 fillers of the objects on the initial chain. Hence, they are in the extension of both the $Grid_0$ and $Grid_1$ concept and we can repeat the entire argument from above to derive the existence of rows/columns 3 and 4. This "doubling" can be repeated n times because of the existence of the features $d_1, \ldots, d_n$ and yields rows/columns $0, \ldots, 2^n$ of the torus. The cyclicity of the initial feature chains ensures that the edges of the grid are properly "glued" to form a torus, i.e., that row/column 2^n coincides with row/column 0. Figure 4 shows a clipping from a grid as enforced by the reduction TBox.

The grid represents the structure to be tiled. The final task is to define the tiling itself. Domino types are represented by atomic concepts D_d. Because of the definition of $Grid_0$, each node in the grid is in the extension of the concept *Tile*. The *Tile* concept ensures that, horizontally as well as vertically, the tiling condition is satisfied (we use $C \rightarrow D$ as an abbreviation for $\neg C \sqcup D$). The *Init* concept enforces the initial condition w. In the following, a formal proof of the correctness of the reduction is given.

Proposition 2. *Satisfiability and subsumption of $\mathcal{ALCF}$ concepts w.r.t. TBoxes is* NExpTime*-hard.*

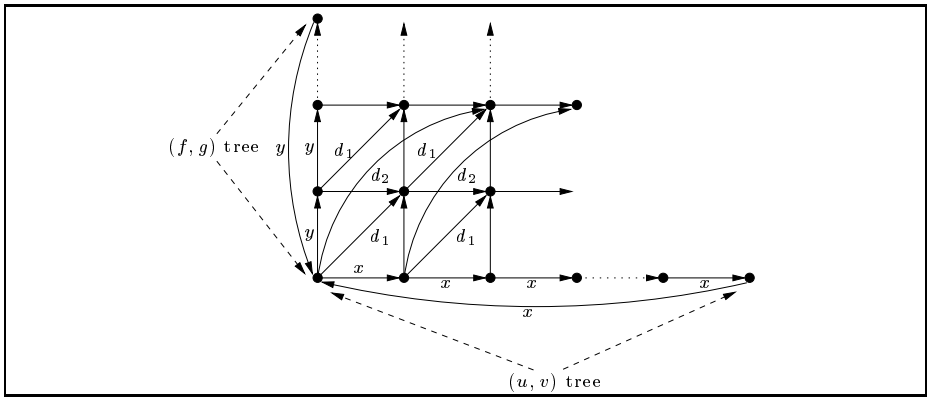

Fig. 4. Clipping of a model of the reduction concept C.

Proof:

($\Rightarrow$) Let $\mathcal{I}$ be a model of C w.r.t. $\mathcal{T}[\mathcal{D}, w, n]$. To prove that $\mathcal{D}$ tiles $U(2^n, 2^n)$ with initial condition w, it needs to be shown that there is a mapping τ as introduced in Definition 8.
As argued above, there exist $2^n \times 2^n$ (not necessarily distinct) objects $a_{i,j}$ in $\Delta_\mathcal{I}$ which form a torus w.r.t. the features x and y, i.e., $x^\mathcal{I}(a_{i,j}) = a_{(i \oplus_{2^n} 1),j}$ and $y^\mathcal{I}(a_{i,j}) = a_{i,(j \oplus_{2^n} 1)}$. All objects in the torus are in the extension of the *Tile* concept. This concept encodes the properties required for τ in Definition 8. Hence, τ can be defined as follows: $\tau := \{(i, j, d) \mid a_{i,j} \in D_d\}$. This function is well-defined since the *Tile* concept ensures that none of the $a_{i,j}$ is in the extension of two concepts D_d and $D_{d'}$, where $d \neq d'$.

($\Leftarrow$) Assume that the domino system $\mathcal{D}$ tiles $U(2^n, 2^n)$ with initial condition w (which is of length n). This means that there exists a mapping τ as defined in Definition 8. In the following, we define a model for C w.r.t. $\mathcal{T}[\mathcal{D}, w, n]$. The model has the form as discussed above: There are two binary trees of depth n whose leaf nodes are connected by a feature chain. These two chains of length 2^n are edges of a grid of size $2^n \times 2^n$. The edges of the grid are "glued" together. Let the interpretation $\mathcal{I}$ be defined as follows:

$$\Delta_\mathcal{I} = \{a_{i,j} \mid 0 \leq i, j < 2^n\} \cup \{b_{i,j}, c_{i,j} \mid 0 \leq i < n, 0 \leq j < 2^i\}$$

$$f^\mathcal{I}(b_{0,0}) := b_{1,0}, \quad g^\mathcal{I}(b_{0,0}) := b_{1,1}, \quad u^\mathcal{I}(b_{0,0}) := c_{1,0}, \quad v^\mathcal{I}(b_{0,0}) := c_{1,1}$$
$$\forall i, j \text{ where } 0 < i < n-1,\ 0 \leq j < 2^i:$$
$$f^\mathcal{I}(b_{i,j}) := b_{(i+1),(2j)}, \quad g^\mathcal{I}(b_{i,j}) := b_{(i+1),(2j+1)}$$
$$u^\mathcal{I}(c_{i,j}) := c_{(i+1),(2j)}, \quad v^\mathcal{I}(c_{i,j}) := c_{(i+1),(2j+1)}$$

$$\forall 0 \leq i < 2^{n-1}:$$
$$f^\mathcal{I}(b_{(n-1),i}) := a_{0,(2i)}, \quad g^\mathcal{I}(b_{(n-1),i}) := a_{0,(2i+1)},$$
$$u^\mathcal{I}(c_{(n-1),i}) := a_{(2i),0}, \quad v^\mathcal{I}(c_{(n-1),i}) := a_{(2i+1),0}$$

$$\forall 0 \leq i,j < 2^n : x^{\mathcal{I}}(a_{i,j}) := a_{(i \oplus_{2^n} 1),j}, \quad y^{\mathcal{I}}(a_{i,j}) := a_{i,(j \oplus_{2^n} 1)}$$
$$\forall 0 \leq i,j < 2^n, 1 \leq k \leq n : d_k^{\mathcal{I}}(a_{i,j}) := a_{(i \oplus_{2^n} 2^{k-1}),(j \oplus_{2^n} 2^{k-1})}$$
$$\forall d \in D : D_d^{\mathcal{I}} := \{a_{x,y} \mid \tau(x,y) = d\}$$

It is straightforward to verify that $\mathcal{I}$ is in fact a model for C w.r.t. $\mathcal{T}[\mathcal{D}, w, n]$: The $b_{i,j}$ objects form a tree of depth n where edges are labelled with f and g. The n-th level of the tree consists of the objects $a_{0,0}, \ldots, a_{0,2^n}$. Similarly, the $c_{i,j}$ objects form a u,v-tree where the n-th level consists of the objects $a_{0,0}, \ldots, a_{2^n,0}$ and the root is the object $b_{0,0}$. The $a_{i,j}$ objects make up a grid w.r.t. the features x and y (and diagonals d_i) which satisfies the *Tile* concept since the extension of the D_d concepts is defined through the tiling τ. Hence, it can be concluded that the object $b_{0,0}$ is an instance of C w.r.t. $\mathcal{T}[\mathcal{D}, w, n]$.

It is easy to verify that the size of $\mathcal{T}[\mathcal{D}, w, n]$ is of order $\mathcal{O}(n^2)$. Hence, the reduction can be performed in polynomial time. □

In contrast to agreements on roles (called "role value maps"), agreements on features are frequently believed to "not harm" w.r.t. decidability and complexity. The presented reduction indicates that this is not always the case. Furthermore, if TBoxes are extended with GCIs, the given reduction can easily be extended to an undecidability proof. Consider the following TBox:

$$D \doteq \top$$
$$\top \sqsubseteq xy{\downarrow}yx$$
$$\top \sqsubseteq \mathit{Tile}$$

where *Tile* is defined as in Figure 3. It induces a (possibly) infinite grid and satisfiability of D implies a complete tiling of the first quadrant.[6] Hence, decidability of $\mathcal{ALCF}$ with GCIs contradicts the undecidability of the general domino problem. For the reduction TBox, only the operators atomic negation, conjunction, disjunction, feature agreement and existential quantification over features is required. The result just obtained is already known in feature logic (see [2, Theorem 6.3], where it was proved by a reduction of the word problem for finitely presented groups).

4 $\mathcal{ALCF}$ and TBoxes: The Upper Bound

In order to prove that the satisfiability of $\mathcal{ALCF}$ concepts w.r.t. TBoxes is a NExpTime-complete problem, it remains to be shown that the satisfiability of $\mathcal{ALCF}$ concepts w.r.t. TBoxes can be decided in nondeterministic exponential time.

[6] The induced grid may also have the form of a torus since we don't enforce distinct nodes. In this case, however, a tiling of the torus induces a periodic tiling of the first quadrant.

In [14], a completion algorithm for deciding satisfiability of $\mathcal{ALCF}(\mathcal{D})$ concepts w.r.t. empty TBoxes is given which can be executed in polynomial space. $\mathcal{ALCF}(\mathcal{D})$ is the extension of $\mathcal{ALCF}$ by so-called concrete domains. By removing the completion rules and clash conditions dealing with the concrete domain, we will adapt this algorithm to $\mathcal{ALCF}$. Furthermore, we will show that an extension of the obtained algorithm to TBoxes as described in Section 2.1 can be executed in exponential time. The algorithm operates on constraint systems of the following form.

Definition 9. *Let f be a feature and a and b elements of O_A. Then, the following expressions are $\mathcal{ALCF}$* constraints*:*

$$\textit{All } \mathcal{ALC} \textit{ constraints}, \quad (a,b):f, \quad a \neq b$$

A finite set of $\mathcal{ALCF}$ constraints is called an $\mathcal{ALCF}$ constraint system*. An* interpretation *for $\mathcal{ALCF}$ constraint systems is defined identically to interpretations for $\mathcal{ALC}$ constraint systems. An interpretation satisfies a constraint*

$$\begin{aligned} (a,b):f \quad &\textit{iff} \quad (a^{\mathcal{I}}, b^{\mathcal{I}}) \in f^{\mathcal{I}} \textit{ and} \\ a \neq b \quad &\textit{iff} \quad a^{\mathcal{I}} \neq b^{\mathcal{I}}. \end{aligned}$$

A constraint system S is said to contain a *fork* (for a feature f) if it contains the two constraints $(a,b):f$ and $(a,c):f$. A fork can be *eliminated* by replacing all occurrences of c in S with b. During rule application, it is assumed that forks are eliminated as soon as they appear (as an integral part of the rule application) with the proviso that newly generated object are replaced by older ones.

Before the algorithm itself is described, we introduce the set of completion rules. In order to provide a succinct description of the rules, two auxiliary functions need to be defined. For an object $a \in O_A$ and a feature chain u, $succ_S(a,u)$ denotes the object b that can be found by following u starting from a in S. If no such object exists, $succ_S(a,u)$ denotes the special object ϵ that cannot be part of any constraint system. Let $a, b \in O_A$ and $u = f_1 \cdots f_k$ be a feature chain. The function *chain* is defined as follows:

$$\begin{aligned} chain_S(a,b,u) := \{&(a,c_1):f_1, \ldots, (c_{k-1},b):f_k\} \\ &\text{where the } c_1, \ldots, c_{k-1} \in O_A \text{ are distinct and fresh in } S. \end{aligned}$$

We now give the completion rules for the algorithm.

Definition 10. *The following* completion rules *replace a given constraint system S nondeterministically by a constraint system S'. In the following, C denotes a concepts, $\hat{R}$ a role, f a feature, u_1 and u_2 feature chains, and a and b object names from O_A.*

***R*$\sqcap$, *R*$\sqcup$** *As in Definition 4*

Rr*$\exists$ *C *The role exists restriction rule.*
If $a : \exists \hat{R}.C \in S$ and there is no $b \in O_A$ such that $\{(a,b):\hat{R},\ b:C\} \subseteq S$
Then $S' := S \cup \{(a,b):\hat{R}\ ,\ b:C\}$ where $b \in O_A$ is fresh in S.

Rf*$\exists$ *C *The feature exists restriction rule (may create forks).*
If $a:\exists f.C \in S$ *and there is no* $b \in O_A$ *such that* $\{(a,b):f,\ b:C\} \subseteq S$
Then $S' := S \cup \{(a,b):f\ ,\ b:C\}$ *where* $b \in O_A$ *is fresh in* S.

Rr*$\forall$ *C *The role value restriction rule.*
If $a:\forall \hat{R}.C \in S$ *and there is a* $b \in O_A$ *such that* $(a,b):\hat{R} \in S \wedge b:C \notin S$
Then $S' := S \cup \{b:C\}$

Rf*$\forall$ *C *The feature value restriction rule.*
If $a:\forall f.C \in S$ *and there is a* $b \in O_A$ *such that* $(a,b):f \in S \wedge b:C \notin S$
Then $S' := S \cup \{b:C\}$

***R*$\downarrow$** *The agreement rule (may create forks).*
If $a:u_1{\downarrow}u_2 \in S$, *there is no* $b \in O_A$ *such that* $succ_S(a,u_1) = succ_S(a,u_2) = b$
Then $S_0 := S \cup chain_S(a,b,u_1)$ *where* $b \in O_A$ *is fresh in* S.
$S' := S_0 \cup chain_{S_0}(a,b,u_2)$

***R*$\uparrow$** *The disagreement rule (may create forks).*
If $a:u_1{\uparrow}u_2 \in S$ *and there are no* $b_1, b_2 \in O_A$ *such that*
$succ_S(a,u_1) = b_1, succ_S(a,u_2) = b_2$ *and* $b_1 \neq b_2 \in S$
Then $S_0 := S \cup chain_S(a,b_1,u_1)$ *and* $S' := S_0 \cup chain_{S_0}(a,b_2,u_2) \cup \{b_1 \neq b_2\}$
where $b_1, b_2 \in O_A$ *are distinct and fresh in* S.

An $\mathcal{ALCF}$ *constraint system* S *is called* contradictory *iff any of the following clash triggers apply:*

- Primitive clash*:* $a:C \in S, \quad a:\neg C \in S$
- Agreement clash*:* $a \neq a \in S$

The algorithm expects the input concept C to be in negation normal form. Conversion to NNF can be done in linear time by applying the rules given in Section 2.1 together with the following rules:

- $\neg(u_1{\downarrow}u_2) \rightarrow \forall u_1.\bot \sqcup \forall u_2.\bot \sqcup u_1{\uparrow}u_2$
- $\neg(u_1{\uparrow}u_2) \rightarrow \forall u_1.\bot \sqcup \forall u_2.\bot \sqcup u_1{\downarrow}u_2$

We are now ready to give the satisfiability algorithm itself.

Definition 11. *The function* sat *decides the satisfiability of* $\mathcal{ALCF}$ *concepts in NNF w.r.t. the empty TBox. To decide the satisfiability of the concept* C, sat *takes the input* $\{x:C\}$.

define procedure sat(S)
  $S' :=$ feature-complete(S)
  if S' *contains a clash* **then**
    return *inconsistent*
  forall $a:\exists \hat{R}.D \in S'$, *where* $\hat{R}$ *is a role,* **do**
    Let b *be an object name from* O_A.
    if sat($\{b:D\} \cup \{b:E \mid a:\forall \hat{R}.E \in S'\}$) = *inconsistent* **then**
      return *inconsistent*
  return *consistent*

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

The correctness of the described algorithm can be easily seen: It corresponds to the algorithm given in [14] for deciding satisfiability of $\mathcal{ALCF}(\mathcal{D})$ concepts with all rules and clash triggers concerning the concrete part left out. Since the original algorithm is correct for $\mathcal{ALCF}(\mathcal{D})$, it is obviously also correct for $\mathcal{ALCF}$. Furthermore, it can easily be verified that, if the original algorithm is started on an $\mathcal{ALCF}$ concept, no concrete domain operators or "concrete objects" are introduced during the algorithm run, and, hence, neither concrete domain related completion rules nor concrete domain related clash rules apply. Thus, they can savely be left away.

Proposition 3. *The* sat *algorithm is sound, complete, and terminates.*

We now investigate the extension of sat to TBoxes as described in Section 2.1. The extended algorithm is called tbsat and takes a pair $(A, \mathcal{T})$ as input, where A is an atomic concept and $\mathcal{T}$ is an $\mathcal{ALCF}$ TBox in simple form. tbsat is also capable of deciding satisfiability of non-atomic concepts w.r.t. TBoxes (see Section 2.1). The correctness of tbsat follows from the correctness of the original algorithm and the fact that a run of tbsat on $(A, \mathcal{T})$ is equivalent to a run of sat on C, where C is the result of unfolding A w.r.t. $\mathcal{T}$ (see Section 2.1). It remains to determine the runtime of the extended algorithm.

Proposition 4. *The algorithm* tbsat *can be executed in exponential time.*

Proof: Let $(A, \mathcal{T})$ be an input to tbsat. Let n denote $||\mathcal{T}||$. It needs to be shown that the number of rule applications performed by tbsat is exponential in n. This is a consequence of the next two claims, since each completion rule can be applied at most once per constraint (for the R$\forall$C rule, this holds for the $(a, b) : R$ constraints)

1. Let ρ be the number of objects created during a tbsat run. ρ is exponential in n.
2. For each objects a, there may exist at most exponentially many constraints which refer to a.

In the following, we can savely ignore constraints of the form $a \neq b$ since they do not appear in the premise of any completion rule.

The validity of claim 1 can be seen as follows: The recursion depth of tbsat is bounded by n since the recursion depth of sat is bounded by the role depth of its input (same argument as in the proof of Proposition 1). In each recursion step, at most n recursive calls are made. Hence, by (implicit) application of the Rr$\exists$C rule, at most $n^n = 2^{n*log(n)} \leq 2^{n^2}$ objects are generated. For each such object, the feature-complete function is called which may generate new objects by

application of the Rf∃C, R↓, and R↑ rules. feature-complete generates a structure which has the form of a tree in which some nodes may coincide. Outdegree and depth of this tree-like structure are bounded by n: The outdegree is bounded by the number of distinct features in $\mathcal{T}$ since there may be at most one successor per feature; the depth of the structure is bounded by n since in sat runs, its depth is bounded by the role depth (see again the argument in the proof of Proposition 1). Hence, the total number of objects generated is bounded by $2^{n^2} * 2^{n^2}$ which is obviously exponential in n.

Concerning point 2, fix an object a in a constraint system S considered by tbsat. It is easy to see that there may be at most n constraints of the form $a : C$—one for each concept definition in $\mathcal{T}$. Furthermore, there may be at most n constraints of the form $(a, a') : f$, since there cannot be more than one filler per feature (please note that constraints $(a, a') : R$ are never explicitly created). There may, however, be n constraints $(a', a) : f$ per object a'. Since the number of objects is exponentially bounded (point 1), the number of $(a', a) : f$ constraints is also exponentially bounded. □

Combining Propositions 2 and 4, we obtain the following result.

Theorem 2. *Deciding the satisfiability of $\mathcal{ALCF}$ concepts w.r.t. acyclic TBoxes is* NExpTime*-complete.*

5 Conclusion

TBoxes are an important component of knowledge representation systems using description logics. However, for most DLs, the exact complexity of reasoning with acyclic TBoxes has never been determinded. This paper concentrates on logics for which satisfiability w.r.t. the empty TBox is in PSpace and investigates how the presence of acyclic TBoxes influences the complexity of reasoning. In the first part of the paper, using the logic $\mathcal{ALC}$, it is demonstrated how completion algorithms for deciding "pure" concept satisfiability can be modified to take into account TBoxes such that the resulting algorithm can still be executed in polynomial space. Using the modified algorithm, it is proved that, for $\mathcal{ALC}$, satisfiability w.r.t. acyclic TBoxes is in PSpace. We claim that the given modification scheme can be applied to a variety of other description logics, too, and give a rule of thumb for when the resulting algorithm can be executed in polynomial space.

In the second part, it is proved that, for the logic $\mathcal{ALCF}$, satisfiability w.r.t. acyclic TBoxes is NExpTime-complete. In contrast, satisfiability of "pure" $\mathcal{ALCF}$ concepts is known to be PSpace-complete and the satisfiability of $\mathcal{ALCF}$ concepts w.r.t. general TBoxes is known to be undecidable. It is suprising that the complexity of reasoning moves up *several* steps in the complexity hierarchy if TBoxes are added. $\mathcal{ALCF}$ is a common description logic appearing as a fragment of several more expressive DLs such as, e.g., the temporal logic $\mathcal{TL}$-$\mathcal{ALCF}$ [1] or the logic $\mathcal{ALCF}(\mathcal{D})$ for reasoning with concrete domains [14]. Hence, satisfiability w.r.t. acyclic TBoxes is NExpTime-hard for these logics, too.

For the description logic $\mathcal{ALC}(\mathcal{D})$, similar complexity results as for $\mathcal{ALCF}$ can be obtained. The logic $\mathcal{ALC}(\mathcal{D})$ can be parameterized with a so-called concrete domain $\mathcal{D}$, and, hence, the complexity of reasoning with $\mathcal{ALC}(\mathcal{D})$ depends on the complexity of reasoning with the concrete domain $\mathcal{D}$. On the one hand, satisfiability of $\mathcal{ALC}(\mathcal{D})$ concepts w.r.t. the empty TBox is PSPACE-complete provided that reasoning with the concrete domain $\mathcal{D}$ is in PSPACE [14]. On the other hand, there exist concrete domains $\mathcal{D}$ for which reasoning is in NP such that satisfiability of $\mathcal{ALC}(\mathcal{D})$ concepts w.r.t. acyclic TBoxes is NEXPTIME-complete [13].

Acknowledgments I am indebted to Franz Baader who provided most of the ideas underlying Section 2.1. The work in this paper was supported by the "Foundations of Data Warehouse Quality" (DWQ) European ESPRIT IV Long Term Research (LTR) Project 22469.

References

1. A. Artale and E. Franconi. A temporal description logic for reasoning about actions and plans. *Journal of Artificial Intelligence Research (JAIR)*, (9), 1998.
2. F. Baader, H.-J. Bürckert, B. Nebel, W. Nutt, and G. Smolka. On the expressivity of feature logics with negation, functional uncertainty, and sort equations. *Journal of Logic, Language and Information*, 2:1–18, 1993.
3. F. Baader and P. Hanschke. A scheme for integrating concrete domains into concept languages. In *Proceedings of IJCAI-91*, pages 452–457, Sydney, Australia, August 24–30, 1991. Morgan Kaufmann Publ. Inc., San Mateo, CA, 1991.
4. F. Baader and B. Hollunder.set A terminological knowledge representation system with complete inference algorithms. In *Processings of PDK'91*, volume 567 of *LNAI*, pages 67–86, Kaiserslautern, Germany, July 1–3, 1991. Springer-Verlag, Berlin – Heidelberg – New York, 1991.
5. F. Baader, B. Hollunder, B. Nebel, H.-J. Profitlich, and E. Franconi. An empirical analysis of optimization techniques for terminological representation systems – or: Making KRIS get a move on. *Journal of Applied Intelligence*, 4:109–132, 1994.
6. E. Börger, E. Grädel, and Y. Gurevich. *The Classical Decision Problem*. Perspectives in Mathematical Logic. Springer-Verlag, Berlin, 1997.
7. D. Calvanese. Reasoning with inclusion axioms in description logics: Algorithms and complexity. In *Proceedings of ECAI'96, Budapest, Hungary*, pages 303–307, 1996.
8. D. Calvanese, G. De Giacomo, M. Lenzerini, and D. Nardi. Reasoning in expressive description logics. In *Handbook of Automated Reasoning*. Elsevier Science Publishers (North-Holland), Amsterdam, 1999. To appear.
9. F. M. Donini, M. Lenzerini, D. Nardi, and W. Nutt. The complexity of concept languages. *Information and Computation*, 134(1):1–58, 10 Apr. 1997.
10. F. M. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. Reasoning in description logics. In G. Brewka, editor, *Foundation of Knowledge Representation*, pages 191–236. CSLI-Publications, 1996.
11. B. Hollunder and W. Nutt. Subsumption algorithms for concept languages. DFKI Research Report RR-90-04, German Research Center for Artificial Intelligence, Kaiserslautern, 1990.

12. I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for expressive description logics. In *Proceedings of LPAR'99*, *LNCS*, Tbilisi, Georgia, 1999. Springer-Verlag, Berlin – Heidelberg – New York, 1999.
13. C. Lutz. On the complexity of terminological reasoning. LTCS-Report 99-04, LuFG Theoretical Computer Science, RWTH Aachen, Germany, 1999.
14. C. Lutz. Reasoning with concrete domains. In *Proceedings of IJCAI-99*, Stockholm, Sweden, July 31 – August 6, 1999. Morgan Kaufmann Publ. Inc., San Mateo, CA, 1999.
15. B. Nebel. Terminological reasoning is inherently intractable. *Artificial Intelligence*, 43:235–249, 1990.
16. B. Nebel. Terminological cycles: Semantics and computational properties. In J. F. Sowa, editor, *Principles of Semantic Networks – Explorations in the Representation of Knowledge*, chapter 11, pages 331–361. Morgan Kaufmann Publ. Inc., San Mateo, CA, 1991.
17. M. Schmidt-Schauß and G. Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26, 1991.
18. S. Tobies. A PSpace algorithm for graded modal logic. In *Proceedings of CADE-16, LNCS*, 1999. Springer-Verlag, Berlin – Heidelberg – New York, 1999.