NExpTime-complete Description Logics with Concrete Domains

Carsten Lutz

LuFG Theoretical Computer Science
RWTH Aachen, Germany
lutz@cs.rwth-aachen.de

Abstract. Concrete domains are an extension of Description Logics (DLs) allowing to integrate reasoning about conceptual knowledge with reasoning about "concrete properties" of objects such as sizes, weights, and durations. It is known that reasoning with $\mathcal{ALC}(\mathcal{D})$, the basic DL admitting concrete domains, is PSpace-complete. In this paper, it is shown that the upper bound is not robust: we give three examples for seemingly harmless extensions of $\mathcal{ALC}(\mathcal{D})$—namely acyclic TBoxes, inverse roles, and a role-forming concrete domain constructor—that make reasoning NExpTime-hard. As a corresponding upper bound, we show that reasoning with all three extensions *together* is in NExpTime.

1 Introduction

Description Logics (DLs) are a family of logical formalisms for the representation of and reasoning about conceptual knowledge. The knowledge is represented on an abstract logical level, i.e., by means of concepts (unary predicates), roles (binary predicates), and logical constructors. This makes it difficult to adequately represent knowledge concerning "concrete properties" of real-world entities such as their sizes, weights, and durations. Since, for many knowledge representation applications, it is essential to integrate reasoning about such concrete properties with reasoning about knowledge represented on an abstract logical level, Baader and Hanschke extended Description Logics by so-called *concrete domains* [1]. A concrete domain consists of a set called the domain and a set of predicates with a fixed interpretation over this domain. For example, one could use the real numbers as the domain and then define predicates such as the unary $=_{23}$, the binary "$<$" and "$=$", and the ternary "$+$" and "$\cdot$" [1]. Or one could use the set of all intervals over, say, the rationals as the domain and then define "temporal" predicates such as *during*, *meets*, and *before* [15]. Baader and Hanschke propose to extend the basic Description Logic $\mathcal{ALC}$ with concrete domains which yields the logic $\mathcal{ALC}(\mathcal{D})$. The interface between $\mathcal{ALC}$ and the concrete domain is provided by a concrete domain concept constructor. To illustrate the use of concrete domains for knowledge representation, consider the example $\mathcal{ALC}(\mathcal{D})$-concept

$$\forall \mathit{subprocess}.\mathit{Drilling} \sqcap \exists \mathit{workpiece}.(\exists \mathit{height}.{=_{5cm}} \sqcap \exists \mathit{height}, \mathit{length}.{>})$$

which describes a process all of whose subprocesses are drilling processes and which involves a workpiece with height 5cm and hight strictly greater than its length. Here, $=_{5cm}$ is a unary predicate from the concrete domain and $>$ is a binary predicate. The subconcept in brackets is a conjunction of two concrete domain concept constructors. Other DLs with concrete domains can be found in [3, 8, 12], while applications of such logics are described in [2, 8].

In this paper, we are interested in the complexity of reasoning with Description Logics providing for concrete domains. The complexity of $\mathcal{ALC}(\mathcal{D})$ itself is determined in [14], where reasoning with $\mathcal{ALC}(\mathcal{D})$ is proved to be PSpace-complete if reasoning with the concrete domain $\mathcal{D}$ is in PSpace. However, for many applications, the expressivity of $\mathcal{ALC}(\mathcal{D})$ is not sufficient which makes it quite natural to consider extensions of this logic with additional means of expressivity. We consider three such extensions—all of them frequently used in the area of Description Logics—and show that, although all these extensions are seemingly "harmless", reasoning in the extended logics is considerably harder than in $\mathcal{ALC}(\mathcal{D})$ itself. Hence, the PSpace upper bound of $\mathcal{ALC}(\mathcal{D})$ cannot be considered robust.

More precisely, we consider the extension of $\mathcal{ALC}(\mathcal{D})$ with (1) acyclic TBoxes, (2) inverse roles, and (3) a role-forming concrete domain constructor. TBoxes are used for representing terminological knowledge and background knowledge of application domains [5, 13], inverse roles are present in most expressive Description Logics [5, 10], and the role-forming constructor is a natural counterpart to the concept-forming concrete domain constructor [8]. By introducing a NExpTime-complete variant of the Post Correspondence Problem [17, 9], we identify a large class of concrete domains $\mathcal{D}$ such that reasoning with each of the above three extensions of $\mathcal{ALC}(\mathcal{D})$ (separately) is NExpTime-hard. This dramatic increase in complexity is rather surprising since, from a computational point of view, all of the proposed extensions look harmless. For example, in [13], it is shown that the extension of many PSpace Description Logics with acyclic TBoxes does not increase the complexity of reasoning. Moreover, it is well-known that the extension with inverse roles does usually not change the complexity class. For example, $\mathcal{ALC}$ extended with inverse roles is still in PSpace [11]. As a corresponding upper bound, we show that, if reasoning with a concrete domain $\mathcal{D}$ is in NP, then reasoning with $\mathcal{ALC}(\mathcal{D})$ and all three above extensions (simultaneously) is in NExpTime. We argue that this upper bound captures a large class of interesting concrete domains. This paper is accompanied by a technical report containing full proofs [16].

2 Description Logics with Concrete Domains

We introduce the Description Logics we are concerned with in the remainder of this paper. First, $\mathcal{ALCI}(\mathcal{D})$ is defined which extends $\mathcal{ALC}(\mathcal{D})$ with inverse roles. In a second step, we add a role-forming concrete domain constructor and obtain the logic $\mathcal{ALCRPI}(\mathcal{D})$. This two-step approach is pursued since the definition of

$\mathcal{ALCRPI}(\mathcal{D})$ involves some rather unusual syntactic restrictions which we like to keep separated from the more straightforward syntax of $\mathcal{ALCI}(\mathcal{D})$.

Definition 1 (Concrete Domain). *A* concrete domain *$\mathcal{D}$ is a pair $(\Delta_\mathcal{D}, \Phi_\mathcal{D})$, where $\Delta_\mathcal{D}$ is a set called the domain, and $\Phi_\mathcal{D}$ is a set of predicate names. Each predicate name $P \in \Phi_\mathcal{D}$ is associated with an arity n and an n-ary predicate $P^\mathcal{D} \subseteq \Delta_\mathcal{D}^n$.*

With $\overline{P}$, we denote the negation of the predicate P, i.e. $\overline{P}^\mathcal{D} = \Delta_\mathcal{D} \setminus P^\mathcal{D}$. Based on concrete domains, we introduce the syntax of $\mathcal{ALCI}(\mathcal{D})$.

Definition 2 (Syntax). *Let N_C, N_R, and N_{cF} be mutually disjoint sets of* concept names, role names, *and* concrete feature names, *respectively, and let N_{aF} be a subset of N_R. Elements of N_{aF} are called* abstract features. *The set of $\mathcal{ALCI}(\mathcal{D})$ roles $\widehat{N_R}$ is $N_R \cup \{R^- \mid R \in N_R\}$. An expression $f_1 \cdots f_n g$, where $f_1, \ldots, f_n \in N_{aF}$ ($n \geq 0$) and $g \in N_{cF}$, is called a* path. *The set of $\mathcal{ALCI}(\mathcal{D})$-concepts is the smallest set such that*

1. *every concept name is a concept*
2. *if C and D are concepts, R is a role, g is a concrete feature, $P \in \Phi$ is a predicate name with arity n, and $u_1, \ldots, u_n$ are paths, then the following expressions are also concepts: $\neg C$, $C \sqcap D$, $C \sqcup D$, $\exists R.C$, $\forall R.C$, $\exists u_1, \ldots, u_n.P$, and $g\uparrow$.*

An $\mathcal{ALCI}(\mathcal{D})$-concept which uses only roles from N_R is called an $\mathcal{ALC}(\mathcal{D})$-*concept.* With *sub*$(C)$, we denote the set of subconcepts of a concept C which is defined in the obvious way. Throughout this paper, we denote concept names with A and B, concepts with C and D, roles with R, abstract features with f, concrete features with g, paths with u, and predicates with P. As usual, we write $\top$ for $A \sqcup \neg A$, $\bot$ for $A \sqcap \neg A$ (where A is some concept name), and $\exists f_1 \cdots f_n.C$ (resp. $\forall f_1 \cdots f_n.C$) for $\exists f_1. \cdots \exists f_n.C$ (resp. $\forall f_1. \cdots \forall f_n.C$).

The syntactical part of a Description Logic is usually given by a concept language and a so-called TBox formalism. The TBox formalism is used to represent terminological knowledge of the application domain.

Definition 3 (TBoxes). *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 the left-hand sides of all concept definitions in $\mathcal{T}$ are unique, then $\mathcal{T}$ is called a* TBox.

TBoxes can be thought of as sets of macro definitions, i.e., the left-hand side of every concept definition is an abbreviation for the right-hand side of the concept definition. There also exist more general TBox formalisms allowing for arbitrary equations over concepts [5, 10]. However, we will see that admitting these general TBoxes makes reasoning with $\mathcal{ALC}(\mathcal{D})$ (and hence also $\mathcal{ALCI}(\mathcal{D})$) undecidable.

Definition 4 (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 path, then $u^\mathcal{I}(a)$ is defined as $g^\mathcal{I}(f_n^\mathcal{I} \cdots (f_1^\mathcal{I}(a)) \cdots)$. The interpretation function is extended to arbitrary roles and concepts as follows:*

$$\begin{aligned}
(R^-)^\mathcal{I} &:= \{(a,b) \mid (b,a) \in R^\mathcal{I}\} \\
(C \sqcap D)^\mathcal{I} &:= C^\mathcal{I} \cap D^\mathcal{I} \quad (C \sqcup D)^\mathcal{I} := C^\mathcal{I} \cup D^\mathcal{I} \quad (\neg C)^\mathcal{I} := \Delta_\mathcal{I} \setminus C^\mathcal{I} \\
(\exists R.C)^\mathcal{I} &:= \{a \in \Delta_\mathcal{I} \mid \{b \mid (a,b) \in R^\mathcal{I}\} \cap C^\mathcal{I} \neq \emptyset\} \\
(\forall R.C)^\mathcal{I} &:= \{a \in \Delta_\mathcal{I} \mid \{b \mid (a,b) \in R^\mathcal{I}\} \subseteq C^\mathcal{I}\} \\
(\exists u_1, \ldots, u_n.P)^\mathcal{I} &:= \{a \in \Delta_\mathcal{I} \mid (u_1^\mathcal{I}(a), \ldots, u_n^\mathcal{I}(a)) \in P^\mathcal{D}\} \\
(g\uparrow)^\mathcal{I} &:= \{a \in \Delta_\mathcal{I} \mid g^\mathcal{I}(a) \textit{ undefined}\}
\end{aligned}$$

An interpretation $\mathcal{I}$ is called a model *for a concept C iff $C^\mathcal{I} \neq \emptyset$ and a* model *for a TBox $\mathcal{T}$ iff $A^\mathcal{I} = C^\mathcal{I}$ for all $A \doteq C \in \mathcal{T}$.*

We call elements from $\Delta_\mathcal{I}$ *abstract objects* and elements from $\Delta_\mathcal{D}$ *concrete objects*. Our definition of $\mathcal{ALC}(\mathcal{D})$ differs slightly from the original version in [1]: 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 choose the separated approach since it allows clearer proofs. Moreover, it is not hard to see that the combined features can be "simulated" using pairs of concrete and abstract features.

Definition 5 (Inference Problems). *Let C and D be concepts. C* subsumes *D w.r.t. a TBox $\mathcal{T}$ (written $D \sqsubseteq_\mathcal{T} C$) iff $D^\mathcal{I} \subseteq C^\mathcal{I}$ for all models $\mathcal{I}$ of $\mathcal{T}$. C is* satisfiable *w.r.t. a TBox $\mathcal{T}$ iff there exists a model of both $\mathcal{T}$ and C.*

Both inferences are also considered without reference to TBoxes, i.e., with reference to the empty TBox. It is well-known that (un)satisfiability and subsumption can be mutually reduced to each other: $C \sqsubseteq_\mathcal{T} D$ iff $C \sqcap \neg D$ is unsatisfiable w.r.t. $\mathcal{T}$, and C is satisfiable w.r.t. $\mathcal{T}$ iff we do not have $C \sqsubseteq_\mathcal{T} \bot$. We call two concepts C and D *equivalent* iff C subsumes D and D subsumes C.

Let us now further extend $\mathcal{ALCI}(\mathcal{D})$ with a role-forming concrete domain constructor, i.e., with a constructor that allows the definition of complex roles with reference to the concrete domain. Such a constructor was first defined in [8], where it is motivated as an appropriate tool for spatial reasoning.

Definition 6 ($\mathcal{ALCRPI}(\mathcal{D})$ Syntax and Semantics). *A* predicate role *is an expression of the form $\exists(u_1, \ldots, u_n), (v_1, \ldots, v_m).P$ where P is an $n + m$-ary predicate. The semantics of predicate roles is*

$$\begin{aligned}
(\exists(u_1, \ldots, u_n), (v_1, \ldots, v_m).P)^\mathcal{I} : = \{&(a,b) \in \Delta_\mathcal{I} \times \Delta_\mathcal{I} \mid \\
&(u_1^\mathcal{I}(a), \ldots, u_n^\mathcal{I}(a), v_1^\mathcal{I}(b), \ldots, v_m^\mathcal{I}(b)) \in P^\mathcal{D}\}.
\end{aligned}$$

With $\mathcal{R}$, we denote the set of predicate roles. The set of $\mathcal{ALCRPI}(\mathcal{D})$ roles $\widehat{\mathcal{R}}$ is defined as $\widehat{N_R} \cup \mathcal{R} \cup \{R^- \mid R \in \mathcal{R}\}$. A role which is either a predicate role or the inverse of a predicate role is called complex role. *An $\mathcal{ALCI}(\mathcal{D})$-concept with roles from $N_R \setminus N_{aF}$ replaced with roles from $\widehat{\mathcal{R}}$ is called* $\mathcal{ALCRPI}(\mathcal{D})$-concept.

An $\mathcal{ALCRPI}(\mathcal{D})$-concept not using the inverse role constructor is called an $\mathcal{ALCRP}(\mathcal{D})$*-concept.* For example, the following is an $\mathcal{ALCRPI}(\mathcal{D})$-concept

$$Error \sqcap \exists time, next\ time.{<} \sqcap \forall next.\forall(\exists(time), (time).{<})^-.\neg Error$$

where *Error* is a concept, *time* is a concrete feature and *next* is an abstract feature. This concept is unsatisfiable since every domain object satisfying it would have to be both in *Error* and $\neg$*Error* which is impossible. In [7], it is proved that satisfiability of $\mathcal{ALCRP}(\mathcal{D})$-concepts is undecidable. However, as shown in [8], there exists a decidable fragment of $\mathcal{ALCRP}(\mathcal{D})$ that is still a useful extension of $\mathcal{ALC}(\mathcal{D})$. In the following, we introduce an analogous fragment of the logic $\mathcal{ALCRPI}(\mathcal{D})$. To do this, we fist need to define the negation normal form for concepts and describe how concepts can be converted into this form.

Definition 7 (NNF). *An $\mathcal{ALCRPI}(\mathcal{D})$-concept is said to be* in negation normal form (NNF) *if negation occurs in front of concept names, only. The following rewrite rules preserve equivalence. Exhaustive rule application yields a concept which is in NNF.*

$$\begin{array}{rll} \neg(C \sqcap D) \Longrightarrow \neg C \sqcup \neg D & \neg(C \sqcup D) \Longrightarrow \neg C \sqcap \neg D & \neg\neg C \Longrightarrow C \\ \neg(\exists R.C) \Longrightarrow (\forall R.\neg C) & \neg(\forall R.C) \Longrightarrow (\exists R.\neg C) & \\ \neg(\exists u_1,\ldots,u_n.P) \Longrightarrow \exists u_1,\ldots,u_n.\overline{P} \sqcup u_1\uparrow \sqcup \cdots \sqcup u_n\uparrow & & \\ \neg(g\uparrow) \Longrightarrow \exists g.\top_{\mathcal{D}} & & \end{array}$$

We may now define restricted concepts.

Definition 8 (Restricted $\mathcal{ALCRPI}(\mathcal{D})$-concept). *Let C be an $\mathcal{ALCRPI}(\mathcal{D})$-concept, and $sub(C)$ the set of subconcepts of C. Then C is called* restricted *iff the result C' of converting C to NNF satisfies the following conditions:*

1. *For any $\forall R.D \in sub(C')$, where R is a complex role, $sub(D)$ does not contain any concepts of the form $\exists u_1,\ldots,u_n.P$ or $\exists S.E$, where S is a complex role.*
2. *For any $\exists R.D \in sub(C')$, where R is a complex role, $sub(D)$ does not contain any concepts of the form $\exists u_1,\ldots,u_n.P$ or $\forall S.E$, where S is a complex role.*

Intuitively, these restrictions enforce the finite model property which leads to decidability, see [8, 16] for details. In the remainder of this paper, we assume all $\mathcal{ALCRPI}(\mathcal{D})$ concepts to be restricted without further notice. Note that the set of restricted $\mathcal{ALCRPI}(\mathcal{D})$-concepts is closed under negation, and, hence, subsumption can be reduced to satisfiability.

3 A NExpTime-complete Variant of the PCP

The Post Correspondence Problem (PCP), as introduced 1946 by Emil Post [17], is an undecidable problem frequently employed in undecidability proofs. In this section, we define a NExpTime-complete variant of the PCP together with a concrete domain $\mathcal{P}$ that is suitable for reducing PCPs to the satisfiability problem of Description Logics with concrete domains.

Definition 9 (PCP). *A* Post Correspondence Problem (PCP) *P is given by a finite, non-empty list $(\ell_1, r_1), \dots, (\ell_k, r_k)$ of pairs of non-empty words over some alphabet Σ. A sequence of integers $i_1, \dots, i_m$, with $m \geq 1$, is called a* solution *for P iff $\ell_{i_1} \cdots \ell_{i_m} = r_{i_1} \cdots r_{i_m}$. Let $f(n)$ be a mapping from $\mathbb{N}$ to $\mathbb{N}$ and let $|P|$ denote the sum of the lengths of all words in the PCP P. A solution $i_1, \dots, i_m$ for P is called an* $f(n)$-solution *iff $m \leq f(|P|)$. With* $f(n)$-PCP*, we denote the version of the PCP that admits only $f(n)$-solutions.*

Analogous to the undecidability result for the general PCP given by Hopcroft and Ullman in [9], we may prove the following result.

Theorem 1. *It is* NExpTime*-complete to decide whether a $2^n + 1$-PCP has a solution.*

Hence, a reduction of the $2^n + 1$-PCP is a candidate for proving NExpTime lower bounds for Description Logics with concrete domains. As we will see now, the problem is in fact well-suited for this task since it is possible to define an appropriate concrete domain. It follows from the proof of the above theorem that it is sufficient to consider some fixed, finite alphabet Σ_U whose cardinality is the number of symbols needed to define a universal Turing machine.

Definition 10 (Concrete Domain $\mathcal{P}$). *The concrete domain $\mathcal{P}$ is defined by setting $\Delta_\mathcal{P} := \Sigma_U^*$ and defining $\Phi_\mathcal{P}$ as the smallest set containing the following predicates:*

- *unary predicates word and nword with $word^\mathcal{P} = \Delta_\mathcal{P}$ and $nword^\mathcal{P} = \emptyset$,*
- *unary predicates $=_\epsilon$ and $\neq_\epsilon$ with $=_\epsilon^\mathcal{P} = \{\epsilon\}$ and $\neq_\epsilon^\mathcal{P} = \Sigma_U^+$,*
- *a binary equality predicate $=$ and a binary inequality predicate $\neq$, and*
- *for each $w \in \Sigma_U^+$, two binary predicates $conc_w$ and $nconc_w$ with*
$$conc_w^\mathcal{P} = \{(u,v) \mid v = uw\} \text{ and } nconc_w^\mathcal{P} = \{(u,v) \mid v \neq uw\}.$$

The complexity of reasoning with a Description Logic providing a concrete domain $\mathcal{D}$ does obviously depend on the complexity of reasoning with $\mathcal{D}$. More precisely, most satisfiability algorithms involve checking the satisfiability of finite conjunctions of concrete domain predicates

$$\bigwedge_{1 \leq i \leq k} (x_0^{(i)}, \dots, x_{n_i}^{(i)}) : P_i,$$

where each P_i is an n_i-ary predicate and the $x_j^{(i)}$ are variables from some fixed set [1]. This is also the case for the tableau algorithm that used to prove the upper bound in Section 7. Hence, we are interested in the complexity of this task which is called *$\mathcal{D}$-satisfiability* in what follows. By devising an algorithm that is based on repeated normalization combined with tests for obvious inconsistencies, the following result can be obtained.

Proposition 1. *$\mathcal{P}$-satisfiability is decidable in deterministic polynomial time.*

On first sight, the concrete domain $\mathcal{P}$ may look somewhat unnatural in the context of knowledge representation. However, it is straightforward to encode words as natural numbers and to define the operations on words as rather simple operations on the naturals [2]: Words over the alphabet Σ_U can be interpreted as numbers written at base $|\Sigma_U| + 1$ (assuming that the empty word represents 0); the concatenation of two words v and w can then be expressed as $vw = v * (|\Sigma_U| + 1)^{|w|} + w$, where $|w|$ denotes the length of the word w. Hence, each concrete domain (Δ, Φ), where Δ contains the natural numbers and Φ contains predicates for (in)equality, (in)equality to zero, addition, and multiplication may also be used for the reductions. A concrete domain with these properties is called *arithmetic.*

4 Satisfiability of $\mathcal{ALC}(\mathcal{P})$-concepts w.r.t. TBoxes

In this section, we show that the satisfiability of $\mathcal{ALC}(\mathcal{P})$-concepts w.r.t. TBoxes is NExpTime-hard. As already mentioned, this result is rather surprising since (1) satisfiability of $\mathcal{ALC}(\mathcal{D})$-concepts without reference to TBoxes is known to be PSpace-complete if reasoning with the concrete domain $\mathcal{D}$ is in PSpace [14], and (2) admitting acyclic TBoxes does "usually" not increase the complexity of reasoning [13].

The proof is by a reduction of the $2^n + 1$-PCP using the concrete domain $\mathcal{P}$ introduced in the previous section. Given a $2^n + 1$-PCP $P = (\ell_1, r_1), \ldots, (\ell_k, r_k)$, we define a TBox $\mathcal{T}_P$ of size polynomial in $|P|$ and a concept (name) C_P such that C_P is satisfiable w.r.t. $\mathcal{T}_P$ iff P has a solution. Figure 1 contains the reduction TBox and Figure 2 an example model for $|P| = 2$. In the figures, ℓ, r, x, and y denote abstract features and g_ℓ and g_r denote concrete features. The first equality in Figure 1 is not a concept definition but an abbreviation: Replace every occurrence of $Ch[u_1, u_2, u_3, u_4]$ in the lower three concept definitions by the right-hand side of the first identity substituting $u_1, \ldots, u_4$ appropriately.

The idea behind the reduction is to define $\mathcal{T}_P$ such that models of C_P and $\mathcal{T}_P$ have the form of a binary tree of depth $|P|$ whose leaves are connected by two "chains" of $conc_w$ predicates. Pairs of corresponding objects (x_i, y_i) on the chains represent partial solutions of the PCP P. More precisely, the first line of the definitions of the $C_0, \ldots, C_{n-1}$ concepts ensures that models have the form of a binary tree of depth n (with $n = |P|$) whose left edges are labeled with the abstract feature ℓ and whose right edges are labeled with the abstract feature r. Let the abstract objects $a_{n,0}, \ldots a_{n,2^n-1}$ be the leaves of this tree. By the second line of the definitions of the $C_0, \ldots, C_{n-1}$ concepts, every $a_{n,i}$ has a g_ℓ-successor x_i and a g_r-successor y_i. These second lines also ensure that the

$$
\begin{aligned}
\mathit{Ch}[u_1,u_2,u_3,u_4] &= (\exists(u_1,u_2).= \;\sqcap\; \exists(u_3,u_4).=) \\
&\quad \sqcup \bigsqcup_{(\ell_i,r_i)\text{ in }P} (\exists(u_1,u_2).\mathit{conc}_{\ell_i} \sqcap \exists(u_3,u_4).\mathit{conc}_{r_i}) \\
C_0 &\doteq \exists\ell.C_1 \sqcap \exists r.C_1 \\
&\quad \sqcap\; \mathit{Ch}[\ell r^{n-1}g_\ell, r\ell^{n-1}g_\ell, \ell r^{n-1}g_r, r\ell^{n-1}g_r] \\
&\quad\vdots \\
C_{n-2} &\doteq \exists\ell.C_{n-1} \sqcap \exists r.C_{n-1} \\
&\quad \sqcap\; \mathit{Ch}[\ell r g_\ell, r\ell g_\ell, \ell r g_r, r\ell g_r] \\
C_{n-1} &\doteq \mathit{Ch}[\ell g_\ell, r g_\ell, \ell g_r, r g_r] \\
C_P &\doteq C_0 \\
&\quad \sqcap\; \exists\ell^n g_\ell.=_\epsilon \;\sqcap\; \exists\ell^n g_r.=_\epsilon \\
&\quad \sqcap\; \exists r^n y.\exists g_\ell, g_r.= \;\sqcap\; \exists r^n y g_\ell.\neq_\epsilon \\
&\quad \sqcap\; \mathit{Ch}[r^n g_\ell, r^n x g_\ell, r^n g_r, r^n x g_r] \\
&\quad \sqcap\; \mathit{Ch}[r^n x g_\ell, r^n y g_\ell, r^n x g_r, r^n y g_r]
\end{aligned}
$$

Fig. 1. The $\mathcal{ALC}(\mathcal{P})$ reduction TBox $\mathcal{T}_P$ ($n = |P|$).

x_i and y_i objects are connected via two predicate chains, where the predicates on the chains are either equality or conc_w. More precisely, for $0 \leq i < 2^n - 1$, either $x_i = x_{i+1}$ and $y_i = y_{i+1}$, or there exists a $j \in \{1,\ldots,k\}$ such that $(x_i, x_{i+1}) \in \mathit{conc}_{\ell_j}^{\mathcal{P}}$ and $(y_i, y_{i+1}) \in \mathit{conc}_{r_j}^{\mathcal{P}}$. Furthermore, by the second line of the definition of C_P, we have $x_1 = y_1 = \epsilon$. Hence, pairs (x_i, y_i) are partial solutions for P. Since we must consider solutions of a length up to $2^n + 1$, the 2^n objects on the fringe of the tree with their $2^n - 1$ connecting predicate edges are not sufficient, and we need to "add" two more objects $a_{n,2^n}$ and $a_{n,2^n+1}$ which behave analogously to the objects $a_{n,0}, \ldots a_{n,2^n-1}$. This is done by the last two lines of the definition of C_P. Finally, the third line of the definition of C_P ensures that $x_{2^n+1} = y_{2^n+1} \neq \epsilon$ and hence that (x_{2^n+1}, y_{2^n+1}) is in fact a full solution.

Obviously, the size of $\mathcal{T}_P$ is polynomial in $|P|$ and $\mathcal{T}_P$ can be constructed in time polynomial in $|P|$ which, together with the fact that $\mathcal{P}$ may be replaced by any arithmetic concrete domain, yields the following theorem.

Theorem 2. *For every arithmetic concrete domain $\mathcal{D}$, satisfiability of $\mathcal{ALC}(\mathcal{D})$-concepts w.r.t TBoxes is* NExpTime*-hard.*

We also obtain a lower bound for subsumption since satisfiability can be reduced to subsumption. With some slight modifications, the reduction just presented can also be applied to the Description Logic $\mathcal{ALCR}(\mathcal{P})$, i.e., $\mathcal{ALC}(\mathcal{P})$ enriched with a role conjunction constructor [6]. Hence, reasoning with this logic is also NExpTime-hard. The corresponding reduction concept can be found in [16].

One may ask why we are interested in the relatively weak acyclic TBoxes instead of using a more general TBox formalism. The answer is that using general TBoxes leads to undecidability.

Definition 11 (General TBox). *A* general concept inclusion (GCI) *has the form $C \sqsubseteq D$, where both C and D are concepts. An interpretation $\mathcal{I}$ is a* model

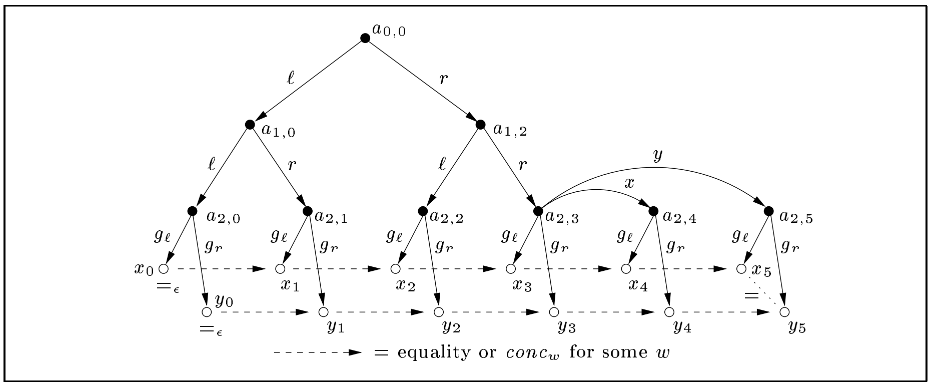

Fig. 2. An example model of C_P and $\mathcal{T}_P$ for $n = 2$.

for a GCI $C \sqsubseteq D$ iff $C^\mathcal{I} \subseteq D^\mathcal{I}$. Finite sets of GCIs are called general TBoxes. *An interpretation $\mathcal{I}$ is a* model *for a general TBox $\mathcal{T}$ iff $\mathcal{I}$ is a model for all GCIs in $\mathcal{T}$.*

Using the concrete domain $\mathcal{P}$ and a reduction of the general PCP, the following theorem can be obtained.

Theorem 3. *For every arithmetic concrete domain $\mathcal{D}$, satisfiability of $\mathcal{ALC}(\mathcal{D})$-concepts w.r.t. general TBoxes is undecidable.*

Proof Let P be an instance of the PCP. Define a concept C_P and a general TBox $\mathcal{T}_P$ as follows:

$$\begin{aligned}
C_P &:= \exists g.=_\epsilon \sqcap \exists fg.=_\epsilon \\
\mathcal{T}_P &:= \{\exists f.\top \sqsubseteq \bigsqcap_{(\ell_i,r_i)\in P} \exists g, f_i g.conc_{\ell_i} \sqcap \exists fg, f_i fg.conc_{r_i} \\
&\qquad \top \sqsubseteq \exists g.=_\epsilon \sqcup \neg\exists g, fg.=\}
\end{aligned}$$

An example model of C_P w.r.t. $\mathcal{T}_P$ can be found in Figure 3. The first GCI ensures that models of C_P and $\mathcal{T}_P$ represent *all* possible solutions of the PCP P. Additionally, the last GCI ensures that no potential solution is a solution. It is hence straightforward to prove that C_P is satisfiable w.r.t. $\mathcal{T}_P$ iff P has no solution, i.e., we have reduced the general, undecidable PCP [17, 9] to the satisfiability of $\mathcal{ALC}(\mathcal{D})$-concepts w.r.t. general TBoxes. ❏

5 Satisfiability of $\mathcal{ALCI}(\mathcal{P})$-Concepts

We now show that satisfiability of $\mathcal{ALCI}(\mathcal{P})$-concepts—without reference to TBoxes—is NExpTime-hard. As in the previous section, it is surprising that a rather small change in the logic, i.e., adding inverse roles, causes a dramatic increase in complexity.

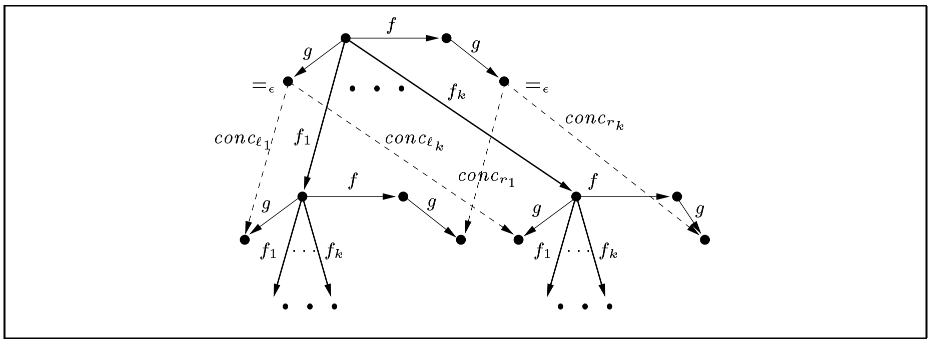

Fig. 3. An example model of C_P w.r.t. $\mathcal{T}_P$

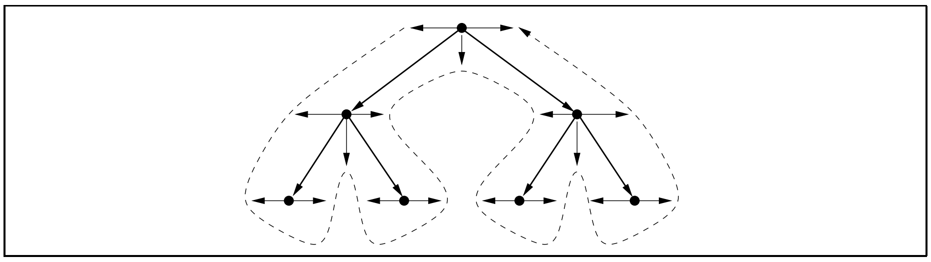

Fig. 4. Predicate chains in models of C_P.

The reduction is similar to the one used in the previous section: it is a reduction of the $2^n + 1$-PCP and uses the concrete domain $\mathcal{P}$. However, we need a slightly different strategy since, in the case of inverse roles, it is not possible to enforce chains of predicates connecting the leaves of the tree. Instead, the predicate chains emulate the structure of the tree following the scheme indicated in Figure 4. Given a PCP $P = (\ell_1, r_1), \ldots, (\ell_k, r_k)$, we define a concept C_P of size polynomial in $|P|$ which has a model iff P has a solution. The concept C_P can be found in Figure 5. In the figure, $h_\ell, h_r, x_\ell, x_r, y_\ell, y_r, z_\ell,$ and z_r are concrete features. Note that the equalities are not concept definitions but abbreviations. As in the previous section, replace every occurrence of $Ch[u_1, u_2, u_3, u_4]$ in the lower three concept definitions by the right-hand side of the first identity substituting $u_1, \ldots, u_4$ appropriately and similarly for every occurrence of X.

Let us discuss the structure of models of C_P. Due to the first line in the definition of C_P and the $\exists f^-$ quantifiers in the definition of X, models of C_P have the form of a tree of depth $|P|-1$ in which all edges are labeled with f^-. This edge labelling scheme is possible since the inverse of an abstract feature is not a feature. Additionally, we establish two chains of concrete domain predicates as indicated in Figure 4. Again, corresponding objects on the two chains represent partial solutions of the PCP P. A more detailed clipping from a model of C_P can be found in Figure 6. The existence of the chains is ensured by the definition

$$
\begin{aligned}
\mathit{Ch}[u_1,u_2,u_3,u_4] = {} & (\exists(u_1,u_2).= \sqcap \exists(u_3,u_4).=)\\
& \sqcup \bigsqcup_{(\ell_i,r_i)\text{ in }P} \exists(u_1,u_2).\mathit{conc}_{\ell_i} \sqcap \exists(u_3,u_4).\mathit{conc}_{r_i}
\end{aligned}
$$

$$
\begin{aligned}
X = {} & \exists f^-.(\mathit{Ch}[fg_\ell,g_\ell,fg_r,g_r] \sqcap \exists(h_\ell,fp_\ell).= \sqcap \exists(h_r,fp_r).=)\\
& \sqcap \exists f^-.(\mathit{Ch}[fp_\ell,g_\ell,fp_r,g_r] \sqcap \exists(h_\ell,fh_\ell).= \sqcap \exists(h_r,fh_r).=)
\end{aligned}
$$

$$
\begin{aligned}
C_P = {} & X \sqcap \forall f^-.X \sqcap \cdots \sqcap \forall(f^-)^{n-1}.X\\
& \sqcap \forall(f^-)^n.(\exists(g_\ell,h_\ell).= \sqcap \exists(g_r,h_r).=)\\
& \sqcap\ \mathit{Ch}[h_\ell,x_\ell,h_r,x_r]\\
& \sqcap\ \mathit{Ch}[x_\ell,y_\ell,x_r,y_r]\\
& \sqcap\ \mathit{Ch}[y_\ell,z_\ell,y_r,z_r]\\
& \sqcap\ \exists g_\ell,=_\epsilon\ \sqcap \exists g_r,=_\epsilon\\
& \sqcap\ \exists z_\ell,z_r.= \sqcap \exists z_\ell.\neq_\epsilon
\end{aligned}
$$

Fig. 5. The $\mathcal{ALCI}(\mathcal{P})$ reduction concept C_P $(n = |P| - 1)$.

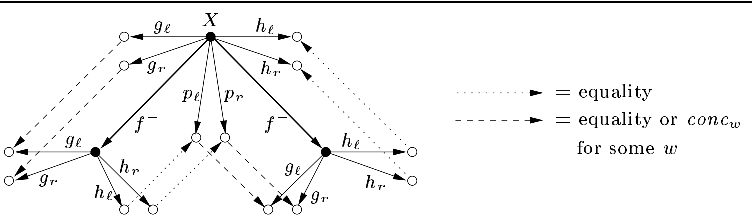

Fig. 6. A clipping from a model of C_P.

of X and the second line in the definition of C_P: The concept X establishes the edges of the predicate chains as depicted in Figure 6 (in fact, Figure 6 is a model of the concept X) while the second line of C_P establishes the edges "leading around" the leaves. Edges of the latter type and the dotted edges in Figure 6 are labeled with the equality predicate. To see why this is the case, let us investigate the length of the chains.

The length of the two predicate chains is twice the number of edges in the tree plus the number of leaves, i.e., $2 * (2^{|P|} - 2) + 2^{|P|-1}$. To eliminate the factor 2 and the summand $2^{|P|-1}$, C_P is defined such that every edge in the predicate chains leading "up" in the tree and every edge "leading around" a leaf is labeled with the equality predicate. To extend the chains to length $2^{|P|} + 1$, we need to add three additional edges (definition of C_P, lines three, four, and five). Finally, the last two lines in the definition of C_P ensure that the first concrete object on both chains represents the empty word and that the last objects on the chains represent a (non-empty) solution for P.

Theorem 4. *For every arithmetic concrete domain $\mathcal{D}$, satisfiability of $\mathcal{ALCI}(\mathcal{D})$-concepts is* NExpTime*-hard.*

$$
\begin{aligned}
\mathit{DistB}[k] &= \bigsqcap_{i=0}^{k}((B_i \rightarrow \forall R.B_i) \sqcap \neg B_i \rightarrow \forall R.\neg B_i) \\
\mathit{Tree} &= \exists R.B_0 \sqcap \exists R.\neg B_0 \\
&\quad \sqcap \forall R.(\mathit{DistB}[0] \sqcap \exists R.B_1 \sqcap \exists R.\neg B_1) \\
&\qquad \vdots \\
&\quad \sqcap \forall R^{n-1}.(\mathit{DistB}[n-1] \sqcap \exists R.B_{n-1} \sqcap \exists R.\neg B_{n-1}) \\
S[g,p] &= \exists(g),(g).\overline{p} \\
\mathit{Edge}[g,p] &= \Bigg(\bigsqcup_{k=0}^{n-1}\Big(\bigsqcup_{j=0}^{k-1}\neg B_j\Big) \sqcap (B_k \rightarrow \forall S[g,p].\neg B_k) \sqcap (\neg B_k \rightarrow \forall S[g,p].B_k) \\
&\quad \sqcup \bigsqcup_{k=0}^{n-1}\Big(\bigsqcap_{j=0}^{k-1} B_j\Big) \sqcap (B_k \rightarrow \forall S[g,p].B_k) \sqcap (\neg B_k \rightarrow \forall S[g,p].\neg B_k)\Bigg) \\
\mathit{DEdge} &= (\mathit{Edge}[g_\ell, =] \sqcap \mathit{Edge}[g_r, =]) \sqcup \\
&\quad \bigsqcup_{(\ell_i, r_i) \text{ in } P}(\mathit{Edge}[g_\ell, \mathit{conc}_{\ell_i}] \sqcap \mathit{Edge}[g_r, \mathit{conc}_{r_i}]) \\
\mathit{Ch}[u_1,u_2,u_3,u_4] &= (\exists(u_1,u_2).= \ \sqcap \exists(u_3,u_4).=) \\
&\quad \sqcup \bigsqcup_{(\ell_i, r_i) \text{ in } P}(\exists(u_1,u_2).\mathit{conc}_{\ell_i} \sqcap \exists(u_3,u_4).\mathit{conc}_{r_i}) \\
C_P &= \mathit{Tree} \sqcap \forall R^n.\exists g_\ell.\mathit{word} \sqcap \forall R^n.\exists g_r.\mathit{word} \\
&\quad \sqcap \forall R^n.\big[(\neg B_0 \sqcap \cdots \sqcap \neg B_{n-1}) \rightarrow (\exists g_\ell.=_\epsilon \ \sqcap \exists g_r.=_\epsilon) \\
&\qquad \sqcap \neg(B_0 \sqcap \cdots \sqcap B_{n-1}) \rightarrow \mathit{DEdge} \\
&\qquad \sqcap (B_0 \sqcap \cdots \sqcap B_{n-1}) \rightarrow \\
&\qquad\qquad \big(\mathit{Ch}(g_\ell, xg_\ell, g_r, xg_r) \sqcap \mathit{Ch}(xg_\ell, yg_\ell, xg_r, yg_r)\big)\big]
\end{aligned}
$$

Fig. 7. The $\mathcal{ALCRP}(\mathcal{P})$ reduction concept C_P $(n = |P|)$.

6 Satisfiability of $\mathcal{ALCRP}(\mathcal{P})$-Concepts

In this section, we prove that satisfiability of $\mathcal{ALCRP}(\mathcal{P})$-concepts without reference to TBoxes is NExpTime-hard. Hence, adding the role-forming concrete domain constructor yields another extension of $\mathcal{ALC}(\mathcal{D})$ in which reasoning is much harder than in $\mathcal{ALC}(\mathcal{D})$ itself.

Given a PCP $P = (\ell_1, r_1), \ldots, (\ell_k, r_k)$, we define a concept C_P of size polynomial in $|P|$ which has a model iff P has a solution. The concept C_P can be found in Figure 7, where x and y denote abstract features and p denotes a predicate (written in lowercase to avoid confusion with the PCP P). Again, the equalities in the figure serve as abbreviations. Moreover, we use $C \rightarrow D$ as an abbreviation for $\neg C \sqcup D$. Note that $S[g,p]$ denotes a predicate role and not a concept, i.e., $S[g,p]$ is an abbreviation for the role-forming concrete domain constructor $\exists(g),(g).\overline{p}$.

Figure 8 contains an example model of C_P with $|P| = n = 2$. Obviously, the models of C_P are rather similar to the ones from the $\mathcal{ALC}(\mathcal{D})$ reduction in Section 4: models have the form of a binary tree of depth n whose edges are

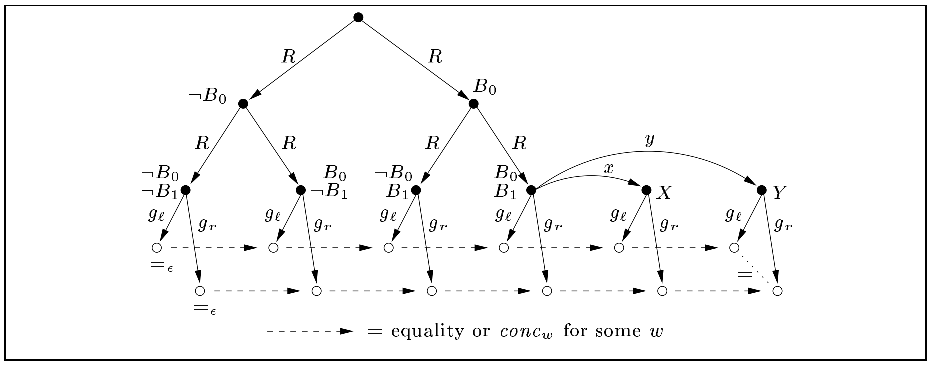

Fig. 8. An example model of C_P with $|P| = 2$.

labelled with the role R and whose leaves (together with two "extra" nodes) are connected by two predicate chains of length $2^n + 1$. The *Tree* concept enforces the existence of the binary tree. The concept names $B_0, \ldots, B_{n-1}$ are used for a binary numbering (from 0 to $2^n - 1$) of the leaves of the tree. More precisely, for a domain object $a \in \Delta^{\mathcal{I}}$, set

$$pos(a) = \Sigma_{i=0}^{n-1} \beta_i(a) * 2^i \quad \text{where} \quad \beta_i(a) = \begin{cases} 1 \text{ if } a \in B_i^{\mathcal{I}} \\ 0 \text{ otherwise.} \end{cases}$$

The *Tree* and *DistB* concepts ensure that, if two leaves a and a' are reachable via different paths from the root node, then we have $pos(a) \neq pos(a')$. Due to the first line of the C_P concept, every leaf has (concrete) g_ℓ- and g_r-successors. The last two lines of C_P guarantee the existence of the two extra nodes which are connected by predicate edges due to the use of the *Ch* concepts. Hence, it remains to describe how the edges between the leaf nodes are established.

There are two main ideas underlying the establishment of these edges: (i) use the role-forming predicate constructor to establish single edges and (ii) use the position *pos*() of leaf nodes together with the fact that counting modulo 2^n can be expressed by $\mathcal{ALC}$-concepts to do this with a concept of size polynomial in $|P|$. We first illustrate Point (i) in an abstract way. Assume that we have two abstract objects a and b, a has g_ℓ-successor x and b has g_ℓ-successor y. Moreover, let $b \in X^{\mathcal{I}}$ for some concept X. We may then establish a p-edge (for some binary predicate $p \in \Phi_{\mathcal{P}}$) between x and y as follows: we enforce that $a \in (\forall S[g_\ell, p].\neg X)^{\mathcal{I}}$; since $b \in X^{\mathcal{I}}$, it follows that $(a, b) \notin S[g_\ell, p]^{\mathcal{I}}$, i.e., $(a, b) \notin (\exists (g_\ell), (g_\ell).\overline{p})^{\mathcal{I}}$ and thus $(x, y) \notin \overline{p}^{\mathcal{P}}$, which obviously implies that $(x, y) \in p^{\mathcal{P}}$.

In the third line of the C_P-concept, the *DEdge* concept is used to establish edges between the leaf nodes. The *DEdge* concept itself is just a disjunction over the various edge types while the *Edge* concept actually establishes the edges. In principle, the *Edge* concept establishes the edges as described above. However, it does this not only for two fixed nodes as in the description above but for *all* neighboring leaf nodes. To see how this is achieved, note that *Edge* is essentially the negation of the well-known propositional formula

$$\bigwedge_{k=0}^{n-1} (\bigwedge_{j=0}^{k-1} x_j = 1) \to (x_k = 1 \leftrightarrow x'_k = 0) \ \wedge \ \bigwedge_{k=0}^{n-1} (\bigvee_{j=0}^{k-1} x_j = 0) \to (x_k = x'_k)$$

which encodes incrementation modulo 2^n, i.e., if t is the number (binarly) encoded by the propositional variables $x_0, \ldots, x_{n-1}$ and t' is the number encoded by the propositional variables $x'_0, \ldots, x'_{n-1}$, then we have $t' = t + 1$ modulo 2^n, c.f. [4]. Assume $a \in (\mathit{Edge}[g_\ell, p])^\mathcal{I}$ (where p is either "=" or conc_{ℓ_i}) and let b be the leaf with $\mathit{pos}(b) = \mathit{pos}(a) + 1$, x be the g_ℓ-successor of a, and y be the g_ℓ-successor of b. The *Edge* concept ensures that, for each $S[g_\ell, p]$-successor c of a, we have $\mathit{pos}(c) \neq \mathit{pos}(a) + 1$, i.e., there exists an i with $0 \leq i \leq n$ such that c differs from b in the interpretation of B_i. It follows that $(a, b) \notin S[g_\ell, p]^\mathcal{I}$. As described above, we can conclude $(x, y) \in p^\mathcal{I}$. All remaining issues such as, e.g., ensuring that one of the partial solutions is in fact a solution, are as in the reduction given in Section 4. Note that the reduction concept is restricted in the sense of Section 2.

Theorem 5. *For every arithmetic concrete domain $\mathcal{D}$, satisfiability of $\mathcal{ALCRP}(\mathcal{D})$-concepts is* NExpTime*-hard.*

7 Upper Bounds

Due to space limitations, we can only give a short sketch of the proof of the upper bound and refer to [16] for details. First, a tableau algorithm for deciding the satisfiability of $\mathcal{ALCRPI}(\mathcal{D})$-concepts without reference to TBoxes is devised. This algorithm combines techniques from [8] for reasoning with $\mathcal{ALCRP}(\mathcal{D})$ with techniques from [10] for reasoning with inverse role. Second, the tableau algorithm is modified to take into account TBoxes by performing "on the fly unfolding" of the TBox as described in [13]. A complexity analysis yields the following theorem.

Theorem 6. *If $\mathcal{D}$-satisfiability is in* NP*, satisfiability of $\mathcal{ALCRPI}(\mathcal{D})$-concepts w.r.t. TBoxes can be decided in nondeterministic exponential time.*

This also gives an upper bound for subsumption since, as mentioned in Section 2, subsumption can be reduced to satisfiability. It should be noted that the above theorem only applies to so-called *admissible* concrete domains, where a concrete domain $\mathcal{D}$ is admissible if the set $\Phi_\mathcal{D}$ if closed under negation and contains a predicate name $\top_\mathcal{D}$ for $\Delta_\mathcal{D}$ [16]. Nevertheless, the given theorem captures a large class of interesting concrete domains such as $\mathcal{P}$ itself and concrete domains for temporal and spatial reasoning [8, 15]. In contrast to the upper bound for $\mathcal{ALC}(\mathcal{D})$ established in [14], the above theorem if concerned with concrete domains for which $\mathcal{D}$-satisfiability is in NP instead of in PSpace. For concrete domains of this latter type, the tableau algorithm in [16] yields an ExpSpace upper bound. A matching lower bound, however, is yet to be proved.

8 Related and Future Work

We demonstrated that the PSpace upper bound for $\mathcal{ALC}(\mathcal{D})$-concept satisfiability is not robust: complexity shifts to NExpTime if seemingly harmless constructors are added and is even undecidable if we admit general TBoxes. However, the situation is not hopeless in all cases. Although the class of arithmetic concrete domains is quite large and captures many interesting concrete domains, there still exist non-trivial concrete domains for which reasoning with general TBoxes is decidable and the NExpTime lower bound obtained in this paper do presumably not hold. An example is presented in [15], where a temporal Description Logic based on concrete domains is defined.

As future work, it would be interesting to extend the obtained logics by additional means of expressivity such as transitive roles and qualifying number restrictions [11]. There are at least two ways to go: In [14] it is proved that reasoning with $\mathcal{ALCF}(\mathcal{D})$, i.e., the extension of $\mathcal{ALC}(\mathcal{D})$ with feature agreements and disagreements, is PSpace-complete (if reasoning with $\mathcal{D}$ is in PSpace). Hence, one could define extensions of $\mathcal{ALCF}(\mathcal{D})$ trying to obtain an expressive logic for which reasoning is still in PSpace. The second approach is to define extensions of $\mathcal{ALCI}(\mathcal{D})$ which means that the obtained logics are at least NExpTime-hard. Moreover, feature (dis)agreements—which are very closely related to concrete domains—cannot be considered since, in [16], we prove that the combination of inverse roles and feature (dis)agreements leads to undecidability.

Acknowledgements. My thanks go to Franz Baader, Ulrike Sattler, and Stephan Tobies for inspiring discussions. The work in this paper was supported by the DFG Project BA1122/3-1 “Combinations of Modal and Description Logics”.

References

1. F. Baader and P. Hanschke. A scheme for integrating concrete domains into concept languages. In *Proc. of IJCAI-91*, pp. 452–457, Sydney, Australia, 1991. Morgan Kaufmann Publ., 1991.
2. F. Baader and P. Hanschke. Extensions of concept languages for a mechanical engineering application. In *Proc. of GWAI-92*, volume 671 of LNCS, pp. 132–143, Bonn (Germany), 1993. Springer–Verlag.
3. F. Baader and U. Sattler. Description logics with concrete domains and aggregation. In *Proc. of ECAI-98*, Brighton, August 23–28, 1998. John Wiley & Sons, New York, 1998.
4. E. Börger, E. Grädel, and Y. Gurevich. *The Classical Decision Problem.* Perspectives in Mathematical Logic. Springer-Verlag, Berlin, 1997.
5. 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, 1998.
6. F. Donini, M. Lenzerini, D. Nardi, and W. Nutt. The complexity of concept languages. DFKI Research Report RR-95-07, German Research Center for Artificial Intelligence, Kaiserslautern, 1995.
7. V. Haarslev, C. Lutz, and R. Möller. Defined topological relations in description logics. In *Proc. of KR’98*, pp. 112–124, Trento, Italy, 1998. Morgan Kaufmann Publ., 1998.

8. V. Haarslev, C. Lutz, and R. Möller. A description logic with concrete domains and role-forming predicates. *Journal of Logic and Computation*, 9(3), 1999.
9. J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, Reading, Mass., 1979.
10. I. Horrocks and U. Sattler. A description logic with transitive and inverse roles and role hierarchies. *Journal of Logic and Computation*, 9(3):385–410, 1999.
11. I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for expressive description logics. In *Proc. of LPAR'99*, number 1705 in LNAI, pp. 161–180. Springer-Verlag, 1999.
12. G. Kamp and H. Wache. CTL - a description logic with expressive concrete domains. Technical Report LKI-M-96/01, Labor für Künstliche Intelligenz, Universität Hamburg, Germany, 1996.
13. C. Lutz. Complexity of terminological reasoning revisited. In *Proc. of LPAR'99*, number 1705 in LNAI, pp. 181–200. Springer-Verlag, 1999.
14. C. Lutz. Reasoning with concrete domains. In *Proc. of IJCAI-99*, pp. 90–95, Stockholm, Sweden, July 31 – August 6, 1999. Morgan-Kaufmann Publishers.
15. C. Lutz. Interval-based temporal reasoning with general TBoxes. In *Proc. of IJCAI-01*, Seattle, 2001.
16. C. Lutz. NExpTime-complete description logics with concrete domains. LTCS-Report 00-01, LuFG Theoretical Computer Science, RWTH Aachen, Germany, 2000. See http://www-lti.informatik.rwth-aachen.de/Forschung/Reports.html.
17. E. M. Post. A variant of a recursively unsolvable problem. *Bull. Am. Math. Soc.*, 52:264–268, 1946.