RWTH
LTCS–Report

Aachen University of Technology
Research group for
Theoretical Computer Science

NExpTime-complete Description Logics with Concrete Domains

Carsten Lutz

LTCS-Report 00-01

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

Ahornstr. 55
52074 Aachen
Germany

NExpTime-complete Description Logics with Concrete Domains

Carsten Lutz
RWTH Aachen, LuFG Theoretical Computer Science
Ahornstr. 55, 52074 Aachen
lutz@informatik.rwth-aachen.de

August 22, 2001

Contents

1 Introduction

Description logics (DLs) are a family of logical formalisms well-suited for the representation of and reasoning about conceptual knowledge on an abstract logical level. However, for many knowledge representation applications, it is essential to integrate the abstract logical knowledge with knowledge of a more concrete nature. As an example, consider the modeling of manufacturing processes, where it is necessary to represent "abstract" entities like subprocesses and workpieces and also "concrete" knowledge, e.g., about the duration of processes and physical dimensions of the manufactured objects [2; 25].

The standard technique for extending Description Logics to allow for the representation of concrete knowledge is to use so-called concrete domains which have been introduced by Baader and Hanschke in [1]. Baader and Hanschke define the description logic $\mathcal{ALC}(\mathcal{D})$, i.e., the extension of the basic propositionally complete description logic $\mathcal{ALC}$ with concrete domains. More precisely, $\mathcal{ALC}(\mathcal{D})$ can be parameterized with a concrete domain $\mathcal{D}$, where $\mathcal{D}$ provides a set of predicates over a given domain like, e.g., the real numbers or the set of time intervals. The concrete domain predicates can then be used inside a concrete domain concept constructor. For example, the $\mathcal{ALC}(\mathcal{D})$-concept

$$\forall \mathit{subprocess}.\mathit{Drilling} \sqcap \exists \mathit{workpiece}\ \mathit{diameter}.{\leq}5\mathit{cm}$$

describes a process all of whose subprocesses are drilling processes and which is related to a workpiece whose diameter is at most 5 centimeters. In the example, *Drilling* is a concept name (unary predicate), *subprocess*, *workpiece*, and *diameter* are roles (binary predicates), and $\leq 5cm$ is a predicate from the concrete domain. The second conjunct demonstrates the use of the concrete domain concept constructor. More information on concrete domains can, e.g., be found in [4; 11; 17].

In this paper, we are interested in the complexity of reasoning with DLs which provide concrete domains, where "reasoning" refers to testing satisfiability and subsumption of concepts. In [20], we proved that reasoning with $\mathcal{ALC}(\mathcal{D})$ is PSpace-complete provided that reasoning with the concrete domain $\mathcal{D}$ (i.e., testing the satisfiability of finite conjunctions of predicates from $\mathcal{D}$) is in PSpace. However, for many applications, the expressivity of $\mathcal{ALC}(\mathcal{D})$ is not su cient and one wants to extend this logic with additional concept- and role-constructors, and with so-called TBoxes. We investigate several such extensions and show that, in the extended logics, reasoning becomes considerably harder. More precisely, we consider the extension of $\mathcal{ALC}(\mathcal{D})$ with

1. acyclic TBoxes,

2. inverse roles (and inverse features), and

3. a role-forming concrete domain constructor.

We prove that reasoning with $\mathcal{ALC}(\mathcal{D})$ and general TBoxes is undecidable which explains why we extend $\mathcal{ALC}(\mathcal{D})$ with the weaker acyclic TBoxes (see, e.g., [21] for acyclic TBoxes and [7; 15] for general TBoxes).

By introducing a NExpTime-complete variant of the Post Correspondence Problem [23; 14], we show that there exists a concrete domain $\mathcal{P}$ for which reasoning is in PTime such that reasoning with each of the above three extensions of $\mathcal{ALC}(\mathcal{D})$ (parameterized with the concrete domain $\mathcal{P}$) 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 [19], we show that the extension of "many" PSpace Description Logics with acyclic TBoxes does not increase the complexity of reasoning. Moreover, it is well-known that $\mathcal{ALC}$ extended with inverse roles is still in PSpace (see, e.g., [16]).

As a corresponding upper bound, we show that, if reasoning with a concrete domain $\mathcal{D}$ is in NP, then reasoning with the DL $\mathcal{ALCRPI}(\mathcal{D})$ with acyclic TBoxes is in NExpTime. The logic $\mathcal{ALCRPI}(\mathcal{D})$ is the extension of $\mathcal{ALC}(\mathcal{D})$ with inverse roles *and* role-forming concrete domain constructors. Finally, we investigate whether $\mathcal{ALCRPI}(\mathcal{D})$ can be augmented by so-called feature agreement and feature disagreement constructors. This step is rather natural since the mentioned constructors are closely related to concrete domains and amenable to a similar algorithmic treatment [1; 20]. We can, however, show that reasoning with the logic $\mathcal{ALCIF}$ is already undecidable.

2 Description Logics

In this section, we introduce the description logics which we are concerned with in the remainder of this paper. We first introduce the logic $\mathcal{ALCI}(\mathcal{D})$ which extends $\mathcal{ALC}(\mathcal{D})$ with inverse roles and then extend it to 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 $\mathcal{ALCI}(\mathcal{D})$.

2.1 The Description Logic $\mathcal{ALCI}(\mathcal{D})$

In this section, the Description Logic $\mathcal{ALCI}(\mathcal{D})$ is introduced. We start by defining concrete domains which were first introduced by Baader and Hanschke [1].

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 $P^\mathcal{D} \subseteq \Delta_\mathcal{D}^n$. A predicate conjunction of the form

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

where P_i is an n_i-ary predicate for $1 \leq i \leq k$ and the $x_j^{(i)}$ are variables, is called *satisfiable* iff there exists a function   mapping the variables in c to elements of $\Delta_\mathcal{D}$ such that $(\ (x_0^{(i)}), \ldots, \ (x_{n_i}^{(i)})) \in P_i^\mathcal{D}$ for $1 \leq i \leq k$. Such a function is called a *solution* for c. A concrete domain $\mathcal{D}$ is called *admissible* iff

1. the set of its predicate names is closed under negation and contains a name $\top_{\mathcal{D}}$ for $\Delta_{\mathcal{D}}$ and
2. the satisfiability problem for finite conjunctions of predicates is decidable.

With $\overline{P}$, we denote the negation of the predicate P, i.e., the predicate with the extension $\overline{P}^{\mathcal{D}} = \Delta_{\mathcal{D}} \setminus P^{\mathcal{D}}$.

We will only consider concrete domains which are admissible. 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 $f_1, \ldots, f_n \in N_{aF}$ and $g \in N_{cF}$, is called a *path*.[1] 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:
   (a) $\neg C$, $C \sqcap D$, $C \sqcup D$,
   (b) $\exists R.C$, $\forall R.C$,
   (c) $\exists u_1, \ldots, u_n.P$, and
   (d) $g\uparrow$.

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 such that $C \in sub(C)$.

In the following, 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 use the following abbreviations:

- $\exists f_1 \cdots f_n.C$ for $\exists f_1. \cdots \exists f_n.C$,
- $\forall f_1 \cdots f_n.C$ for $\forall f_1. \cdots \forall f_n.C$, and
- $(f_1 \cdots f_n g)\uparrow$ for $\forall f_1. \cdots \forall f_n.g\uparrow$.

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 the terminological knowledge of an application domain and is introduced in the following.

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.

[1] A concrete feature is a path of length 1.

- 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. In the DL literature, researchers often consider TBox formalisms which are more expressive than the one just introduced. For example, one may admit cyclic TBoxes [22; 26] or so called general TBoxes in which the left-hand sides of concept definitions may be arbitrary concepts instead of just concept names [7; 15]. However, we will see that admitting general TBoxes makes reasoning with $\mathcal{ALC}(\mathcal{D})$ (and hence also $\mathcal{ALCI}(\mathcal{D})$) undecidable. We now define the semantics of $\mathcal{ALCI}(\mathcal{D})$.

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 $f_1^\mathcal{I} \quad \cdots \quad f_n^\mathcal{I} \quad g^\mathcal{I}$, where denotes function composition and $f_1 \quad f_2(a) = f_2(f_1(a))$ for f_1 and f_2 functions. The interpretation function is extended to arbitrary roles and concepts as follows:

$$
\begin{aligned}
(R^-)^\mathcal{I} &:= \{(a,b) \mid (b,a) \in R^\mathcal{I}\} \\
(C \sqcap D)^\mathcal{I} &:= C^\mathcal{I} \cap D^\mathcal{I} \\
(C \sqcup D)^\mathcal{I} &:= C^\mathcal{I} \cup D^\mathcal{I} \\
(\neg C)^\mathcal{I} &:= \Delta_\mathcal{I} \setminus C^\mathcal{I} \\
(\exists R.C)^\mathcal{I} &:= \{a \in \Delta_\mathcal{I} \mid \{b \mid (a,b) \in R^\mathcal{I}\} \cap C^\mathcal{I} \neq \emptyset\} \\
(\forall R.C)^\mathcal{I} &:= \{a \in \Delta_\mathcal{I} \mid \{b \mid (a,b) \in R^\mathcal{I}\} \subseteq C^\mathcal{I}\} \\
(\exists u_1, \ldots, u_n.P)^\mathcal{I} &:= \{a \in \Delta_\mathcal{I} \mid u_i^\mathcal{I}(a) = x_i \text{ for } 1 \le i \le n \text{ and } (x_1, \ldots, x_n) \in P^\mathcal{D}\} \\
(g\uparrow)^\mathcal{I} &:= \{a \in \Delta_\mathcal{I} \mid g^\mathcal{I}(a) \text{ undefined}\}
\end{aligned}
$$

Let C be a concept and $\mathcal{T}$ be a TBox. If $C^\mathcal{I} \neq \emptyset$, then $\mathcal{I}$ is called a *model* for C. If $A^\mathcal{I} = D^\mathcal{I}$ for all $A \doteq D \in \mathcal{T}$, then $\mathcal{I}$ is called a *model* for $\mathcal{T}$. If R is a role (g a concrete feature) and we have $(a,b) \in R^\mathcal{I}$ ($g^\mathcal{I}(x) = y$), then b is called R-filler of a (y g-filler of x) in $\mathcal{I}$.

Throughout this paper, we will 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 which was introduced 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 $\Delta_\mathcal{I} \cup \Delta_\mathcal{P}$. Obviously, Baader and Hanschke's logic is slightly more expressive than ours. However, in knowledge representation it seems rather hard to find any cases in which the additional expressiveness is really needed. Furthermore, separating concrete and abstract features allows a clearer algorithmic treatment and clearer proofs.

To avoid considering roles such as R^{--}, we define a function *Inv* which returns the inverse of a role. More precisely, $\mathit{Inv}(R) = R^-$ if R is a role name, and $\mathit{Inv}(R) = S$ if $R = S^-$. We generally assume that concepts contain only roles of the form R and R^- (where R is a role name) which can obviously be done without loss of generality. The basic reasoning problems on concepts are defined as follows.

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

$$D^\mathcal{I} \subseteq C^\mathcal{I} \text{ for all models } \mathcal{I} \text{ of } \mathcal{T}.$$

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: C *subsumes* D iff C subsumes D w.r.t. the empty TBox. C is *satisfiable* iff it is satisfiable w.r.t. the empty TBox.

It is well-known that (un)satisfiability and subsumption can be mutually reduced to each other, i.e., $C \sqsubseteq_\mathcal{T} D$ 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 $C \sqsubseteq_\mathcal{T} \bot$ (where $\bot$ abbreviates $A \sqcap \neg A$ for an arbitrary concept name A). We prove decidability of satisfiability and subsumption of $\mathcal{ALCI}(\mathcal{D})$-concepts in Section 4. Throughout this paper, we call two concepts C and D *equivalent* iff C subsumes D and D subsumes C.

2.2 The Description Logic $\mathcal{ALCRPI}(\mathcal{D})$

The Description Logic $\mathcal{ALCRP}(\mathcal{D})$ was introduced in [11] and extends $\mathcal{ALC}(\mathcal{D})$ with a role-forming concrete domain constructor, i.e., it allows the definition of roles with reference to the concrete domain. In this section, we extend the logic $\mathcal{ALCI}(\mathcal{D})$ with this role-forming constructor.

Definition 6 (Predicate Roles). A *predicate role* is an expression of the form

$$\exists(u_1, \dots, u_n), (v_1, \dots, v_m).P$$

where P is an $n+m$-ary predicate. The semantics is given as follows:

$$\begin{aligned}(\exists(u_1, \dots, u_n),&(v_1, \dots, v_m).P)^\mathcal{I} := \\ &\{(a,b) \in \Delta_\mathcal{I} \times \Delta_\mathcal{I} \mid u_i^\mathcal{I}(a) = x_i \text{ for } 1 \leq i \leq n, \\ &\ v_i^\mathcal{I}(b) = y_i \text{ for } 1 \leq i \leq m, \text{ and } (x_1, \dots, x_n, y_1, \dots, y_m) \in P^\mathcal{D}\}\end{aligned}$$

With $\mathcal{R}$, we denote the set of predicate roles. The set of $\mathcal{ALCRPI}(\mathcal{D})$ roles $\widehat{\mathcal{R}}$ is defined as $\widehat{N_R} \cup \mathcal{R} \cup \{R^- \mid R \in \mathcal{R}\}$. An $\mathcal{ALCRPI}(\mathcal{D})$-*concept* is an $\mathcal{ALCI}(\mathcal{D})$-concept whose roles are from $\widehat{\mathcal{R}}$. Hence, in $\mathcal{ALCRPI}(\mathcal{D})$, predicate roles may be used everywhere where a role name from $N_R \setminus N_{aF}$ is allowed in $\mathcal{ALCI}(\mathcal{D})$. An $\mathcal{ALCRPI}(\mathcal{D})$-concept which does not contain the converse constructor on roles is called an $\mathcal{ALCRP}(\mathcal{D})$-*concept.* In the following, a role which is either a predicate role or the inverse of a predicate role is called *complex role.*

$\mathcal{ALCRPI}(\mathcal{D})$ TBoxes are defined in the obvious way. For example, the following concept is an $\mathcal{ALCRPI}(\mathcal{D})$-concept:

$$A \sqcap \exists g, fg.P \sqcap \forall f.\forall(\exists(g),(g).P)^-.\neg A$$

This concept is unsatisfiable since every domain object satisfying it would have to be in both A and $\neg A$ which is impossible. In [10], it is proved that satisfiability and subsumption of $\mathcal{ALCRP}(\mathcal{D})$-concepts is undecidable. Furthermore, as shown in [11], there exists a decidable fragment of the logic $\mathcal{ALCRP}(\mathcal{D})$ which contains $\mathcal{ALC}(\mathcal{D})$ as a sublogic. 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.

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

We may now define restricted concepts.

Definition 8 (Restricted $\mathcal{ALCRPI}(\mathcal{D})$-concept). An $\mathcal{ALCRPI}(\mathcal{D})$-concept 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.

All $\mathcal{ALCRPI}(\mathcal{D})$-concepts we use in this paper (also inside TBoxes) are restricted. Hence, we will in the following write "$\mathcal{ALCRPI}(\mathcal{D})$-concept" for "restricted $\mathcal{ALCRPI}(\mathcal{D})$-concept". Note that the et of restricted $\mathcal{ALCRPI}(\mathcal{D})$-concepts is closed negation, and, hence, subsumption of restricted $\mathcal{ALCRPI}(\mathcal{D})$-concepts can be reduced to satisfiability of restricted $\mathcal{ALCRPI}(\mathcal{D})$-concepts.

The restrictions given in [11] for the logic $\mathcal{ALCRP}(\mathcal{D})$ are slightly less restrictive than the ones given here. They additionally admit concepts of the form $\exists u_1, \ldots, u_n.P$ "inside" universal restrictions of the form $\forall R.D$, where R is a predicate role, provided that (i) the feature chains $u_1, \ldots, u_n$ do not contain any abstract features and (ii) the $\exists u_1, \ldots, u_n.P$ concept is not nested inside additional value or exists restrictions in $\forall R.D$. For example, the concept $\forall(\exists(g),(g).P).(A \sqcap \exists g.P)$ is restricted in the sense of [11] but not in our sense. The concepts $\forall(\exists(g),(g).P).\exists S.(A \sqcap \exists g.P)$ with S a role name and $\forall(\exists(g),(g).P).(A \sqcap \exists fg.P)$ are not restricted in either sense. The reason for the more restricted definition given above is the presence of the inverse role constructor. When constructing a tableau algorithm for $\mathcal{ALCRPI}(\mathcal{D})$ with the weaker restrictions given in [11], one runs into termination problems.[2] Consider, for example, the concept

$$\exists g.\top_{\mathcal{D}} \ \sqcap\ \exists f^{-}g.\top_{\mathcal{D}} \ \sqcap\ \forall(\exists(g),(fg).P_2).(\exists g.\top_{\mathcal{D}} \ \sqcap\ \exists f^{-}.\top)$$

where $P_2^{\mathcal{D}} = \Delta_{\mathcal{D}} \times \Delta_{\mathcal{D}}$. A straightforward tableau algorithm would generate an infinite "f^{-}-path" of objects, each of which has a "concrete g-successor". In fact, it seems rather easy to prove undecidability of $\mathcal{ALCRPI}(\mathcal{D})$ with the weaker restrictions using a technique similar to the one used in [10] to show undecidability of unrestricted $\mathcal{ALCRP}(\mathcal{D})$.

In Section 4, we prove that satisfiability and subsumption of restricted $\mathcal{ALCRPI}(\mathcal{D})$-concepts (as defined above) are decidable in nondeterministic exponential time. Before we do this, we establish several lower bounds for the complexity of reasoning with concrete domains.

3 Lower Complexity Bounds

In this section, we define a NExpTime-complete variant of Post's Correspondence Problem (PCP) and a concrete domain $\mathcal{P}$. We then reduce the NExpTime-complete variant of the PCP to the satisfiability of $\mathcal{ALC}(\mathcal{P})$-concepts w.r.t. TBoxes, the satisfiability of $\mathcal{ALCI}(\mathcal{P})$-concepts, and the satisfiability of $\mathcal{ALCRP}(\mathcal{P})$-concepts (the latter two without reference to TBoxes).

3.1 Post's Correspondence Problem

Post's Correspondence Problem was introduced by Emil Post [23] and is a very useful undecidable problem which is defined as follows.

Definition 9 (PCP). A *Post Correspondence Problem (PCP) P* is given by a finite, non-empty list $(\ell_1, r_1), \ldots, (\ell_k, r_k)$ of pairs of non-empty words over some alphabet Σ.[3] A sequence of integers $i_1, \ldots, i_m$, with $m \geq 1$, is called a *solution* for P iff

$$\ell_{i_1} \cdots \ell_{i_m} = r_{i_1} \cdots r_{i_m}.$$

[2] Readers not familiar with tableau algorithms may skip this comment or return to it after reading Section 4.

[3] Usually, the word lists may also contain the empty word. We use this formulation since, in our case, it allows for simpler proofs.

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, i.e.,

$$|P| = \sum_{1 \leq i \leq k} |\ell_i| + |r_i|.$$

A solution $i_1, \ldots, i_m$ 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.

In the following, when talking of "the PCP" (as opposed to "a PCP"), we refer to the problem of deciding whether a given PCP P has a solution. Undecidability of the (general) PCP was first proved in [23] and later reproved by Hopcroft and Ullman in [14]. In [9], a variant of the PCP is listed as an NP-complete problem (problem number [SR11]). In this variant, a PCP is given by a finite lists of word pairs $(\ell_1, r_1), \ldots, (\ell_k, r_k)$ and a positive integer $K \leq k$. As solutions, only sequences of length at most K are admitted. Inspired by this result, we prove NExpTime-completeness of the $2^n + 1$-PCP. The main di culty is proving NExpTime-hardness. First, we introduce another variant of the PCP.

Definition 10 (MPCP). Let $P = (\ell_1, r_1), \ldots, (\ell_k, r_k)$ be a PCP. A solution $i_1, \ldots, i_m$ for P is called an *MPCP-solution* iff $i_1 = 1$. With *MPCP*, we denote the version of the PCP that admits only MPCP-solutions.

For a function $f(n)$ from $\mathbb{N}$ to $\mathbb{N}$, we define $f(n)$-MPCPs and $f(n)$-MPCP-solutions in the obvious way. The next lemma illustrates the relationship between the 2^n-PCP and the 2^n-MPCP.

Lemma 11. *If the 2^n-MPCP is* NExpTime*-hard, then the 2^n+1-PCP is* NExpTime*-hard.*

Proof It has to be shown that the 2^n-MPCP reduces to the $2^n + 1$-PCP. Hopcroft and Ullman give a reduction from the MPCP to the PCP [14], i.e., they define a translation γ which maps MPCPs to PCPs such that P has an MPCP-solution iff $\gamma(P)$ has a solution. A close examination reveals that P has an MPCP-solution of length at most i iff $\gamma(P)$ has a solution of length at most $i + 1$. ❑

By the lemma just proved, it is su cient to show that the 2^n-MPCP is NExpTime-hard. Before we do this, we give a lemma showing the relationship between different variants of the MPCP.

Lemma 12. *Let $g(n) = 2^{a*n^d}$, where $a \in \mathbb{N}_+$ and $d \in \mathbb{N}_+$ are constants.*[4] *If the $g(n)$-MPCP is* NExpTime*-hard, then the 2^n-MPCP is* NExpTime*-hard.*

Proof We need to show that the $g(n)$-MPCP reduces to the 2^n-MPCP. Let $P = (\ell_1, r_1), \ldots, (\ell_k, r_k)$ be an MPCP with $|P| = n$. W.l.o.g., we may assume $n > 2$ since the claim is trivial if $n = 2$.

For the reduction, augment P by new Σ^+-words ℓ_{k+1} and r_{k+1} yielding P' such that $|P'| = a * n^d$ and ℓ_{k+1} and r_{k+1} do not appear in solutions of P'. It is easy to

[4] When writing 2^{n^d}, we mean $2^{(n^d)}$.

see that this is possible: We need to increase the size of P by $m = a * n^d - n$. Pick a symbol σ not appearing in P (it is not important whether $\sigma \in \Sigma$ since we may just change the underlying alphabet). We set $\ell_{k+1} := \sigma^{m-1}$ and $r_{k+1} := \sigma$. Since $n > 2$, we have $m > 2$ and hence it is obvious that ℓ_{k+1} and r_{k+1} do not appear in solutions of P'. Furthermore, P has a $g(x)$-solution iff P' has a 2^n-solution. ❑

In order to do show that the 2^n-MPCP is NExpTime-hard, we use a reduction of the acceptance problem of Turing Machines.

Definition 13 (Turing Machine). A *nondeterministic Turing Machine* is given by a tuple $M := (Q, \Gamma, \ , q_0, Q_f)$. Q is a finite set of *states* where $q_0 \in Q$ is the *initial state* and $Q_f \subseteq Q$ is a set of *final states*. Γ is a finite set of *symbols* with $\Gamma \cap Q = \emptyset$ which always contains the special symbol B called the *blank symbol*. Finally,   is a *transition function* which maps $Q \times \Gamma$ to the power set of $Q \times \Gamma \times \{\mathit{left}, \mathit{right}, \mathit{stay}\}$.
Let $M = (Q, \Gamma, \ , q_0, Q_f)$ be a (nondeterministic) Turing Machine. An *ID* uqv of M is a word in $\Gamma^* Q \Gamma^*$. An ID has the usual interpretation, i.e., it describes the inscription of the infinite tape (all tape cells "before u" and "behind v" are labeled with B), the current state q, and the head position of M, which is on the rightmost symbol of u. The usual transition relation on IDs is denoted by $\longmapsto$. Intuitively. $uqv \longmapsto u'q'v'$ if a single step of M in ID uqv may result in ID $u'q'v'$. An exact definition is omitted and can be found in any book on recursion theory, see e.g. [14]. By $\overset{*}{\longmapsto}$, we denote the reflexive transitive closure of $\longmapsto$.
M *accepts* an input w (given as an initial tape inscription) iff there exists a $q_f \in Q_f$ such that $q_0 w \overset{*}{\longmapsto} u q_f v$ for some $u, v \in \Gamma^*$.

We now give a transformation from Turing Machines and their inputs to MPCPs which is crucial for proving the central result of this section. The transformation is identical to the one used by Hopcroft and Ullman to prove undecidability of the general PCP [14]. We repeat it for the sake of completeness. Let $M = (Q, \Gamma, \ , q_0, Q_f)$ be a Turing Machine and fix an input w for M. We now define the corresponding MPCP $P^M_w = (\ell_1, r_1), \ldots, (\ell_k, r_k)$. The first pair (ℓ_1, r_1) is defined as

$$\ell_1 := \sharp \qquad r_1 := \sharp q_0 w \sharp.$$

The set of remaining pairs (ℓ_i, r_i) is partitioned into 4 groups and can be found in Figure 1. As stated by Hopcroft and Ullman, if P^M_w has a solution, then this solution corresponds to a word starting with $\sharp q_0 w \sharp u_1 q_1 v_1 \sharp \cdots \sharp u_n q_n v_n$, where subwords between successive $\sharp$'s are successive IDs in a computation of M with input w and q_n is a final state.

In the following, we fix a turing machine M and a word w and prove several properties of the PCP P^M_w. We call a pair of words (x, y) a *partial solution* iff x is a prefix of y and there exists a sequence of integers $i_1, \ldots, i_m$ such that $x = \ell_{i_1} \cdots \ell_{i_m}$ and $y = r_{i_1} \cdots r_{i_m}$. Hopcroft and Ullman prove the following lemma.

Lemma 14. *Suppose that there exists a sequence of IDs* $q_0 w \longmapsto u_1 q_1 v_1 \longmapsto \cdots \longmapsto u_n q_n v_n$. *Then there exists a partial solution*

$$\begin{aligned}(x, y) = (&\sharp q_0 w \sharp u_1 q_1 v_1 \sharp \cdots \sharp u_{n-1} q_{n-1} v_{n-1} \sharp, \\ &\sharp q_0 w \sharp u_1 q_1 v_1 \sharp \cdots \sharp u_{n-1} q_{n-1} v_{n-1} \sharp u_n q_n v_n \sharp).\end{aligned}$$

Group I			
	Left word	Right word	
	X	X	for each $X \in \Gamma$
	$\sharp$	$\sharp$	
Group II. For each $q \in Q \setminus Q_f$, $p \in Q$, and $X, Y, Z \in \Gamma$:			
	Left word	Right word	
	qX	Yp	if $(q, X) = (p, Y, R)$
	ZqX	pZY	if $(q, X) = (p, Y, L)$
	$q\sharp$	$Yp\sharp$	if $(q, B) = (p, Y, R)$
	$Zq\sharp$	$pZY\sharp$	if $(q, B) = (p, Y, L)$
Group III. For each $q \in Q_f$ and $X, Y \in \Gamma$:			
	Left word	Right word	
	XqY	q	
	Xq	q	
	qY	q	
Group IV.			
	Left word	Right word	
	$q\sharp\sharp$	$\sharp$	for each $q \in Q_f$

Figure 1: The MPCP translation.

In the following, we denote the length of a word w with $|w|$.

Lemma 15. *If there exists a partial solution*

$$
\begin{aligned}
(x, y) = (&\sharp q_0 w \sharp u_1 q_1 v_1 \sharp \cdots \sharp u_{n-1} q_{n-1} v_{n-1} \sharp, \\
&\sharp q_0 w \sharp u_1 q_1 v_1 \sharp \cdots \sharp u_{n-1} q_{n-1} v_{n-1} \sharp u_n q_n v_n \sharp),
\end{aligned}
$$

with $q_n \in Q_f$, and $|u_i v_i| \leq n + |w|$ for $1 \leq i \leq n$, then there exists a (partial) solution

$$
\begin{aligned}
(x', y') = (&\sharp q_0 w \sharp u_1 q_1 v_1 \sharp \cdots \sharp u_{n-1} q_{n-1} v_{n-1} \sharp u_n q_n v_n \sharp \cdots \sharp u_r q_r v_r \sharp\sharp, \\
&\sharp q_0 w \sharp u_1 q_1 v_1 \sharp \cdots \sharp u_{n-1} q_{n-1} v_{n-1} \sharp u_n q_n v_n \sharp \cdots \sharp u_r q_r v_r \sharp\sharp)
\end{aligned}
$$

with $r \leq 2n + |w|$ and $|u_i v_i| \leq n + |w|$ for all $1 \leq i \leq r$.

Proof By induction over $\max(|u_n|, |v_n|)$, it can be shown that each partial solution

$$
\begin{aligned}
(x, y) = (&\sharp q_0 w \sharp u_1 q_1 v_1 \sharp \cdots \sharp u_{n-1} q_{n-1} v_{n-1} \sharp, \\
&\sharp q_0 w \sharp u_1 q_1 v_1 \sharp \cdots \sharp u_{n-1} q_{n-1} v_{n-1} \sharp u_n q_n v_n \sharp),
\end{aligned}
$$

with $q_n \in Q_f$ and $\max(|u_n|, |v_n|) > 0$ can be extended to a partial solution

$$
\begin{aligned}
(x, y) = (&\sharp q_0 w \sharp u_1 q_1 v_1 \sharp \cdots \sharp u_{n-1} q_{n-1} v_{n-1} \sharp u_n q_n v_n \sharp, \\
&\sharp q_0 w \sharp u_1 q_1 v_1 \sharp \cdots \sharp u_{n-1} q_{n-1} v_{n-1} \sharp u_n q_n v_n \sharp u_{n+1} q_n v_{n+1} \sharp),
\end{aligned}
$$

where $\max(|u_n|, |v_n|) > \max(|u_{n+1}|, |v_{n+1}|)$. Both the induction start and step can easily be shown by using (at most) $|u_n| + |v_n|$ concatenations of pairs from Group I and a single concatenation of a pair from Group III.

Obviously, the induction needs at most $|u_n|+|v_n|$ steps, and, hence, it follows that (x, y) can be extended to a partial solution

$$\begin{aligned}(x'', y'') = (&\sharp q_0 w \sharp u_1 q_1 v_1 \sharp \cdots \sharp u_{n-1} q_{n-1} v_{n-1} \sharp u_n q_n v_n \sharp \cdots \sharp u_{r-1} q_n v_{r-1} \sharp,\\ &\sharp q_0 w \sharp u_1 q_1 v_1 \sharp \cdots \sharp u_{n-1} q_{n-1} v_{n-1} \sharp u_n q_n v_n \sharp \cdots \sharp u_{r-1} q_n v_{r-1} \sharp q_n \sharp)\end{aligned}$$

where $q_n \in Q_f$ and $r \leq n+|u_n v_n|$. Since $|u_n v_n| \leq n+|w|$, we have (i) $r \leq 2n+|w|$ and (ii) $|u_i v_i| \leq n+|w|$ for all $1 \leq i \leq r$ by construction of (x'', y''). A single concatenation with the pair from Group IV yields the desired solution (x', y'). ❑

We now establish the lower bound for the 2^n-MPCP.

Proposition 16. *It is* NExpTime*-hard to decide whether a 2^n-MPCP has a solution.*

Proof Let M be a Turing Machine which solves a NExpTime-complete problem and stops after at most $2^{|w|^d}$ steps on any input w. W.l.o.g., we assume that M makes at least $\max\{|w|, 2\}$ steps on w before stopping.[5] The reason for this will become clear later. We show that

$$M \text{ accepts } w \text{ iff } P_w^M \text{ has a } 2^{a*|P_w^M|^d}\text{-solution} \qquad (*)$$

for some integer $a > 2$. It then remains to apply Lemma 12 to obtain NExpTime-hardness.

First for the "only if" direction. Let w be an input to M and assume that M accepts w in n steps, where $n \leq 2^{|w|^d}$. Then there exists a sequence of IDs $q_0 w \longmapsto u_1 q_1 v_1 \longmapsto \cdots \longmapsto u_n q_n v_n$ such that $q_n \in Q_f$. By Lemma 14, there exists a partial solution

$$\begin{aligned}(x, y) = (&\sharp q_0 w \sharp u_1 q_1 v_1 \sharp \cdots \sharp u_{n-1} q_{n-1} v_{n-1} \sharp,\\ &\sharp q_0 w \sharp u_1 q_1 v_1 \sharp \cdots \sharp u_{n-1} q_{n-1} v_{n-1} \sharp u_n q_n v_n \sharp),\end{aligned}$$

for P_w^M. Since a Turing Machine writes at most one symbol per step, we obviously have $|u_i v_i| \leq n+|w|$ for $1 \leq i \leq n$. By Lemma 15, there exists a solution $I = i_1, \ldots, i_m$ corresponding to a word

$$w_I = \ell_{i_1} \cdots \ell_{i_m} = r_{i_1} \cdots r_{i_m} = \sharp q_0 w \sharp u_1 q_1 v_1 \sharp \cdots \sharp u_r q_r v_r \sharp\sharp$$

with $r \leq 2n + |w|$ and $|u_i v_i| \leq n + |w|$ for all $1 \leq i \leq r$. Since, by assumption, M makes at least $|w|$ steps if started on w, it follows that $r \leq 3n$ and $|u_i v_i| \leq 2n$ for all $1 \leq i \leq r$. We need an estimation for the length m of the solution $i_1, \ldots, i_m$. Obviously, we have $m \leq r * (2n + 2) + 2$ since m is clearly bounded by the number of symbols in w_I, and the length of each subword of w_I of the form $\sharp u_1 q_i v_i$ is bounded by $2n + 2$. It follows that $m \leq 6n^2 + 6n + 2$ and hence $m \leq n^6$ since M makes at least

[5] To be precise, this implies that we also assume $|w| \geq 1$. This can, however, also be done w.l.o.g.

2 steps before stopping, i.e., $n > 2$. Since $m \leq n^6$ and $n \leq 2^{|w|^d}$, we have $m \leq 2^{6*|w|^d}$, and, since $|w| \leq |P_w^M|$, we have $m \leq 2^{6*|P_w^M|^d}$.

Now for the "if" direction. Assume that M does not accept w, i.e., no computation of M on w reaches a final state. We claim that, for each partial solution (x, y) of P_w^M, there exists a sequence of IDs $q_0 w \vdash u_1 q_1 v_1 \vdash \cdots \vdash u_n q_n v_n$ such that x is a prefix of

$$\sharp q_0 w \sharp u_1 q_1 v_1 \sharp \cdots \sharp u_{n-1} q_{n-1} v_{n-1} \sharp u_n,$$

and y is a prefix of

$$\sharp q_0 w \sharp u_1 q_1 v_1 \sharp \cdots \sharp u_{n-1} q_{n-1} v_{n-1} \sharp u_n q_n v_n \sharp u_n).$$

It is straightforward to prove this by induction on the length m of the sequence of integers $i_1, \ldots, i_m$ corresponding to the partial solution (x, y). Obviously, this implies that the pair from Group IV do not appear in partial solutions since this pair refers to final states and final states are never reached by computations of M on w. It follows that, for all partial solutions (x, y), x contains strictly more $\sharp$ symbols than y which implies $|x| > |y|$. Hence, there exists no solution for P_w^M. ❑

The main result of this section is now easily obtained.

Theorem 17. *It is* NExpTime-*complete to decide whether a* $2^n + 1$-*PCP has a solution.*

Proof NExpTime-hardness is an immediate consequence of Proposition 16 and Lemma 11. To decide the $2^n + 1$-PCP, a nondeterministic Turing Machine may simply "guess" a $2^n + 1$-solution and then check its validity. Since it is not hard to see that this can be done in exponential time, the $2^n + 1$-PCP is in NExpTime. ❑

3.2 A Concrete Domain for Encoding the PCP

In this section, we introduce a concrete domain that will allow to reduce the $2^n + 1$-PCP to concept satisfiability.

Definition 18 (Concrete Domain $\mathcal{P}$). Let Σ be an alphabet. The concrete domain $\mathcal{P}$ is defined by setting $\Delta_\mathcal{P} := \Sigma^*$ 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 $\neq_\epsilon^\mathcal{P} = \Sigma^+$,
- a binary equality predicate $=$ and a binary inequality predicate $\neq$, and
- for each $w \in \Sigma^+$, two binary predicates $conc_w$ and $nconc_w$ with

$$conc_w^\mathcal{P} = \{(u, v) \mid v = uw\} \text{ and } nconc_w^\mathcal{P} = \{(u, v) \mid v \neq uw\}.$$

Since the definition of $\mathcal{P}$ depends on Σ, it would be more precise to define a concrete domain $\mathcal{P}_\Sigma$ for each alphabet Σ. For simplicity, we assume Σ to be fixed. It is obvious that $\Phi_\mathcal{P}$ is closed under negation. To show that $\mathcal{P}$ is admissible, we need to show that the satisfiability of finite predicate conjunctions is decidable. We do this by developing an appropriate algorithm.

We start by introducing a normal form for predicate conjunctions. Let c be a predicate conjunction. Then there exists a predicate conjunction c' which is satisfiable iff c is satisfiable and which contains only predicates from the set $\{nword, =_\epsilon, \neq, conc_w\}$. The conjunction c' can be computed from c by applying the following normalization steps.

1. Eliminate all occurrences of the *word* predicate from c and call the result c_1.

2. Let x be a variable not appearing in c_1. Augment c_1 by the conjunct $=_\epsilon(x)$ and then replace every occurrence of $\neq_\epsilon(y)$ in c_1 with $\neq(x, y)$ calling the result c_2.

3. Let $\beta_1, \ldots, \beta_k$ be all conjuncts in c_2 which are of the form $nconc_w(x, y)$ and let $x_1, \ldots, x_k$ be variables not appearing in c_2. For each i with $1 \leq i \leq k$ and $\beta_i = conc_w(y, z)$, augment c_2 by the conjuncts $conc_w(y, x_i)$ and $\neq(x_i, z)$. Then delete the conjunct β_i from c_2. Call the result c_3.

4. Remove occurrences of the $=$ predicate from c_3 by "filtration": Let $\sim$ be the equivalence relation induced by occurrences of the $=$ predicate in c_3. For each variable x occurring in c_3, substitute every occurrence of x in c_3 by $[x]_\sim$, i.e., by the equivalence class of x w.r.t. $\sim$. Then delete all occurrences of the $=$ predicate from c_3. The result of this step is the normal form c' of c.

Obviously, the normalization process preserves satisfiability, i.e., a predicate conjunction c is satisfiable iff its normal form c' is satisfiable. The blowup of the size of c produced by the normalization is at most linear.

Before the algorithm itself can be given, we introduce some notions. Let c be a predicate conjunction (not necessarily in normal form). With $V(c)$, we denote the set of variables used in c. The *conc-graph* $G(c) = (V, E)$ of c is the directed graph described by occurrences of $conc_w$ predicates in c, i.e., $V = V(c)$ and $(x, y) \in E$ iff $conc_w(x, y)$ is a conjunct of c for some word w. A conjunction c is said to have a *conc*-cycle if $G(c)$ has a cycle. The *distance* $dist(v, v')$ of two variables $v, v' \in V$ in c is the length of the *longest* path leading from v to v' in $G(c)$. With $lev(v)$, we denote

$$\max\{k \mid dist(v, v') = k \text{ and } v' \text{ is a sink}\}$$

where a sink is a node which has no outgoing edges. Let $w, w' \in \Sigma^+$. The function *pre* is defined as follows:

$$pre(w, w') = \begin{cases} v & \text{if } w = vw' \text{ with } v \neq \epsilon \\ \text{undefined} & \text{if no such } v \text{ exists} \end{cases}$$

The algorithm for deciding the satisfiability of conjunctions of predicates in normal

define procedure *sat*-$\mathcal{P}(c)$
  if c contains the *nword* predicate or $norm(c) = inconsistent$ **then**
    return *inconsistent*
  for i = 1 **to** $|V(c)|$ **do**
    while there exist $x, y, y' \in V(c)$ with $lev(x) = i$
        and $w, w' \in \Sigma^+$ with $w \neq w'$ such that
        $conc_w(y, x)$ and $conc_{w'}(y', x)$ are in c **do**
      if neither w is a suffix of w' not vice versa **then** (A3)
        return *inconsistent*
      w.l.o.g., assume that w' is a suffix of w. (A4)
      // since *norm* was just applied, we have $w = vw'$ for a $v \neq \epsilon$.
      replace $conc_w(y, x)$ by $conc_{pre(w,w')}(y, y')$ in c
      if $norm(c) = inconsistent$ **then**
        return *inconsistent*
  if there exist $x, y \in V(c)$ and a $w \in \Sigma^+$ such that
      $conc_w(y, x)$ and $=_\epsilon(x)$ are in c **then**
    return *inconsistent*
  if there are $x_1, \ldots, x_k, y_1, \ldots, y_k \in V(c)$
      and $w_1, \ldots, w_{k-1} \in \Sigma^+$ such that
      ($=_\epsilon (x_1)$, $=_\epsilon (y_1)$ are in c or $x_1 = y_1$), and $\neq(x_k, y_k)$ is in c and
      $conc_{w_i}(x_i, x_{i+1})$ and $conc_{w_i}(y_i, y_{i+1})$ are in c for $1 \leq i \leq k-1$ **then**
    return *inconsistent*
  return *consistent*

define procedure $norm(c)$ // c is passed "by reference" (see text)
  while there exist $x, y, y' \in V(c)$ with $y \neq y'$ and a $w \in \Sigma^+$ such that
      $conc_w(y, x)$ and $conc_w(y', x)$ are in c **do** (A1)
    replace every occurrence of y' in c by y
  if c contains a *conc*-cycle **then**
    return *inconsistent*
  if there exist $x, y \in V(c)$ and $w, w' \in \Sigma^+$ with $w \neq w'$ such that (A2)
      $conc_w(y, x)$ and $conc_{w'}(y, x)$ are in c **then**
    return *inconsistent*
  return *consistent*

Figure 2: The $\mathcal{P}$ satisfiability algorithm.

form can be found in Figure 2. Note that the parameter to *norm* is passed "by reference", i.e., changes made to c in *norm* are also effective in the calling procedure. Before the correctness of the algorithm is proved formally, we explain the underlying intuitions. Assume that the satisfiability of a conjunction c_0 is to be decided. The algorithm repeatedly performs several normalization steps and inconsistency checks. If all normalization is done and no inconsistencies were found, the resulting conjunction is satisfiable. Since the normalization is satisfiability-preserving, this implies satisfiability of c_0. The normalizations in the algorithm are concerned with situations of the form

$$conc_w(y, x), conc_{w'}(y', x)$$

where several cases can be distinguished:

A1. $y \neq y'$ and $w = w'$. In this case, y and y' describe the same word and can be identified.

A2. $y = y'$ and $w \neq w'$. In this case, c is unsatisfiable.

A3. $y \neq y'$ and neither w is a su x of w' nor vice versa. In this case, c is unsatisfiable.

A4. $y \neq y'$ and w' is a su x of w. We may replace the above situation by $conc_{pre(w,w')}(y, y'), conc_{w'}(y', x)$.

These cases and several additional inconsistencies checked by the algorithm (e.g., *conc*-cycles) are discussed in detail in the correctness proof. If all normalization was performed and no inconsistency is found, we have obtained a conjunction c for which the corresponding *conc*-graph is a forest, i.e., a collection of trees, and which satisfies some additional properties guaranteeing that we can construct a solution for c. We now formally prove correctness and termination of the algorithm. For a conjunction c, let $|c|$ denote the number of conjuncts in c.

Lemma 19 (Correctness and Termination). *Let c be an input to sat-$\mathcal{P}$. The algorithm terminates after $\mathcal{O}(|c|^k)$ steps (where $k \in \mathbb{N}$ is constant) returning* consistent *if c has a solution and* inconsistent *otherwise.*

Proof We first prove correctness, i.e., that c has a solution if *sat-$\mathcal{P}$* returns *consistent* and that c has no solution if *sat-$\mathcal{P}$* returns *inconsistent*. After doing this, we establish termination. To prove correctness, we walk through the algorithm and examine the performed steps in detail.

If the algorithm returns *unsatisfiable* in the first **if** statement, this is either because c contains the *nword* predicate (in this case, c is obviousy unsatisfiable) or because *norm* returned *unsatisfiable*. Hence, let us examine the *norm* procedure. The **while** loop eliminates situations of the form $conc_w(y, x), conc_w(y', x)$ with $y \neq y'$. In this situation, we clearly have that $(y) = (y')$ for all solutions for c and hence it is a satisfiability-preserving operation to identify y and y'. After this normalization step, *norm* checks if c contains a *conc*-cycle or if there is a situation of the form $conc_w(y, x), conc_{w'}(y, x)$ with $w \neq w'$. Obviously, c is unsatisfiable in both cases.

We now return to the main procedure. The **for** loop iterates from 1 to $|V(c)|$. The **while** loop inside the **for** loop examines situations of the form $conc_w(y, x), conc_{w'}(y', x)$ with $w \neq w'$ and $lev(x) = i$. Since *norm* was just applied (note that it is also applied at the end of the **while** loop), we have $y \neq y'$. In the following, we call a situation $conc_w(y, x), conc_{w'}(y', x)$ with $w \neq w'$ and $y \neq y'$ a *fork for x*. If neither w is a su x of w' nor vice versa, c is obviously unsatisfiable. If w' is a su x of w, replacing $conc_w(y, x)$ by $conc_{pre(w,w')}(y, y')$ is a satisfiability-preserving operation. Since, in doing so, we may have created any of the situations that *norm* checks for, the *norm* procedure needs to be re-applied. Note that both the elimination of a fork and the application of *norm* may create new forks $conc_{\tilde{w}}(\tilde{y}, \tilde{x}), conc_{\tilde{w}'}(\tilde{y}', \tilde{x})$. However, it is not hard to see that (in both cases) $lev(\tilde{x}) > lev(x)$ and hence the newly generated forks will be eliminated during a later step of the **for** loop. Finally, situations of the form $conc_w(y, x), =_\epsilon(x)$ and certain situations involving the $\neq$ predicate are checked for whose existence imply unsatisfiability of c.

We need to show that c has a solution if the algorithm returns *consistent*. By the above considerations, c has the following properties if *consistent* is returned:

1. c contains no *conc*-cycles,
2. c contains no situations of the form $conc_w(y, x), conc_{w'}(y', x)$ with $y \neq y'$,
3. if $conc_w(y, x), conc_{w'}(y, x)$ are in c, then $w = w'$, and
4. if $=_\epsilon(x)$ is in c, then there exist no $y \in V(c)$ and $w \in \Sigma^+$ such that $conc_w(y, x)$ is in c.

Properties 1 and 2 imply that the *conc*-graph $G(c) = (V, E)$ of c is a forest. We define a solution for c inductively. Our strategy is to start defining (v) for the variables v which are the root of a tree in the forest $G(c)$ and then "move downwards the trees" to define (v') for all remaining variables v'. Since the edges in the trees correspond to $conc_w$-predicates, our choice of (v) for the root v of a tree determines (v') for all remaining nodes v' in the same tree. We must, however, carefully choose (v) for the roots v of the trees to guarantee that all $=_\epsilon$ and $\neq$ predicates in c are satisfied. Let t be the number of trees in $G(c)$ and let $w_1, \ldots, w_t$ be words from Σ^+ such that

$$|w_{i+1}| - |w_i| \geq |V| * \max\{w \mid conc_w \text{ is used in } c\} \text{ for } 1 \leq i < t.$$

For the induction start, fix an ordering on the trees in $G(c)$ and let $x_1, \ldots, x_t \in V$ such that x_i is the root of the i'th tree in $G(c)$. For all $1 \leq i \leq t$, set

- $(x_i) = \epsilon$ if $=_\epsilon(x_i)$ is in c and
- $(x_i) = w_i$ otherwise.

For the induction step, if x is a node with $(x) = w$ and $conc'_w(x, y)$ is in c, then set $(y) = ww'$. is well-defined since $G(c)$ is a forest and Property 3 from above holds. Obviously, satisfies all $conc_w$ predicates in c. Property 4 from above implies that, if $=_\epsilon(x)$ is in c, then x is the root of a tree and hence also satisfies all $=_\epsilon(x)$ predicates in c. Now for $\neq(x, y)$ predicates. We make a case distinction:

- x and y are in the same tree. By definition of and since the last **if** clause in the main procedure did not apply, $\neq(x, y)$ is satisfied.
- x and y are in different trees, and both trees have a root z with $(z) = \epsilon$, i.e., $=_\epsilon(z)$ is in c. Identical to the above case.

- x and y are in different trees and at least one of the trees has a root z with $(z) \neq \epsilon$. Let z and z' be the roots of the two trees. By definition of , we have

  $$\text{abs}(|\ (z)| - |\ (z')|) \geq |V| * \max\{w \mid conc_w \text{ is used in } c\}$$

  where abs(x) denotes the absolute value of x. This implies that, for any two nodes x' and y', where x' is in the tree with root z and y' is in the tree with root z', we have $(x') \neq\ (y')$.

It remains to show termination after at most polynomially many steps. This amounts to showing that the two **while** loops terminate after at most polynomially many steps since it is easy to see that all the tests (in the **if** clauses) and operations (node and conjunction replacements) can be performed in polynomial time.

Termination after polynomially many steps is obvious for the loop in the *norm* procedure since, in every iteration, the number of variables in c decreases and the algorithm never introduces new variables. Now for the **while** loop in the main procedure. If a fork $conc_w(y, x), conc_{w'}(y', x)$ is found, then $conc_w(y, x)$ is replaced by $conc_{pre(w,w')}(y, y')$. As was already noted, this and the application of *norm* may generate new forks $conc_{\tilde{w}}(\tilde{y}, \tilde{x}), conc_{\tilde{w}'}(\tilde{y}', \tilde{x})$ but only with the restriction $lev(\tilde{x}) > lev(x)$. Hence the newly generated fork will not be considered during the current iteration step of the **for** loop. We conclude that the **while** loop terminates after polynomially many steps since the number of forks $conc_w(y, x), conc_{w'}(y', x)$ with $lev(x) = i$ is clearly bounded by $|c|$. ❑

The following proposition is an immediate consequence of the lemma just proved and the fact that the blowup produced by the normalization is at most linear.

Proposition 20. *It is decidable in deterministic polynomial time whether a finite conjunction of predicates from $\mathcal{P}$ has a solution.*

Corollary 21. *The concrete domain $\mathcal{P}$ is admissible.*

3.3 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. 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 [20], and (2) admitting acyclic TBoxes does "usually" not increase the complexity of reasoning [19].

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 such that C is satisfiable w.r.t. $\mathcal{T}[P]$ iff P has a solution. Figure 3 contains the reduction TBox and Figure 4 an example model for $|P| = 2$. In the figures, l, r, x, and y denote abstract features. The first equality in Figure 3 is not meant as a concept definition but as 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 $u_1, \ldots, u_4$

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

$$
\begin{aligned}
C_0 \doteq {} & \exists\ell.C_1 \sqcap \exists r.C_1 \\
& \sqcap\ Ch[\ell r^{n-1}g_\ell, r\ell^{n-1}g_\ell, \ell r^{n-1}g_r, r\ell^{n-1}g_r] \\
& \vdots \\
C_{n-2} \doteq {} & \exists\ell.C_{n-1} \sqcap \exists r.C_{n-1} \\
& \sqcap\ Ch[\ell r g_\ell, r\ell g_\ell, \ell r g_r, r\ell g_r] \\
C_{n-1} \doteq {} & Ch[\ell g_\ell, r g_\ell, \ell g_r, r g_r]
\end{aligned}
$$

$$
\begin{aligned}
C \doteq {} & C_0 \\
& \sqcap\ \exists\ell^n g_\ell. =_\epsilon \;\sqcap\; \exists\ell^n g_r. =_\epsilon \\
& \sqcap\ \exists r^n y.\exists g_\ell, g_r. = \;\sqcap\; \exists r^n y g_\ell. \neq_\epsilon \\
& \sqcap\ Ch[r^n g_\ell, r^n x g_\ell, r^n g_r, r^n x g_r] \\
& \sqcap\ Ch[r^n x g_\ell, r^n y g_\ell, r^n x g_r, r^n y g_r]
\end{aligned}
$$

Figure 3: The $\mathcal{ALC}(\mathcal{P})$ reduction TBox $\mathcal{T}[P]$ ($n = |P|$).

appropriately. We first informally explain the intuition underlying the reduction and then give a formal proof of its correctness.

The general idea is to define $\mathcal{T}[P]$ such that models of C and $\mathcal{T}[P]$ have the form of a binary tree of depth $|P|$ whose edges 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 $\mathcal{I}$ of C and $\mathcal{T}[P]$ 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 (not to be confused with pairs $(\ell_i, r_i) \in P$). Let the abstract objects $a_{n,0}, \ldots a_{n,2^n-1}$ be the leaves of this tree (see Figure 4 for the naming scheme). By the second line of the definitions of the $C_0, \ldots, C_{n-1}$ concepts, every $a_{n,i}$ $(0 \leq i < 2^n)$ has a filler x_i for the concrete feature g_ℓ and a filler y_i for the concrete feature g_r. These second lines also ensure that the x_i and y_i objects are connected via two "predicate chains" where the predicates on the chains are either equality or $conc_w$ predicates. 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 $(x_i, x_{i+1}) \in conc^{\mathcal{P}}_{\ell_j}$ and $(y_i, y_{i+1}) \in conc^{\mathcal{P}}_{r_j}$. Furthermore, by the second line of the definition of C, 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 su cient, and we need to "add" two more

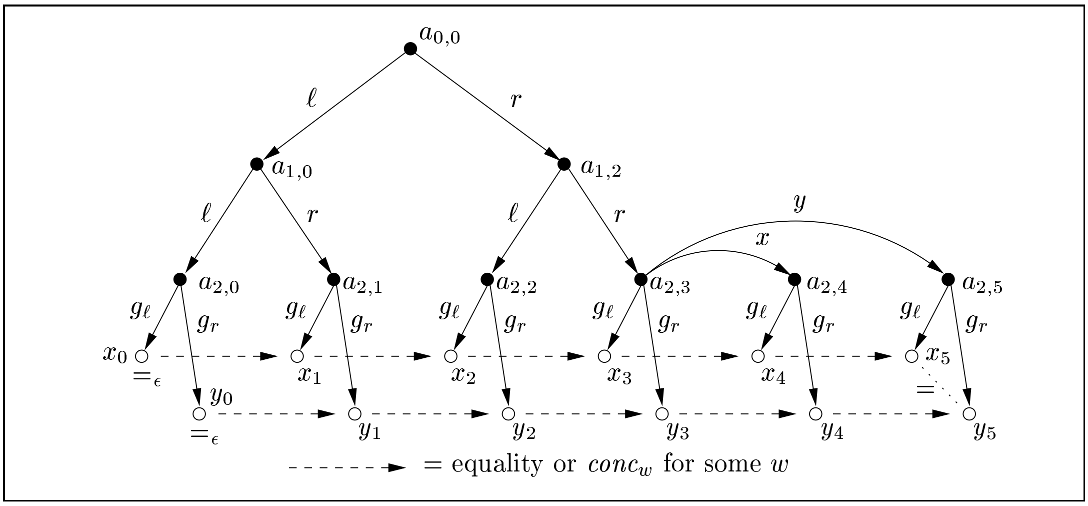

Figure 4: An example model of C for $n = 2$.

objects $a_{n,2^n}$ and $a_{n,2^n+1}$ which behave analogously to the objects $a_{n,0}, \ldots a_{n,2^n-1}$, i.e., have associated concrete objects x_{2^n}, y_{2^n} and x_{2^n+1}, y_{2^n+1}, respectively. This is done by the last two lines of the definition of C. Finally, the third line of the definition of C 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.

Lemma 22. *Let $P = (\ell_1, r_1), \ldots, (\ell_k, r_k)$ be a PCP. Then P has a solution iff the concept (name) C is satisfiable w.r.t. the TBox $\mathcal{T}[P]$.*

Proof During the proof, we abbreviate $|P|$ by n. First assume that $C[P]$ is satisfiable. Using induction over n and considering the definitions of the C_i concepts, it is easy to show that there exist objects $a_{i,j}$ for $0 \leq i \leq n$ and $0 \leq j < 2^i$ such that $a_{0,0} \in C^{\mathcal{I}}$,

1. $\ell^{\mathcal{I}}(a_{i,j}) = a_{(i+1),2j}$ and $r^{\mathcal{I}}(a_{i,j}) = a_{(i+1),(2j+1)}$ for $0 \leq i < n$ and $0 \leq j < 2^i$, and
2. $a_{i,j} \in (Ch[\ell r^{n-(i+1)} g_\ell, r\ell^{n-(i+1)} g_\ell, \ell r^{n-(i+1)} g_r, r\ell^{n-(i+1)} g_r])^{\mathcal{I}}$ for $0 \leq i < n$.

The first Property implies that the $a_{i,j}$ form a binary tree in which edges connecting left successors are labeled with ℓ, edges connecting right successors are labeled with r, and nodes are not necessarily distinct. The naming scheme for nodes is as indicated in Figure 4.

We now establish a certain property for every two neighbouring fringe nodes $a_{n,j}$ and $a_{n,(j+1)}$ which will then allow us to deduce the existence of two sequences of concrete objects related by $conc_w$ predicates and the equality predicate. Corresponding nodes from the two sequences represent partial solutions of P. Fix two nodes $a_{n,j}$ and $a_{n,(j+1)}$ with $0 \leq j < 2^n - 1$. By induction over n, it is straightforward to prove that there exists a common ancestor $a_{m,r}$ of $a_{n,j}$ and $a_{n,(j+1)}$ such that

$$(\ell r^{n-(m+1)})^{\mathcal{I}}(a_{m,r}) = a_{n,j} \text{ and } (r\ell^{n-(m+1)})^{\mathcal{I}}(a_{m,r}) = a_{n,j+1}$$

By Property 2 from above, we have

$$a_{m,r} \in (Ch[\ell r^{n-(m+1)} g_\ell, r\ell^{n-(m+1)} g_\ell, \ell r^{n-(m+1)} g_r, r\ell^{n-(m+1)} g_r])^{\mathcal{I}}.$$

Since this holds independently from the choice of j, we may use the definition of the $Ch[u_1, u_2, u_3, u_4]$ concept to conclude that there exist concrete objects $x_0, \ldots, x_{2^n-1}$ and $y_0, \ldots, y_{2^n-1}$ and indexes $i_1, \ldots, i_{2^n-1} \in \{1, \ldots, k\} \cup \{\clubsuit\}$ such that

1. $g_\ell^{\mathcal{I}}(a_{n,j}) = x_j$ and $g_r^{\mathcal{I}}(a_{n,j}) = y_j$ for $0 \leq j < 2^n$, and
2. for all $1 \leq j \leq 2^n - 1$,
   - if $i_j = \clubsuit$ then $x_{j-1} = x_j$ and $y_{j-1} = y_j$, and
   - $(x_{j-1}, x_j) \in conc^{\mathcal{P}}_{\ell_{i_j}}$ and $(y_{j-1}, y_j) \in conc^{\mathcal{P}}_{r_{i_j}}$ otherwise.

Analogously, by the last two lines of the definition of C, there exist abstract objects $a_{n,2^n}, a_{n,(2^n+1)}$, concrete objects $x_{2^n}, x_{2^n+1}, y_{2^n}, y_{2^n+1}$, and indexes $i_{2^n}, i_{2^n+1} \in \{1, \ldots, k\} \cup \{\clubsuit\}$ such that

1. $x^{\mathcal{I}}(a_{n,2^n-1}) = a_{n,2^n}$ and $y^{\mathcal{I}}(a_{n,2^n-1}) = a_{n,(2^n+1)}$,
2. $g_\ell^{\mathcal{I}}(a_{n,i}) = x_i$ and $g_r^{\mathcal{I}}(a_{n,i}) = y_i$ for $i \in \{2^n, 2^n + 1\}$,
3. for all $j \in \{2^n, 2^n + 1\}$,
   - if $i_j = \clubsuit$ then $x_{j-1} = x_j$ and $y_{j-1} = y_j$, and
   - $(x_{j-1}, x_j) \in conc^{\mathcal{P}}_{\ell_{i_j}}$ and $(y_{j-1}, y_j) \in conc^{\mathcal{P}}_{r_{i_j}}$ otherwise.

Moreover, by the second and third line of the definition of C, we have $x_0 = y_0 = \epsilon$ and $x_{2^n+1} = y_{2^n+1} \neq \epsilon$. Taking together these observations, it is clear that the sequence $i'_1, \ldots, i'_p$, which can be obtained from $i_1, \ldots, i_{2^n+1}$ by eliminating all i_j with $i_j = \clubsuit$, is a solution for P. Furthermore, we obviously have $1 < p \leq 2^n + 1$.

Now for the "only if" direction. Assume that P has a solution $i_1, \ldots, i_m$ with $m \leq 2^{|P|} + 1$. With L_j (resp. R_j), we denote the concatenation $\ell_{i_1} \cdots \ell_{i_j}$ (resp. $r_{i_1} \cdots r_{i_j}$) for $1 \leq j \leq m$ and set $L_0 = R_0 = \epsilon$ and $L_j = L_m$ (resp. $R_j = R_m$) for all $j > m$. We define a model $\mathcal{I}$ for $\mathcal{T}[P]$ with the form of a binary tree of depth n such that $C^{\mathcal{I}} \neq \emptyset$. Again, the object names in Figure 4 indicate the naming scheme used.

$$\Delta^{\mathcal{I}} := \{a_{i,j} \mid 0 \leq i \leq n, 0 \leq j < 2^i\} \cup \{a_{n,2^n}, a_{n,(2^n+1)}\}$$

For all i, j with $0 \leq i < n$ and $0 \leq j < 2^i$ set

$$\ell^{\mathcal{I}}(a_{i,j}) := a_{(i+1),(2j)} \text{ and } r^{\mathcal{I}}(a_{i,j}) := a_{(i+1),(2j+1)}.$$

Set $x^{\mathcal{I}}(a_{n,(2^n-1)}) := a_{n,2^n}$ and $y^{\mathcal{I}}(a_{n,(2^n-1)}) := a_{n,(2^n+1)}$.

For all i with $0 \leq i \leq 2^n + 1$ set

$$g_\ell^{\mathcal{I}}(a_{n,i}) := L_i \text{ and } g_r^{\mathcal{I}}(a_{n,i}) := R_i.$$

It is not hard to verify that $\mathcal{I}$ is a model for $\mathcal{T}[P]$ and that $a_{0,0} \in C^{\mathcal{I}}$. ❑

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

$$
\begin{aligned}
Tree = {} & \exists(R \sqcap \ell).\top \sqcap \exists(R \sqcap r).\top \\
& \sqcap\; Ch[\ell r^{n-1} g_\ell, r\ell^{n-1} g_\ell, \ell r^{n-1} g_r, r\ell^{n-1} g_r] \\
& \sqcap \forall R.\big(\exists(R \sqcap \ell).\top \sqcap \exists(R \sqcap r).\top \\
& \qquad \sqcap\; Ch[\ell r^{n-2} g_\ell, r\ell^{n-2} g_\ell, \ell r^{n-2} g_r, r\ell^{n-2} g_r]\big) \\
& \qquad\qquad \vdots \\
& \sqcap \forall R^{n-2}.\big(\exists(R \sqcap \ell).\top \sqcap \exists(R \sqcap r).\top \\
& \qquad\quad \sqcap\; Ch[\ell r g_\ell, r\ell g_\ell, \ell r g_r, r\ell g_r]\big) \\
& \sqcap \forall R^{n-1}.Ch[\ell g_\ell, r g_\ell, \ell g_r, r g_r]
\end{aligned}
$$

$$
\begin{aligned}
C[P] = {} & Tree \\
& \sqcap\; \exists \ell^n g_\ell.=_\epsilon \;\sqcap\; \exists \ell^n g_r.=_\epsilon \\
& \sqcap\; \exists r^n y.\exists g_\ell, g_r.= \;\sqcap\; \exists r^n y g_\ell.\neq_\epsilon \\
& \sqcap\; Ch[r^n g_\ell, r^n x g_\ell, r^n g_r, r^n x g_r] \\
& \sqcap\; Ch[r^n x g_\ell, r^n y g_\ell, r^n x g_r, r^n y g_r]
\end{aligned}
$$

Figure 5: The $\mathcal{ALCR}(\mathcal{P})$ reduction concept C[P].

Obviously, the size of $\mathcal{T}[P]$ is polynomial in $|P|$ and $\mathcal{T}[P]$ can be constructed in time polynomial in $|P|$. Since subsumption can be reduced to satisfiability, we obtain the following theorem.

Theorem 23. *There exists an admissible concrete domain $\mathcal{D}$ for which satisfiability is in* PTIME *such that satisfiability and subsumption of $\mathcal{ALC}(\mathcal{D})$-concepts w.r.t TBoxes are* NEXPTIME-*hard.*

On first sight, the concrete domain employed for the reduction may look somewhat unnatural since it operates on words. 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 an alphabet Σ can be interpreted as numbers written at base $|\Sigma|+1$ (assuming that the empty word represents 0); the concatenation of two words v and w can then be expressed as $vw = v*(|\Sigma|+1)^{|w|}+w$, where $|w|$ denotes the length of the word w. Hence, a concrete domain which provides the natural numbers, (in)equality, (in)equality to zero, addition, and multiplication is also appropriate for the reductions.

As already noted, there exist other variants of TBoxes than the ones introduced in Section 2. A popular one are so-called general TBoxes which are formally defined

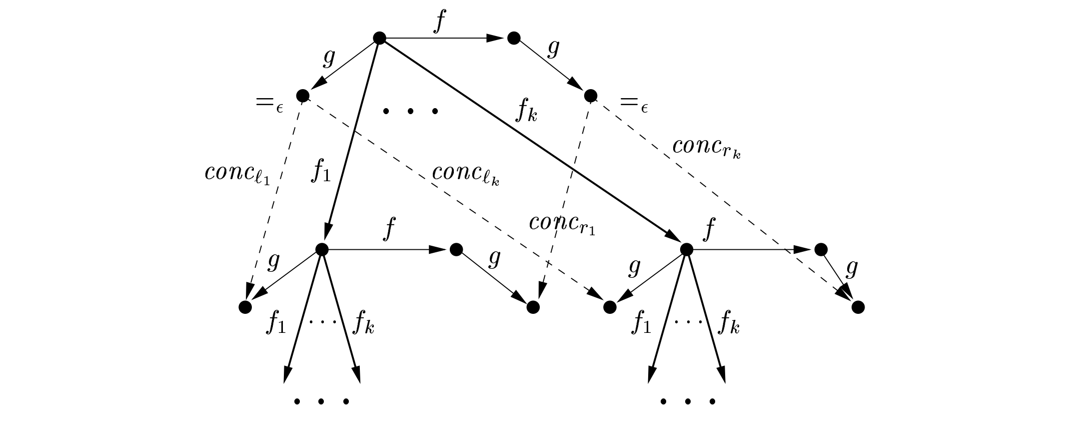

Figure 6: An example model of C w.r.t. $\mathcal{T}$

as follows.

Definition 24 (General TBox). A *general concept inclusion (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 $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. 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 $\mathcal{T}$.

Similar to the main result presented in this section, the following theorem can be obtained.

Theorem 25. *There exists an admissible concrete domain $\mathcal{D}$ such that satisfiability and subsumption of $\mathcal{ALC}(\mathcal{D})$-concepts w.r.t. general TBoxes is undecidable.*

Proof Let P be an instance of the PCP and consider a concept C and a general TBox $\mathcal{T}$ as follows:

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

Here, $C \to D$ is used as an abbreviation for $\neg C \sqcup D$. The first two GCIs ensure that models of C and $\mathcal{T}$ 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 is satisfiable w.r.t. $\mathcal{T}$ iff P has no solution, i.e., we have reduced the general, undecidable PCP to the satisfiability of $\mathcal{ALC}(\mathcal{D})$-concepts w.r.t. general TBoxes. An example model of C w.r.t. $\mathcal{T}$ can be found in Figure 6. It remains to remind the reader that satisfiability reduces to subsumption. ❑

The reduction technique employed to show the lower bound is a rather general one and there surely exist more description logics with concrete domains to which they can

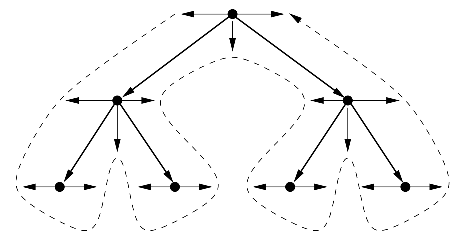

Figure 7: Predicate chains in models of $C[P]$.

be applied. As an example, consider $\mathcal{ALCR}(\mathcal{D})$, i.e., $\mathcal{ALC}(\mathcal{D})$ with role conjunction (see, e.g., [8]). We conjecture that NExpTime-hardness of this logic can be proved analogously to the proof of Theorem 29. The reduction concept $C[P]$ can be found in Figure 5.

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

In this section, we 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.

We employ a reduction that is similar to the one used in the previous section, i.e., it is a reduction of the 2^n+1-PCP and uses the concrete domain $\mathcal{P}$. 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 8. 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. We first informally explain the structure of models of $C[P]$ and then give a formal proof of the correctness of the reduction.

In the reduction given in the previous section, the models of $\mathcal{T}[P]$ are binary trees of depth $|P|$ whose leaves are connected by two chains of concrete domain predicates such that pairs of corresponding nodes (x, y) represent partial solutions of the PCP P. In the $\mathcal{ALCI}(\mathcal{P})$ reduction, 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. As in the previous reduction, we define two chains of concrete domain predicates, only this time they do not connect the leaves of the tree but emulate the structure of the tree following the scheme indicated in Figure 7. 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 9. The existence of the chains is ensured by the definition of X and the second line in the definition of $C[P]$. The concept X establishes the

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

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

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

Figure 8: The $\mathcal{ALCI}(\mathcal{P})$ reduction concept $C[P]$ ($n = |P| - 1$).

edges of the predicate chains as depicted in Figure 9 (more precisely, Figure 9 is a model of the concept X) while the second line of $C[P]$ establishes the edges "leading around" the fringe nodes. Edges of the latter type and the dotted edges in Figure 9 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 length of the number of edges in the tree plus the number of fringe nodes, 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 fringe node 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.

Lemma 26. *Let $P = (\ell_1, r_1), \ldots, (\ell_k, r_k)$ be a PCP. Then P has a solution iff the concept $C[P]$ is satisfiable.*

Proof Let $n = |P| - 1$ during the proof (this implies $n \geq 1$). First assume that $C[P]$ is satisfiable, i.e., that there exists an interpretation $\mathcal{I} = (\Delta_\mathcal{I}, \cdot^\mathcal{I})$ and an $a \in \Delta^\mathcal{I}$ such that $a \in C[P]^\mathcal{I}$. We show that $\mathcal{I}$ has the form of a binary tree of depth n. Using induction over n and considering the first line of the definition of $C[P]$ and the definition of X, it is easy to show that there exist abstract objects $b_{i,j}$ for $0 \leq i \leq n$ and $0 \leq j < 2^i$ such that $b_{0,0} \in C[P]^\mathcal{I}$ and, for $0 \leq i < n$ and $0 \leq j < 2^i$,

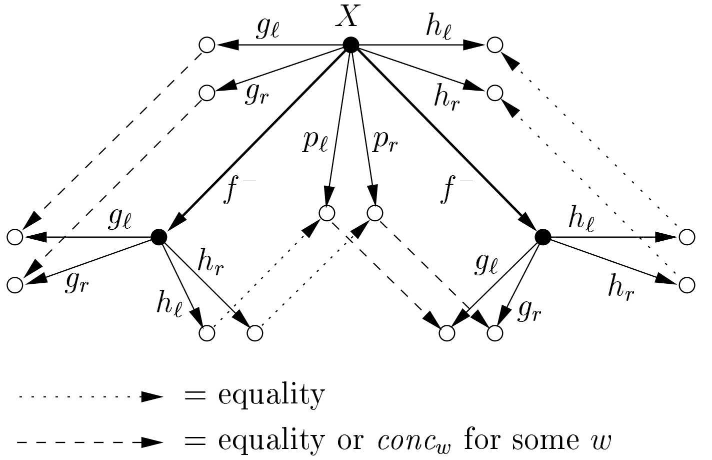

Figure 9: A clipping from a model of $C[P]$.

1. $\{(b_{i,j}, b_{(i+1),2j}), (b_{i,j}, b_{(i+1),(2j+1)})\} \subseteq (f^-)^{\mathcal{I}}$,
2. $b_{(i+1),2j} \in (Ch[fg_\ell, g_\ell, fg_r, g_r])^{\mathcal{I}}$, and
3. $b_{(i+1),(2j+1)} \in (Ch[fp_\ell, g_\ell, fp_r, g_r])^{\mathcal{I}}$.

Obviously, the first property implies that the $b_{i,j}$ form a binary tree of depth n whose edges are labelled with f^-. However, for the remaining proof, it is more convenient to number the nodes in the tree in a different way. For doing this, we define three auxiliary functions.

Let T be a binary tree of depth n whose nodes are labeled with natural numbers in preorder (this tree is independent of the $b_{i,j}$ and of $\mathcal{I}$ in general).[6] With $sucl(n)$ and $sucr(n)$ we denote the node label of the left resp. right successor of the node labeled with n in T ($sucl(n)$ and $sucr(n)$ are undefined if the given node has no successors). Furthermore, for $n \in \mathbb{N}$, $lev(n)$ denotes the level of the node in T labeled with n and is undefined if no such node exists. By "renaming" the nodes $b_{i,j}$, it is easy to show that there exist abstract objects $a_1, \ldots, a_{2^{n+1}-1}$ such that, for all $1 \leq i \leq 2^{n+1} - 1$ with $lev(i) < n$,

1. $f^{\mathcal{I}}(a_{sucl(i)}) = a_i$ and $f^{\mathcal{I}}(a_{sucr(i)}) = a_i$,
2. $a_{sucl(i)} \in (Ch[fg_\ell, g_\ell, fg_r, g_r])^{\mathcal{I}}$, and
3. $a_{sucr(i)} \in (Ch[fp_\ell, g_\ell, fp_r, g_r])^{\mathcal{I}}$.

Note that the a_i form a binary tree of depth n labeled in preorder whose edges are labeled with f^- and whose nodes are not necessarily distinct. Hence, when we in the following talk of the nodes of the tree, we mean the objects $a_1, \ldots, a_{2^{n+1}-1}$. By the second

[6]To label a tree in preorder, first label its root, then inductively label the subtree induced by the root's left successor and finally label the subtree induced by the root's right successor.

line of $C[P]$ and definition of X, there exist concrete objects $x_1, \dots, x_{2^{n+1}-1}, y_1, \dots, y_{2^{n+1}-1}$ such that $g_\ell^{\mathcal{I}}(a_i) = x_i$ and $g_r^{\mathcal{I}}(a_i) = y_i$ for all $1 \leq i < 2^{n+1}$. Next, we prove the following claim:

Claim: For all $1 \leq j < 2^{n+1} - 1$, we have either $x_j = x_{j+1}$ and $y_j = y_{j+1}$ or there exists an $i \in \{1, \dots, k\}$ such that $(x_j, x_{j+1}) \in conc_{\ell_i}^{\mathcal{P}}$ and $(y_j, y_{j+1}) \in conc_{r_i}^{\mathcal{P}}$.

Fix a j with $1 \leq j < 2^{n+1} - 1$. From the preorder numbering scheme, it follows that two cases need to be distinguished:

1. $i + 1 = sucl(i)$. By Property 2 from above, we have $a_{i+1} \in (Ch[fg_\ell, g_\ell, fg_r, g_r])^{\mathcal{I}}$. By definition of Ch, this implies the claim.

2. There exists a node a_t and nodes $a_{s_0}, \dots, a_{s_m}$ $(m \geq 0)$ such that

   - $i + 1 = sucr(t)$,
   - $s_0 = sucl(t)$,
   - for all ℓ with $0 \leq \ell < m$, $s_{\ell+1} = sucr(s_\ell)$, and
   - $s_m = i$

   By the Properties given above, we have $a_{i+1}, a_{s_1}, \dots, a_{s_m} \in (Ch[fg_\ell, g_\ell, fg_r, g_r])^{\mathcal{I}}$ and $a_{s_0} \in (Ch[fp_\ell, g_\ell, fp_r, g_r])^{\mathcal{I}}$. Furthermore, from the numbering scheme, it follows that $lev(i) = n$, and, by the second line of $C[P]$,

   $$a_i \in (\exists(g_\ell, h_\ell).= \sqcap \exists(g_r, h_r).=)^{\mathcal{I}}.$$

   Using the definition of Ch, it is now straightforward to prove the claim.

It is an immediate consequence of the claim that there exist indexes $i_1, \dots, i_{2^{n+1}-2} \in \{1, \dots, k\} \cup \{\clubsuit\}$ such that, for all $1 \leq j < 2^{n+1} - 1$,

- if $i_j = \clubsuit$ then $x_j = x_{j+1}$ and $y_j = y_{j+1}$, and
- $(x_j, x_{j+1}) \in conc_{\ell_{i_j}}^{\mathcal{P}}$ and $(y_j, y_{j+1}) \in conc_{r_{i_j}}^{\mathcal{P}}$ otherwise.

Similarly, by the third, fourth, and fifth line of the definition of $C[P]$, there exist objects $x_{2^{n+1}}, x_{2^{n+1}+1}, x_{2^{n+1}+2}, y_{2^{n+1}}, y_{2^{n+1}+1}, y_{2^{n+1}+2}$ and indexes $i_{2^{n+1}-1}, i_{2^{n+1}}, i_{2^{n+1}+1} \in \{1, \dots, k\} \cup \{\clubsuit\}$ such that

1. $x_\ell^{\mathcal{I}}(a_0) = x_{2^{n+1}}$ and $x_r^{\mathcal{I}}(a_0) = y_{2^{n+1}}$,
   $y_\ell^{\mathcal{I}}(a_0) = x_{2^{n+1}+1}$ and $y_r^{\mathcal{I}}(a_0) = y_{2^{n+1}+1}$,
   $z_\ell^{\mathcal{I}}(a_0) = x_{2^{n+1}+2}$ and $z_r^{\mathcal{I}}(a_0) = y_{2^{n+1}+2}$, and

2. for all $j \in \{2^{n+1} - 1, 2^{n+1}, 2^{n+1} + 1\}$

   - if $i_j = \clubsuit$ then $x_j = x_{j+1}$ and $y_j = y_{j+1}$, and
   - $(x_j, x_{j+1}) \in conc_{\ell_{i_j}}^{\mathcal{P}}$ and $(y_j, y_{j+1}) \in conc_{r_{i_j}}^{\mathcal{P}}$ otherwise.

Moreover, by the last two lines of the definition of $C[P]$, we have $x_0 = y_0 = \epsilon$ and $x_{2^{n+1}+1} = y_{2^{n+1}+1} \neq \epsilon$. Taking together these observations, it is clear that the sequence $i'_1, \ldots, i'_p$, which can be obtained from $i_1, \ldots, i_{2^{n+1}+1}$ by eliminating all i_j with $i_j = \clubsuit$, is a solution for P. Furthermore, since $n = |P| - 1$, we have $0 < p \leq 2^{|P|} + 1$.

Now for the "only if" direction. Assume that P has a solution $i_1, \ldots, i_m$ with $m \leq 2^{|P|} + 1 = 2^{n+1} + 1$. With K^ℓ_j (resp. K^r_j), we denote the concatenation $\ell_{i_1}, \cdots, \ell_{i_j}$ (resp. $r_{i_1}, \cdots, r_{i_j}$) for $1 \leq j \leq m$ and set $K^\ell_0 = K^r_0 = \epsilon$ and $K^\ell_j = K^\ell_m$ (resp. $K^r_j = K^r_m$) for all $j > m$. We define a model for $C[P]$ with the form of a binary tree of depth n.

$$\Delta^\mathcal{I} := \{a_i \mid 1 \leq i < 2^{n+1}\}$$

For all i with $1 \leq i < 2^{n+1}$ and $lev(i) < n$ set
$$f^\mathcal{I}(a_{sucl(i)}) = a_i \text{ and } f^\mathcal{I}(a_{sucr(i)}) = a_i.$$

It remains to set up the concrete features. We first set up only some of the features.

1. $g^\mathcal{I}_\ell(a_i) = K^\ell_{i-1}$ and $g^\mathcal{I}_r(a_i) = K^r_{i-1}$ for $1 \leq i < 2^{n+1}$
2. $x^\mathcal{I}_t(a_0) = K^t_{2^{n+1}-1}$ for $t \in \{\ell, r\}$
3. $y^\mathcal{I}_t(a_0) = K^t_{2^{n+1}}$ for $t \in \{\ell, r\}$
4. $z^\mathcal{I}_t(a_0) = K^t_{2^{n+1}+1}$ for $t \in \{\ell, r\}$

Based on this, we now define the interpretation of the remaining concrete features. With $sucl^j(i)$, we denote the j-fold composition of *sucl*. For $i \in \{1, \ldots, 2^{n+1} - 1\}$ and $t \in \{\ell, r\}$, we set

$$h^\mathcal{I}_t(a_i) := \begin{cases} g^\mathcal{I}_t(a_i) & \text{if } lev(i) = n \\ g^\mathcal{I}_t(a_{sucr^{n-lev(i)}(i)}) & \text{otherwise} \end{cases}$$

$$p^\mathcal{I}_t(a_i) := \begin{cases} g^\mathcal{I}_t(a_{sucl(i)}) & \text{if } lev(i) = n-1 \\ g^\mathcal{I}_t(a_{sucr^{n-lev(i)-1}(sucl(i))}) & \text{if } lev(i) < n-1 \end{cases}$$

Note that nodes a_i with $lev(i) = n$ do not need to have fillers for the concrete feature p. It is straightforward to check that $\mathcal{I}$ is well-defined and that $a_0 \in C[P]^\mathcal{I}$. ❑

Obviously, the size of $C[P]$ is polynomial in $|P|$ and $C[P]$ can be constructed in time polynomial in $|P|$. Since subsumption can be reduced to satisfiability, we obtain the following theorem.

Theorem 27. *There exists an admissible concrete domain $\mathcal{D}$ for which satisfiability is in* PTIME *such that satisfiability and subsumption of $\mathcal{ALCI}(\mathcal{D})$-concepts are* NEXPTIME*-hard.*

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

In this section, we prove that satisfiability of $\mathcal{ALCRP}(\mathcal{P})$-concepts is NExpTime-hard. Hence, adding the role-forming concrete domain constructor yields another extension of $\mathcal{ALC}(\mathcal{D})$ in which reasoning is considerably harder than in $\mathcal{ALC}(\mathcal{D})$ itself.

As in the previous two sections, we give a reduction of the 2^n+1-PCP using the concrete domain $\mathcal{P}$. 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 10, where X and Y denote concept names, x and y denote abstract features, and p denotes a predicate. The equalities in the figure are not concept definitions but serve as abbreviations (c.f. Section 3.4). Note that $S[g,p]$ is a predicate role and not a concept, i.e., $S[g,p]$ is an abbreviation for the role-forming concrete domain constructor $\exists(g),(g).\overline{p}$ (a lowercase p is used for predicates to avoid confusion with the PCP P). $C \to D$ is used as an abbreviation for $\neg C \sqcup D$. We informally explain the structure of models of $C[P]$ before giving a formal proof of its correctness.

Figure 11 contains an example model of $C[P]$ with $|P| = n = 2$. Obviously, the structure of models of $C[P]$ is rather similar to the structure of models of the $\mathcal{ALC}(\mathcal{D})$ reduction TBox $\mathcal{T}[P]$ from Section 3.3: Models have the form of a binary tree of depth n whose fringe nodes (together with two "extra" nodes) are connected by two predicate chains of length 2^n+1. Corresponding nodes on the two chains represent words x and y from partial solutions (x,y) of the PCP P. The *Tree* concept ensures 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 fringe nodes of the tree. More precisely, for a domain object $a \in \Delta^{\mathcal{I}}$, set

$$pos(a) = \Sigma_{i=0}^{n-1} \beta_i(a) * 2^i$$

where

$$\beta_i(a) = \begin{cases} 1 & \text{if } a \in B_i^{\mathcal{I}} \\ 0 & \text{otherwise,} \end{cases}$$

i.e, the number $pos(a)$ is binarily coded by the concept names $B_0, \ldots, B_{n-1}$. The *Tree* and *DistB* concepts ensure that, for two fringe nodes a and a' with $a \neq a'$, we have $pos(a) \neq pos(a')$. Due to the first line of the $C[P]$ concept, every fringe node has (concrete) successors for the g_ℓ and g_r features. The last two lines of $C[P]$ guarantee the existence of the two extra nodes such that (i) both nodes have concrete g_ℓ- and g_r-fillers, and (ii) one of the extra nodes is in $X^{\mathcal{I}}$ while the other is in $Y^{\mathcal{I}}$. It remains to describe how the edges of the two predicate chains are established.

For the sake of simplicity, let us start with describing how the edges ending at the extra nodes are generated. W.l.o.g., we concentrate on the extra node b with $b \in X^{\mathcal{I}}$ and on edges between g_ℓ-fillers. Let a be the fringe node with $pos(a) = 2^n - 1$, x be the g_ℓ-successor of a, and y be the g_ℓ-successor of b (both concrete objects exist according to the definition of $C[P]$). By the fifth line of the definition of $C[P]$, we have $a \in Ext[X]^{\mathcal{I}}$. The concept $Ext[X]$ has the form of a disjunction where each disjunct establishes a different "type" of edge between x and y (and another edge between the corresponding g_r-fillers). We exemplarily use the subconcept $\forall S[g_\ell,=].\neg X$ of $Ext[X]$

$$
\begin{aligned}
DistB[k] &= \prod_{i=0}^{k}((B_i \rightarrow \forall R.B_i) \sqcap \neg B_i \rightarrow \forall R.\neg B_i) \\[1em]
Tree &= \exists R.B_0 \sqcap \exists R.\neg B_0 \\
&\quad \sqcap \forall R.(DistB[0] \sqcap \exists R.B_1 \sqcap \exists R.\neg B_1) \\
&\qquad\qquad\qquad \vdots \\
&\quad \sqcap \forall R^{n-1}.(DistB[n-1] \sqcap \exists R.B_{n-1} \sqcap \exists R.\neg B_{n-1}) \\[1em]
S[g,p] &= \exists (g),(g).\overline{p} \\[1em]
Edge[g,p] &= \left(\bigsqcup_{k=0}^{n-1}\left(\bigsqcup_{j=0}^{k-1}\neg B_j\right) \sqcap (B_k \rightarrow \forall S[g,p].\neg B_k) \sqcap (\neg B_k \rightarrow \forall S[g,p].B_k)\right. \\
&\quad \left.\sqcup \bigsqcup_{k=0}^{n-1}\left(\prod_{j=0}^{k-1} B_j\right) \sqcap (B_k \rightarrow \forall S[g,p].B_k) \sqcap (\neg B_k \rightarrow \forall S[g,p].\neg B_k)\right) \\[1em]
DEdge[P] &= (Edge[g_\ell,=] \sqcap Edge[g_r,=]) \sqcup \\
&\quad \bigsqcup_{(\ell_i,r_i) \text{ in } P} (Edge[g_\ell, conc_{\ell_i}] \sqcap Edge[g_r, conc_{r_i}]) \\[1em]
Ext[D] &= (\forall S[g_\ell,=].\neg D \sqcap \forall S[g_r,=].\neg D) \sqcup \\
&\quad \bigsqcup_{(\ell_i,r_i) \text{ in } P} (\forall S[g_\ell, conc_{\ell_i}].\neg D \sqcap \forall S[g_r, conc_{r_i}].\neg D \\[1em]
C[P] &= Tree \sqcap \forall R^n.\exists g_\ell.word \sqcap \forall R^n.\exists g_r.word \\
&\quad \sqcap \forall R^n.\big[(\neg B_0 \sqcap \cdots \sqcap \neg B_{n-1}) \rightarrow (\exists g_\ell. =_\epsilon \ \sqcap \exists g_r. =_\epsilon) \\
&\qquad\qquad \sqcap \neg(B_0 \sqcap \cdots \sqcap B_{n-1}) \rightarrow DEdge[P] \\
&\qquad\qquad \sqcap (B_0 \sqcap \cdots \sqcap B_{n-1}) \rightarrow \\
&\qquad\qquad\qquad \big(Ext[X] \sqcap \forall x.Ext[Y] \\
&\qquad\qquad\qquad\ \sqcap \exists x.(X \sqcap \exists g_\ell.word \sqcap \exists g_r.word) \\
&\qquad\qquad\qquad\ \sqcap \exists y.(Y \sqcap \exists g_\ell, g_r. = \ \sqcap \exists g_\ell.\neq_\epsilon)\big)\big]
\end{aligned}
$$

Figure 10: The $\mathcal{ALCRP}(\mathcal{P})$ reduction concept $C[P]$ $(n = |P|)$.

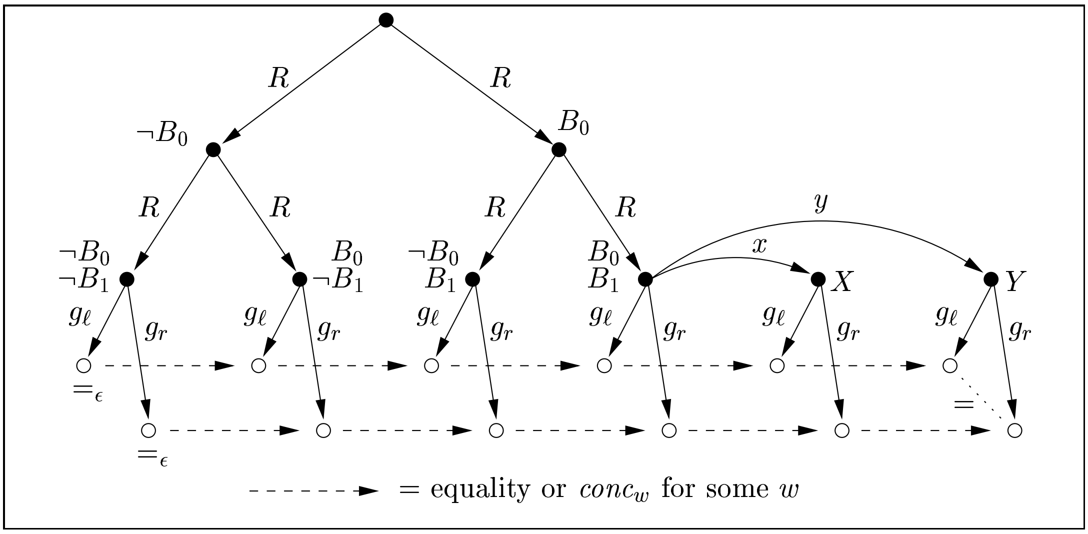

Figure 11: An example model of $C[P]$ with $|P| = 2$.

to demonstrate how the edge between x and y is established. From the fact that $a \in (\forall S[g_\ell, =].\neg X)^\mathcal{I}$ and $b \in X^\mathcal{I}$, it follows that $(a, b) \notin S[g_\ell, =]^\mathcal{I}$, i.e., $(a, b) \notin (\exists (g_\ell), (g_\ell). \neq)^\mathcal{I}$ and thus $(x, y) \notin \neq^\mathcal{D}$, which obviously implies that $(x, y) \in =^\mathcal{D}$. In the case $a \in (\forall S[g_\ell, conc_{\ell_i}].\neg X)^\mathcal{I}$, an analogous argument leads to $(x, y) \in conc^\mathcal{P}_{\ell_i}$.

The edges which do not end at extra nodes are established in a similar way by the *DEdge* and *Edge* concepts. The *DEdge* concept is just a disjunction over the various edge types while the *Edge* concept actually establishes the edges. The *Edge* concept is essentially the negation of the well-known propositional formula

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

which encodes incrementation modulo 2^n, i.e., if k is the number binarly encoded by the propositional variables $x_0, \ldots, x_{n-1}$ and k' is the number binarly encoded by the propositional variables $x'_0, \ldots, x'_{n-1}$, then we have $k' = k + 1$ modulo 2^n (see, e.g., [6]). Assume $a \in (Edge[g_\ell, p])^\mathcal{I}$ (where p is either "=" or $conc_{l_i}$) and let b be the fringe node with $pos(b) = pos(a) + 1$, 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 $pos(c) \neq pos(a) + 1$, 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 in the case of the edges ending at one of the extra nodes, 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 3.3.

Lemma 28. *Let $P = (\ell_1, r_1), \ldots, (\ell_k, r_k)$ be a PCP. Then P has a solution iff the concept $C[P]$ is satisfiable.*

Proof During the proof, we abbreviate $|P|$ by n. First assume that $C[P]$ is satisfiable. Using induction over n and the definitions of the *Tree* and *DistB* concepts, it is easy to show that there exist objects $a_{i,j}$ for $0 \leq i \leq n$ and $0 \leq j < 2^i$ such that

1. $R^{\mathcal{I}}(a_{i,j}) = \{a_{(i+1),2j}, a_{(i+1),(2j+1)}\}$ for $0 \leq i < n$ and $0 \leq j < 2^i$ and
2. $pos(a_{n,j}) = j$ for $0 \leq j < 2^n$.

The first property implies that the $a_{i,j}$ form a binary tree whose edges are labeled by R and whose nodes are not necessarily distinct.[7] The naming scheme for nodes is as indicated in Figure 4. By the first line of the $C[P]$ concept, there exist concrete objects $x_0, \ldots, x_{2^n-1}$ and $y_0, \ldots, y_{2^n-1}$ such that

$$g_\ell^{\mathcal{I}}(a_{n,j}) = x_j \text{ and } g_r^{\mathcal{I}}(a_{n,j}) = y_j \text{ for all } 0 \leq j < 2^n.$$

By the third line of $C[P]$, we have $a_{n,j} \in (DEdge[P])^{\mathcal{I}}$ for all $a_{n,j}$ with $pos(a_{n,j}) \neq 2^n - 1$, i.e., for all $a_{n,j}$ with $0 \leq j < 2^n - 1$. By definition of $DEdge[P]$, for each j with $0 \leq j < 2^n$, we have either

$$a_{n,j} \in (Edge[g_\ell, =] \sqcap Edge[g_r, =])^{\mathcal{I}}$$

or there exists a pair $(\ell_i, r_i) \in P$ such that

$$a_{n,j} \in (Edge[g_\ell, conc_{\ell_i}] \sqcap Edge[g_r, conc_{r_i}])^{\mathcal{I}}.$$

As was already shown in the intuitive explanations, the first property implies $x_j = x_{j+1}$ and $y_j = y_{j+1}$ while the second implies $(x_j, x_{j+1}) \in conc_{\ell_i}^{\mathcal{P}}$ and $(y_j, y_{j+1}) \in conc_{r_i}^{\mathcal{P}}$ (we refrain from repeating the arguments here). Summing up, there exist concrete objects $x_0, \ldots, x_{2^n-1}$ and $y_0, \ldots, y_{2^n-1}$ and indexes $i_1, \ldots, i_{2^n-1} \in \{1, \ldots, k\} \cup \{\clubsuit\}$ such that

1. $g_\ell^{\mathcal{I}}(a_{n,j}) = x_j$ and $g_r^{\mathcal{I}}(a_{n,j}) = y_j$ for $0 \leq j < 2^n$, and
2. for all $1 \leq j \leq 2^n - 1$,
    - if $i_j = \clubsuit$ then $x_{j-1} = x_j$ and $y_{j-1} = y_j$, and
    - $(x_{j-1}, x_j) \in conc_{\ell_{i_j}}^{\mathcal{P}}$ and $(y_{j-1}, y_j) \in conc_{r_{i_j}}^{\mathcal{P}}$ otherwise.

Analogously, by definition of the $Ext[D]$ concept and the last four lines of the definition of $C[P]$, there exist abstract objects $a_{n,2^n}, a_{n,(2^n+1)}$, concrete objects $x_{2^n}, x_{2^n+1}, y_{2^n}, y_{2^n+1}$, and indexes $i_{2^n}, i_{2^n+1} \in \{1, \ldots, k\} \cup \{\clubsuit\}$ such that

1. $x^{\mathcal{I}}(a_{n,2^n-1}) = a_{n,2^n}$ and $y^{\mathcal{I}}(a_{n,2^n-1}) = a_{n,(2^n+1)}$,
2. $g_\ell^{\mathcal{I}}(a_{n,i}) = x_i$ and $g_r^{\mathcal{I}}(a_{n,i}) = y_i$ for $i \in \{2^n, 2^n + 1\}$,
3. for all $j \in \{2^n, 2^n + 1\}$,
    - if $i_j = \clubsuit$ then $x_{j-1} = x_j$ and $y_{j-1} = y_j$, and

[7] The fringe nodes must obvioulsy be distinct because of the B_i concepts. However, some "inner" nodes may coincide.

- $(x_{j-1}, x_j) \in conc^{\mathcal{P}}_{\ell_{i_j}}$ and $(y_{j-1}, y_j) \in conc^{\mathcal{P}}_{r_{i_j}}$ otherwise.

Moreover, by the second and last line of the definition of $C[P]$, we have $x_0 = y_0 = \epsilon$ and $x_{2^n+1} = y_{2^n+1} \neq \epsilon$. Taking together these observations, it is clear that the sequence $i'_1, \ldots, i'_p$, which can be obtained from $i_1, \ldots, i_{2^n+1}$ by eliminating all i_j with $i_j = \clubsuit$, is a solution for P. Furthermore, we obviously have $1 < p \leq 2^n + 1$.

Now for the "only if" direction. Assume that P has a solution $i_1, \ldots, i_m$ with $m \leq 2^{|P|} + 1$. With L_j (resp. R_j), we denote the concatenation $\ell_{i_1} \cdots \ell_{i_j}$ (resp. $r_{i_1} \cdots r_{i_j}$) for $1 \leq j \leq m$ and set $L_0 = R_0 = \epsilon$ and $L_j = L_m$ (resp. $R_j = R_m$) for all $j > m$. We define a model $\mathcal{I}$ for $C[P]$ with the form of a binary tree of depth n. The object names in Figure 4 indicate the naming scheme used. Set

$$\Delta^{\mathcal{I}} := \{a_{i,j} \mid 0 \leq i \leq n, 0 \leq j < 2^i\} \cup \{a_{n,2^n}, a_{n,(2^n+1)}\}.$$

For all $0 \leq j < n$, $B^{\mathcal{I}}_j$ is the smallest superset S of $\{a_{j+1,i} \mid 0 \leq i < 2^j \wedge i \; mod \; 2 \neq 0\}$ which is closed under the following condition:

$$a_{i,j} \in S \text{ and } i < n \Longrightarrow a_{(i+1),(2j)}, a_{(i+1),(2j+1)} \in S.$$

Now for the interpretation of the roles.

For all i, j with $0 \leq i < n$ and $0 \leq j < 2^i$ set $R^{\mathcal{I}}(a_{i,j}) := \{a_{(i+1),(2j)}, a_{(i+1),(2j+1)}\}$.

Set $x^{\mathcal{I}}(a_{n,(2^n-1)}) := a_{n,2^n}$ and $y^{\mathcal{I}}(a_{n,(2^n-1)}) := a_{n,(2^n+1)}$.

For all i with $0 \leq i \leq 2^n + 1$ set $g^{\mathcal{I}}_\ell(a_{n,i}) := L_i$ and $g^{\mathcal{I}}_r(a_{n,i}) := R_i$.

It is not hard to verify that $\mathcal{I}$ is a model for $C[P]$. ❑

Obviously, the size of $C[P]$ is polynomial in $|P|$ and $C[P]$ can be constructed in time polynomial in $|P|$. Since subsumption can be reduced to satisfiability, we obtain the following theorem.

Theorem 29. *There exists an admissible concrete domain $\mathcal{D}$ for which satisfiability is in* PTime *such that satisfiability and subsumption of $\mathcal{ALCRP}(\mathcal{D})$-concepts are* NExpTime*-hard.*

4 Upper Complexity Bound

In this section, we establish an upper bound corresponding to the lower bounds given in the previous section. We consider concrete domains $\mathcal{D}$ for which satisfiability is in NP and show that satisfiability and subsumption of (restricted) $\mathcal{ALCRPI}(\mathcal{D})$-concepts w.r.t. TBoxes is in NExpTime. First, a tableau algorithm for deciding satisfiability of $\mathcal{ALCRPI}(\mathcal{D})$-concepts without reference to TBoxes is devised. Then, we modify the presented algorithm to take into account TBoxes by using "on the fly unfolding" as proposed in [19].

4.1 A Completion Algorithm for $\mathcal{ALCRPI}(\mathcal{D})$

In this section, we prove satisfiability of $\mathcal{ALCRPI}(\mathcal{D})$-concepts (without reference to TBoxes) to be in NExpTime by devising an appropriate algorithm. The presented algorithm is a so-called tableau algorithm which tries to construct a canonical model for the input concept by repeatedly applying completion rules to a completion system. Input concepts are required to be in negation normal form. We start with introducing completion systems, which are the fundamental data structure of the completion algorithm presented in this section.

Definition 30 (Completion System). Let O_a and O_c be disjoint sets of *abstract nodes* and *concrete nodes* (both countably infinite). A *completion tree* for an $\mathcal{ALCRPI}(\mathcal{D})$-concept D is a tree whose set of nodes is a subset of $O_a \uplus O_c$. Each node $a \in O_a$ of the tree is labeled with a subset $\mathcal{L}(a)$ of $sub(D)$, each edge (a,b) with $a,b \in O_a$ is labeled with a (possibly complex) role $\mathcal{L}(a,b)$ occurring in D, and each edge (a,x) with $a \in O_a$ and $x \in O_c$ is labeled with a concrete feature $\mathcal{L}(a,x)$ occurring in D.[8] The following properties have to be satisfied.

1. concrete nodes have no successors,
2. if b and c are successors of a and $\mathcal{L}(a,b) = \mathcal{L}(a,c) = f$ for an abstract feature f, then $b = c$.
3. if b is successor of a and $\mathcal{L}(a,b) = f^-$ for an abstract feature f, then for all successors c of b, we have $\mathcal{L}(b,c) \neq f$.
4. if x and y are successors of a and $\mathcal{L}(a,x) = \mathcal{L}(a,y) = g$ for a concrete feature g, then $x = y$.

A *completion system* for an $\mathcal{ALCRPI}(\mathcal{D})$-concept D is a pair $(\mathbf{T}, \mathcal{P})$, where $\mathbf{T}$ is a completion tree for D and $\mathcal{P}$ is a function mapping each $P \in \Phi_\mathcal{D}$ with arity n appearing in D to a subset of $(O_c)^n$.

Let $\mathbf{T}$ be a completion tree, $R \in \widehat{\mathcal{R}}$, and $a,b \in O_a$. b is called *R-successor* of a in $\mathbf{T}$ iff b is a successor of a and $\mathcal{L}(a,b) = R$ (for concrete features g, the notion *g-successor* is defined analogously). b is called *R-neighbor* of a iff b is R-successor of a or a is $Inv(R)$-successor of b. The notion R-neighbor is extended to paths in the obvious way: Let $u = f_1 \cdots f_n g$ be a path and $x \in O_c$; x is *u-neighbor* of a in $\mathbf{T}$ if there exist nodes $b_1, \ldots, b_n \in O_a$ such that b_1 is f_1-neighbor of a, b_i is f_i-neighbor of b_{i-1} for $1 < i \leq n$, and x is g-successor of b_n (resp. if x is a g-successor of a in the case that $u = g$). With $neighb_\mathbf{T}(a,u)$, we denote the u-neighbor of a in $\mathbf{T}$ (which is unique due to Properties 2 to 4 of completion trees). The index $\mathbf{T}$ is omitted if clear from the context. If R is a predicate role, then b is a *virtual* R-successor of a if

1. $R = \exists(u_1, \ldots, u_n),(v_1, \ldots, v_n).P$,
2. there exist concrete nodes $x_1, \ldots, x_n$ such that $x_i = neighb(a, u_i)$ for $1 \leq i \leq n$,

[8] Recall that predicate roles are expressions of the form $\exists(u_1, \ldots, u_n),(v_1, \ldots, v_n).P$ and a role is called complex if it is either a predicate role or the inverse of a predicate role.

3. there exist concrete nodes $y_1, \dots, y_m$ such that $y_i = neighb(b, v_i)$ for $1 \leq i \leq m$, and

4. $(x_1, \dots, x_n, y_1, \dots, y_m) \in \mathcal{P}(P)$.

If $R = S^-$ and S is a predicate role, then b is a *virtual* R-successor of a if a is a virtual S-successor of b. A node $a \in O_a$ is a *general* R-neighbor of a node $b \in O_a$ if b is an R-neighbor of a or b is a virtual R-successor of a.

If the satisfiability of a concept D is to be decided, the completion algorithm is started with the *initial completion system* $S_D = (\mathbf{T}_D, \mathcal{P}_\emptyset)$, where $\mathbf{T}_D$ is the tree consisting of a single node a with $\mathcal{L}(a) = \{D\}$ and $\mathcal{P}_\emptyset$ maps each $P \in \Phi_\mathcal{D}$ to $\emptyset$. The algorithm repeatedly applies the (yet to be defined) completion rules until (1) it finds a completion system to which no more rules are applicable or (2) it finds a completion system containing a contradiction. If the final completion system contains a contradiction (be it complete or not), D is not satisfiable. Otherwise, the final completion system represents a model for D. Before the completion rules are defined, we introduce a bit of notation.

Definition 31 ("+" operation). An abstract or concrete node is called *fresh* w.r.t. a completion tree $\mathbf{T}$ if it does not appear in $\mathbf{T}$. Let $S = (\mathbf{T}, \mathcal{P})$ be a completion system. By $S + aRb$ (resp. $S + agx$), where a is a node in $\mathbf{T}$ and b (resp. x) is fresh in S, we denote the completion system S' which can be obtained from S as follows:

- If $R \in N_{aF}$ and a has an R-neighbor b' (resp. $g \in N_{cF}$ and a has a g-successor x'), then rename b' in $\mathbf{T}$ with b (resp. x' in $\mathbf{T}$ and $\mathcal{P}$ with x).

- Otherwise, augment $\mathbf{T}$ by a new successor b of a (resp. x of a) and set $\mathcal{L}(a, b) = R$ (resp. $\mathcal{L}(a, x) = g$).

When nesting the +-operation, we ommit brackets writing, e.g., $S + aRb + bRc$ for $(S + aRb) + bRc$. Let $u = f_1 \cdots f_n g$ be a path. By $S + aux$, where x is fresh in S, we denote the completion system S' which can be obtained from S as follows: Let $b_1, \dots, b_n$ be distinct objects which are fresh in S. Set

$$S' := S + af_1b + b_1f_2b_2 + \cdots + b_{n-1}f_nb_n + b_ngx.$$

The completion rules can be found in Figure 12. With *roles*(D) in the R*ch* rule, we denote the set of role names and predicate roles used (directly or as inverse) in the input concept D. The R$\sqcup$ rule is nondeterministic, i.e., it has more than one possible outcome. The algorithm returns *unsatisfiable* only if there is no way to apply the completion rules such that a complete and clash-free completion system is obtained. Intuitively, the algorithm can be thought of as "guessing" the "right" outcome of the R$\sqcup$ rule. The notion "clash" formalizes what it means for a completion system to be contradictory.

Definition 32 (Clash). Let $S = (\mathbf{T}, \mathcal{P})$ be a completion system for a concept D. S is *concrete domain satisfiable* iff the conjunction

$$\zeta_\mathcal{P} = \bigwedge_{P \text{ used in } D} \bigwedge_{(x_1,\dots,x_n)\in\mathcal{P}(P)} (x_1, \dots, x_n) : P$$

R$\sqcap$ if $C_1 \sqcap C_2 \in \mathcal{L}(a)$, $C_1 \notin \mathcal{L}(a)$, or $C_2 \notin \mathcal{L}(a)$
then $\mathcal{L}(a) := \mathcal{L}(a) \cup \{C_1, C_2\}$

R$\sqcup$ if $C_1 \sqcup C_2 \in \mathcal{L}(a)$ and $C_1 \notin \mathcal{L}(a)$ or $C_2 \notin \mathcal{L}(a)$
then $\mathcal{L}(a) := \mathcal{L}(a) \cup \{C\}$ for some $C \in \{C_1, C_2\}$

R$\exists$ if $\exists R.C \in \mathcal{L}(a)$ and, for all general R-neighbors b of a, $C \notin \mathcal{L}(b)$
then set $S := S + aRb$ for a fresh $b \in O_a$ and set $\mathcal{L}(b) := \{C\}$

R$\forall$ if $\forall R.C \in \mathcal{L}(a)$, b is general R-neighbor of a, and $C \notin \mathcal{L}(b)$
then set $\mathcal{L}(b) := \mathcal{L}(b) \cup \{C\}$

Rc if $\exists u_1, \ldots, u_n.P \in \mathcal{L}(a)$ and there exist no $x_1, \ldots, x_n \in O_c$ such that $x_i = \mathit{neighb}(a, u_i)$ for $1 \le i \le n$ and $(x_1, \ldots, x_n) \in \mathcal{P}(P)$
then augment S as follows:
Set $S_0 := S$ and, for each $1 \le i \le n$, set $S_i = S_{i-1} + au_ix_i$ with x_i fresh in S_{i-1}.
Finally, set $S := S_n$ and $\mathcal{P}(P) := \mathcal{P}(P) \cup \{(x_1, \ldots, x_n)\}$

R$\mathcal{R}$ if b is $\exists(u_1, \ldots, u_n), (v_1, \ldots, v_m).P$-neighbor of a and there exist no $x_1, \ldots, x_n, y_1, \ldots, y_m \in O_c$ such that $x_i = \mathit{neighb}(a, u_i)$ for $1 \le i \le n$, $y_i = \mathit{neighb}(b, v_i)$ for $1 \le i \le m$, and $(x_1, \ldots, x_n, y_1, \ldots, y_m) \in \mathcal{P}(P)$
then augment S as follows:
$S_0 := S$; for $1 \le i \le n$, $S_i = S_{i-1} + au_ix_i$ with x_i fresh in S_{i-1}
$S'_0 := S_n$; for $1 \le i \le m$, $S'_i = S'_{i-1} + bv_iy_i$ with y_i fresh in S'_{i-1}.
Set $S := S'_m$ and $\mathcal{P}(P) := \mathcal{P}(P) \cup \{(x_1, \ldots, x_n, y_1, \ldots, y_m)\}$

Rch if $\exists(u_1, \ldots, u_n), (v_1, \ldots, v_m).P \in \mathit{roles}(D)$, $x_i = \mathit{neighb}(a, u_i)$ for $1 \le i \le n$, $y_i = \mathit{neighb}(b, v_i)$ for $1 \le i \le m$, and $(x_1, \ldots, x_n, y_1, \ldots, y_m) \notin \mathcal{P}(P) \cup \mathcal{P}(\overline{P})$
then $\mathcal{P}(P') := \mathcal{P}(P') \cup \{(x_1, \ldots, x_n, y_1, \ldots, y_m)\}$ for a $P' \in \{P, \overline{P}\}$

Figure 12: Completion rules for $\mathcal{ALCRPI}(\mathcal{D})$ on input C_0.

define procedure *sat*(S)
  if S contains a clash **then**
    return *unsatisfiable*
  if S is complete **then**
    return *satisfiable*
  Apply a (possibly nondeterministic) completion rule to S yielding S'
  return *sat*(S')

Figure 13: The *sat* algorithm.

is satisfiable. S is said to contain a *clash* iff there occurs a node $a \in O_a$ in $\mathbf{T}$ such that

1. $\{A, \neg A\} \subseteq \mathcal{L}(a)$ for a concept name A,
2. $g\uparrow \in \mathcal{L}(a)$ and there exists an $x \in O_c$ such that x is g-successor of a, or
3. S is not concrete domain satisfiable.

If S does not contain a clash, S is called *clash-free*. S is called *complete* iff no completion rule is applicable to S.

The completion algorithm (called *sat* from now on) itself can be found in Figure 13 in a pseudo code notation.

In the following, we introduce some notions needed for proving termination of the algorithm. A completion system S' is *derived* from a completion system S if S' can be obtained from S by repeatedly applying completion rules. For a node a in $\mathbf{T}$, let $\ell(a)$ denote the *level* of a in $\mathbf{T}$, i.e., its distance to the root node. With $|C|$, we denote the *size* of a concept C which is defined as the number of symbols (constructors, concept names, role names, concrete feature names, and predicate names) in C. With $rd(C)$, we denote the role depth of a concept C which is defined inductively as follows (for technical reasons, we also define the role depth of *roles*):

1. $rd(A) = rd(\neg A) = 0$ for concept names A,
2. $rd(R) = 1$ for role names R,
3. $rd(\exists(u_1, \dots, u_n), (v_1, \dots, v_m).P)$ is the length of the longest path in the set $\{u_1, \dots, u_n, v_1, \dots, v_m\}$ (where the length of a path $u = f_1 \cdots f_n g$ is $n + 1$),
4. $rd(C_1 \sqcap C_2) = rd(C_1 \sqcup C_2) = \max(rd(C_1), rd(C_2))$,
5. $rd(\exists R.C) = rd(\forall R.C) = \max(rd(R), 1 + rd(C))$,
6. $rd(\exists u_1, \dots, u_n.P)$ is the length of the longest path in $\{u_1, \dots, u_n\}$, and
7. $rd(g\uparrow) = 0$.

Let C be a concept. With $nf(C)$, we denote the number of distinct abstract features used in C. Furthermore, $rex(C)$, denotes the number of concepts in $sub(C)$ of the form $\exists R.D$ with $R \in N_R \setminus N_{aF}$. Let $\mathcal{C}$ be a set of concepts. With $rd(\mathcal{C})$, we denote the maximum role depth of all concepts in $\mathcal{C}$. We set

$$\mathcal{C}|_{\exists cR} := \{C \in \mathcal{C} \mid sub(C) \text{ contains a concept of the form } \exists R.E \text{ with } R \text{ complex role}\}$$

and

$$\mathcal{C}|_{\exists P} := \{C \in \mathcal{C} \mid sub(C) \text{ contains a concept of the form } \exists u_1, \dots, u_n.P\}.$$

For showing termination, we that the depth and outdegree of completion trees constructed by the algorithm is bounded. In order to show the bound on the depth, we prove that the level of abstract nodes having concrete g-successors (for some $g \in N_{cF}$) is bounded. This is important since it implies that, if a node b is a virtual successor of a node a, then the depth of b is bounded. It is not hard to see that this fact is crucial for the boundedness of the depth of completion trees. To show the mentioned bound on the level of objects with concrete successors, we prove that, if $\mathcal{L}(a)$ contains a concept of the form $\exists u_1, \ldots, u_n.P$ or $\exists S.C$ with S complex role, then the level of the node a is bounded. We start with establishing this latter two bounds (one for each concept type).

Lemma 33. *Let $S = (\mathbf{T}, \mathcal{P})$ be a completion system derived from an initial completion system S_D. For all abstract nodes a in $\mathbf{T}$, we have $rd(\mathcal{L}(a)|_{\exists cR}) \leq rd(D) - \ell(a)$.*

Proof The proof is by induction over the number of rule applications. The lemma is obviously true for the initial completion system S_D. For the induction step, we make a case distinction according to the rule applied. R$\sqcap$ and R$\sqcup$ are straightforward since they only add concepts C to labels $\mathcal{L}(a)$ with $rd(C) \leq rd(\mathcal{L}(a))$. R$c$, R$\mathcal{R}$, and R$ch$ are trivial since they do not change node labels at all. Hence, the only interesting cases are R$\exists$ and R$\forall$.

- Assume R$\exists$ is applied to a concept $\exists R.C \in \mathcal{L}(a)$ where $sub(C)$ contains a concept of the form $\exists S.E$ with S complex role. The rule application generates an R-successor b of a and sets $\mathcal{L}(b) = \{C\}$. By induction hypothesis, we have $rd(\exists R.C) \leq rd(D) - \ell(a)$. It follows that $rd(C) \leq rd(D) - \ell(b)$ since $rd(\exists R.C) \geq rd(C) + 1$ ("$\geq$" since R may be a complex role) and $\ell(b) = \ell(a) + 1$.
- Assume R$\forall$ is applied to a concept $\forall R.C \in \mathcal{L}(a)$ adding C to $\mathcal{L}(b)$ where $sub(C)$ contains a concept of the form $\exists S.E$ with S complex role. Since D is in restricted form, $\forall R.C$ is also in restricted form, and, hence, R is not a complex role (see Definition 8). This implies that b is R-neighbor of a and hence $\ell(b) \in \{\ell(a)-1, \ell(a)+1\}$ implying $\ell(b) \leq \ell(a)+1$. By induction hypothesis, $rd(\forall R.C) \leq rd(D) - \ell(a)$. It follows that $rd(C) \leq rd(D) - \ell(b)$ since $rd(\forall R.C) = rd(C) + 1$ ("$=$" since R is not a complex role) and $\ell(b) \leq \ell(a) + 1$. ❑

Lemma 34. *Let $S = (\mathbf{T}, \mathcal{P})$ be a completion system derived from an initial completion system S_D. For all abstract nodes a in $\mathbf{T}$, we have $rd(\mathcal{L}(a)|_{\exists P}) \leq rd(D) - \ell(a)$.*

Proof Straightforward by induction on the number of rule applications, employing the definition of restrictedness (similar to the proof of Lemma 33). ❑

Now for the bound on the level of objects having concrete successors.

Lemma 35. *Let $S = (\mathbf{T}, \mathcal{P})$ be a completion system derived from an initial completion system S_D. Then, for all abstract nodes a and concrete nodes x in $\mathbf{T}$, if $(a, x) \in g^{\mathcal{I}}$, where g is a concrete feature, then $\ell(a) \leq rd(D)$.*

Proof Only the Rc and R$\mathcal{R}$ rules may introduce successors for concrete features. We first treat the Rc rule. Assume that the rule was applied to a concept $\exists u_1, \ldots, u_n.P \in \mathcal{L}(a)$ and generates a g-successor x for an abstract node b, where g is a concrete feature. By Lemma 34, we have $\ell(a) \leq rd(D) - rd(\exists u_1, \ldots, u_n.P)$. Furthermore, by definition of the Rc rule, we have $\ell(b) < \ell(a) + rd(\exists u_1, \ldots, u_n.P)$, and, hence, $\ell(b) < rd(D)$.

Now assume that the R$\mathcal{R}$ rule was applied to an object a and its S-neighbor b. Then either (i) b is successor of a and $\mathcal{L}(a,b) = S$ or (ii) a is successor of b and $\mathcal{L}(b,a) = S^-$. First consider case (i). In this case, $\mathcal{L}(a,b)$ was generated by an application of the R$\exists$ rule to a concept $\exists S.C \in \mathcal{L}(a)$. From Lemma 33, it follows that $\ell(a) \leq rd(D) - rd(\exists S.C)$. Furthermore, we have $\ell(b) = \ell(a) + 1$. Suppose that the rule application generates a g-successor x for an abstract node c, where g is a concrete feature. By definition of the R$\mathcal{R}$ rule, it is easy to see that we have $\ell(c) < \ell(b) + rd(\exists S.C)$. Since $\ell(b) = \ell(a) + 1$, this yields $\ell(c) < \ell(a) + 1 + rd(\exists S.C)$, and, from $\ell(a) \leq rd(D) - rd(\exists S.C)$, we obtain $\ell(c) < rd(D) - rd(\exists S.C) + 1 + rd(\exists S.C)$ which clearly implies $\ell(c) \leq rd(D)$. Case (ii) is analogous. ❑

We can now prove the bounds on the size of completion trees.

Lemma 36. *Let D be an $\mathcal{ALCRPI}(\mathcal{D})$-concept and let $S = (\mathbf{T}, \mathcal{P})$ be a completion system derived from an initial completion system S_D.*

1. *The out-degree of $\mathbf{T}$ is bounded by $nf(D) + rex(D)$ and*
2. *the depth of $\mathbf{T}$ is bounded by $3 * rd(D)$.*

Proof We first prove Point 1. Only applications of the R$\exists$, Rc, and R$\mathcal{R}$ rules may generate successors. The Rc and R$\mathcal{R}$ roles generate only f-successors with f abstract feature. Since, by definition of $\mathbf{T}$, there can be at most one f-successor per node and abstract feature f, applications of the Rc and R$\mathcal{R}$ rules may generate at most $nf(D)$ successors per node. Applications of the R$\exists$ rule may additionally generate R-successors with $R \in N_R \setminus N_{aF}$. However, by definition of R$\exists$, it is easy to see that the number of successors per node generated in this way is bounded by $rex(D)$.

Now for Point 2. We prove the following claim:

$$\text{For all abstract nodes } a \text{ in } \mathbf{T}, rd(\mathcal{L}(a)) \leq 3 * rd(D) - \ell(a). \tag{1}$$

The claim obviously implies $\ell(a) \leq 3 * rd(D)$. The proof is by induction over the number of rule applications. The claim is obviously true for the initial completion system S_D. Now for the induction step. Note that $rd(\mathcal{L}(a)) \leq rd(D)$ for all abstract nodes a in $\mathbf{T}$. This implies that the claim holds true for all nodes a with $\ell(a) \leq 2 * rd(D)$. Hence, we will in the following consider only nodes a with $\ell(a) > 2 * rd(D)$. We make a case distinction according to the rule applied. R$\sqcap$ and R$\sqcup$ are straightforward since they only add concepts C to labels $\mathcal{L}(a)$ with $rd(C) \leq rd(\mathcal{L}(a))$.

- Assume R$\exists$ is applied to a concept $\exists R.C \in \mathcal{L}(a)$ adding C to $\mathcal{L}(b)$. By induction hypothesis, $rd(\exists R.C) \leq 3 * rd(D) - \ell(a)$. It follows that $rd(C) \leq 3 * rd(D) - \ell(b)$ since $rd(\exists R.C) = rd(C) + 1$ and $\ell(b) = \ell(a) + 1$.
- Assume R$\forall$ is applied to a concept $\forall R.C \in \mathcal{L}(a)$ adding C to $\mathcal{L}(b)$. As noted above, we may safely assume $\ell(b) > 2 * rd(D)$. By Lemma 35 and since the maximum length of paths in D is bounded by $rd(D)$, we have that $u^{\mathcal{I}}(b)$ is

undefined for each path u in D.[9] It follows that R is not a virtual R-successor of a, and, hence, $\ell(b) \leq \ell(a) + 1$. We can now argue as in the R$\exists$ case.

- If the Rc rule is applied to a concept $\exists u_1, \ldots, u_n.P \in \mathcal{L}(a)$, then $\ell(a) \leq 3 * \mathit{rd}(D) - \mathit{rd}(\exists u_1, \ldots, u_n.P)$ by induction hypothesis. Hence, by definition of the Rc rule, for all (abstract and concrete) nodes b created by the rule application, we have $\ell(b) \leq 3 * \mathit{rd}(D)$. Since R$c$ does not augment node labels with new concepts, this proves the claim.
- The case of the R$\mathcal{R}$ rule is similar to the Rc rule. ❑

Using the lemma just established, we can now prove termination.

Proposition 37 (Termination). *Let D be an input to the completion algorithm and let $K = (\mathit{nf}(D) + \mathit{rex}(D))^{3*\mathit{rd}(D)}$. The algorithm terminates after at most $\mathcal{O}(|\mathit{sub}(D)| * K + K^2)$ rule applications.*

Proof We first examine the maximum number of applications of the R$\sqcap$, R$\sqcup$, R$\exists$, and R$\forall$ rules. Each such application adds a new concept to a node label. By Lemma 36, there exist at most K nodes. Obviously, the size of each node label is bounded by $|\mathit{sub}(D)|$. Since nodes are never removed from the tree and concepts are never removed from node labels, there may be at most $|\mathit{sub}(D)| * K$ applications of the mentioned rules. It remains to treat applications of the Rc, R$\mathcal{R}$, and Rch rules.

Rc This rule may be applied at most once per concept $\exists u_1, \ldots, u_n.P$ appearing in a node label. Hence, the above considerations imply that there may be at most $|\mathit{sub}(D)| * K$ applications.

R$\mathcal{R}$ R$\mathcal{R}$ may be applied at most once per edge and each node has at most one incoming edge. Hence, the number of R$\mathcal{R}$ applications is bounded by K.

Rch This rule may be be applied at most once per pair of abstract nodes, i.e., at most K^2 times.

Taking the above observations together, we obtain the bound stated in the lemma: Applications of the R$\sqcap$, R$\sqcup$, R$\exists$, and R$\forall$ rules yield the first summand, applications of Rch the second, and all remaining applications just yield a constant factor. ❑

We now prove the correctness of the algorithm.

Lemma 38 (Soundness). *If there exists a complete and clash-free completion system $S = (\mathbf{T}, \mathcal{P})$ derived from the initial completion system S_D, then D is satisfiable.*

[9] This does not necessarily hold for nodes b with $\ell(b) > \mathit{rd}(D)$. To see this, note that we may, e.g., have fg as a path in D, $\mathcal{L}(a,b) = f^-$ and $\mathcal{L}(a,x) = g$ with $\ell(a) = \mathit{rd}(D)$ and $\ell(b) = \ell(a) + 1$. Obviously, $(fg)^{\mathcal{I}}(b)$ is defined.

Proof Let $S = (\mathbf{T}, \mathcal{P})$ be as in the lemma. Since S is clash-free, there exists a solution for $\zeta_{\mathcal{P}}$, i.e., a mapping from the set of concrete nodes used in $\mathbf{T}$ to $\Delta_{\mathcal{D}}$. Define the interpretation $\mathcal{I}$ by setting $\Delta_{\mathcal{I}}$ to the set of abstract nodes in $\mathbf{T}$,

$A^{\mathcal{I}}$ to $\{a \mid A \in \mathcal{L}(a)\}$ for all $A \in N_C$,

$R^{\mathcal{I}}$ to $\{(a,b) \mid \mathcal{L}(a,b) = R \text{ or } \mathcal{L}(b,a) = \mathit{Inv}(R)\}$ for all $R \in N_R$, and

$g^{\mathcal{I}}$ to $\{(a, \ (x)) \mid \mathcal{L}(a,x) = g\}$ for all $g \in N_{cF}$.

Considering Properties 2 to 4 of completion trees, it is obvious that $\mathcal{I}$ is well-defined.We first show the following claim.

Claim: For all $a, b \in \Delta_{\mathcal{I}}$ and roles $R \in \widehat{\mathcal{R}}$, we have $(a,b) \in R^{\mathcal{I}}$ iff b is general R-neighbor of a.

We make a case distinction according to the type of R.

1. $R \in N_R$. Then b is a general R-neighbor of a iff b is R-neighbor of a. By definition of $\mathcal{I}$ and of R-neighbors, b is R-neighbor of a iff $(a,b) \in R^{\mathcal{I}}$.

2. $R = S^-$ with $S \in N_R$. Again, b is a general R-neighbor of a iff b is R-neighbor of a. By definition, b is R-neighbor of a iff a is S-neighbor of b. As in Case 1, a is S-neighbor of b iff $(b,a) \in S^{\mathcal{I}}$. By semantics, $(b,a) \in S^{\mathcal{I}}$ iff $(a,b) \in R^{\mathcal{I}}$ which proves the claim.

3. $R = \exists (u_1, \ldots, u_n), (v_1, \ldots, v_m).P$ is a predicate role. If b is R-neighbor of a, then the non-applicability of the R$\mathcal{R}$ rule ensures that b is also a virtual R-successor of a. Hence, b is general R-neighbor of a iff b is virtual R-successor of b. By definition, b is virtual R-successor of b iff $(*)$ there exist concrete nodes $x_1, \ldots, x_n, y_1, \ldots, y_m$ such that

   - $x_i = \mathit{neighb}(a, u_i)$ for $1 \le i \le n$,
   - $y_i = \mathit{neighb}(b, v_i)$ for $1 \le i \le m$, and
   - $(x_1, \ldots, x_n, y_1, \ldots, y_m) \in \mathcal{P}(P)$.

   We need to show that this is the case iff $(**)$ there exist $\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_m \in \Delta_{\mathcal{D}}$ such that

   - $u_i^{\mathcal{I}}(a) = \alpha_i$ for $1 \le i \le n$,
   - $v_i^{\mathcal{I}}(b) = \beta_i$ for $1 \le i \le m$, and
   - $(\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_m) \in P^{\mathcal{D}}$.

   This proves the claim, since, by semantics, we have $(**)$ iff $(a,b) \in R^{\mathcal{I}}$. The direction from $(*)$ to $(**)$ is straightforward by definition of $\mathcal{I}$. Now for the direction from $(**)$ to $(*)$. Assume that $(*)$ holds. By Case 1, this implies the existence of concrete nodes $x_1, \ldots, x_n, y_1, \ldots, y_m$ such that $x_i = \mathit{neighb}(a, u_i)$ for $1 \le i \le n$ and $y_i = \mathit{neighb}(b, v_i)$ for $1 \le i \le m$. Since the Rch rule is not applicable to S and $R \in \mathit{roles}(D)$, we have either $(x_1, \ldots, x_n, y_1, \ldots, y_m) \in \mathcal{P}(P)$ or

$(x_1, \ldots, x_n, y_1, \ldots, y_m) \in \mathcal{P}(\overline{P})$. The latter implies $(\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_m) \in \overline{P}^{\mathcal{D}}$ which is a contradiction. Hence, we conclude $(x_1, \ldots, x_n, y_1, \ldots, y_m) \in \mathcal{P}(P)$.

4. $R = S^-$ with S predicate role. As in Case 3, b is general R-neighbor of a iff b is virtual R-successor of a. By definition, b is virtual R-successor of a iff a is virtual S-successor of b. As in Case 3, we conclude that this is the case iff $(a, b) \in S^{\mathcal{I}}$. By semantics, $(b, a) \in S^{\mathcal{I}}$ iff $(a, b) \in R^{\mathcal{I}}$.

This finishes the proof of the claim. By induction over the concept structure, we show that $C \in \mathcal{L}(a)$ implies $a \in C^{\mathcal{I}}$ for all $a \in \Delta_{\mathcal{I}}$ and subconcepts C of D. The induction start, i.e., the case that C is a concept name, is an immediate consequence of the definition of $\mathcal{I}$. For the induction step, we make a case distinction according to the topmost constructor in C.

- $C = \neg E$. Since D is in negation normal form, E is a concept name. Since S is clash-free, $E \notin \mathcal{L}(a)$ and, by definition of $\mathcal{I}$, $a \notin E^{\mathcal{I}}$. Hence, $a \in (\neg E)^{\mathcal{I}}$.

- $C = C_1 \sqcap C_2$. Since the R$\sqcap$ rule is not applicable to S, we have $\{C_1, C_2\} \subseteq \mathcal{L}(a)$. By induction, $a \in C_1^{\mathcal{I}}$ and $a \in C_2^{\mathcal{I}}$, which implies $a \in (C_1 \sqcap C_2)^{\mathcal{I}}$.

- $C = C_1 \sqcup C_2$. Similar to the previous case.

- $C = \exists R.E$. Since the R$\exists$ rule is not applicable to S, there exists an abstract node b in $\mathbf{T}$ such that b is general R-neighbor of a in $\mathbf{T}$ and $E \in \mathcal{L}(b)$. The above claim yields $(a, b) \in R^{\mathcal{I}}$. By induction, we have $b \in E^{\mathcal{I}}$. Hence, we conclude $a \in (\exists R.E)^{\mathcal{I}}$.

- $C = \forall R.E$. Let $b \in \Delta_{\mathcal{I}}$ such that $(a, b) \in R^{\mathcal{I}}$. By the above claim, b is a general R-neighbor of a in $\mathbf{T}$. Since the R$\forall$ rule is not applicable to S, we have $E \in \mathcal{L}(b)$. By induction, it follows that $b \in E^{\mathcal{I}}$. Since this holds for all b, we can conclude $a \in (\forall R.E)^{\mathcal{I}}$.

- $C = \exists u_1, \ldots, u_n.P$. For each i with $1 \leq i \leq n$, the following holds: Since the Rc rule is not applicable to S, there exist abstract nodes $b_1, \ldots, b_n$ in $\mathbf{T}$ such that b_1 is f_1-neighbor of a, b_i is f_i-neighbor of b_{i-1} for $1 < i \leq n$, and there exists a concrete node x_i such that x_i is g-successor of b_n. By definition of $\mathcal{I}$, we have $f_1^{\mathcal{I}}(a) = b_1$, $g_i^{\mathcal{I}}(b_n) = \;(x_i)$, and $f_i^{\mathcal{I}}(b_{i-1}) = b_i$ for $1 < i \leq n$. Furthermore, we have $(x_1, \ldots, x_n) \in \mathcal{P}(P)$ and since is a solution for $\zeta_{\mathcal{P}}$, $(\;(x_1), \ldots, \;(x_n)) \in P^{\mathcal{D}}$. Summing up, $a \in (\exists u_1, \ldots, u_n.P)^{\mathcal{I}}$.

- $C = g\uparrow$. Since S is clash-free, a has no g-successor x in $\mathbf{T}$. By definition of $\mathcal{I}$, $g^{\mathcal{I}}(a)$ is undefined and hence $a \in (g\uparrow)^{\mathcal{I}}$.

Since $D \in \mathcal{L}(a_0)$ for the root a_0 of $\mathbf{T}$, we have $D^{\mathcal{I}} \neq \emptyset$ and hence $\mathcal{I}$ is a model for D. ❑

It remains to prove completeness.

Lemma 39 (Completeness). *For any satisfiable $\mathcal{ALCRPI}(\mathcal{D})$-concept D, the expansion rules can be applied such that they yield a complete and clash-free completion system for D.*

Proof Let $\mathcal{I} = (\Delta_\mathcal{I}, \cdot^\mathcal{I})$ be a model for D. We use this model to "guide" the application of the non-deterministic completion rule R$\sqcup$ such that a complete and clash-free completion system for D is obtained. A completion system $S = (\mathbf{T}, \mathcal{P})$ is called $\mathcal{I}$-*compatible* iff there exists a mapping π from the abstract nodes in $\mathbf{T}$ to $\Delta_\mathcal{I}$ and from the concrete nodes in $\mathbf{T}$ to $\Delta_\mathcal{D}$ such that

a) $C \in \mathcal{L}(a) \Rightarrow \pi(a) \in C^\mathcal{I}$

b) b is general R-neighbor of $a \Rightarrow (\pi(a), \pi(b)) \in R^\mathcal{I}$

c) x is g-successor of $a \;\Rightarrow g^\mathcal{I}(\pi(a)) = \pi(x)$

d) $(x_1, \ldots, x_n) \in \mathcal{P}(P) \Rightarrow (\pi(x_1), \ldots, \pi(x_n)) \in P^\mathcal{D}$

for all abstract nodes a, b in $\mathbf{T}$, subconcepts C of D, roles $R \in \widehat{\mathcal{R}}$, concrete nodes $x, x_1, \ldots, x_n$ in $\mathbf{T}$, concrete features g, and predicates $P \in \Phi_\mathcal{D}$.

Claim: If a completion system S is $\mathcal{I}$-compatible and a rule $\mathcal{R}$ is applicable to S, then it can be applied such that it yields an $\mathcal{I}$-compatible completion system S'.

Let S be an $\mathcal{I}$-compatible completion system, let π be a function satisfying a) to d), and let $\mathcal{R}$ be a completion rule applicable to S. We make a case distinction according to the type of $\mathcal{R}$.

R$\sqcap$ Since the rule is applicable, there exists an abstract node a such that $C_1 \sqcap C_2 \in \mathcal{L}(a)$. By a), this implies $\pi(a) \in (C_1 \sqcap C_2)^\mathcal{I}$ and hence $\pi(a) \in C_1^\mathcal{I}$ and $\pi(a) \in C_2^\mathcal{I}$. Obviously, π satisfies a) to d) w.r.t. the obtained completion system S'.

R$\sqcup$ There exists an abstract node a such that $C_1 \sqcup C_2 \in \mathcal{L}(a)$. This implies $\pi(a) \in C_1^\mathcal{I}$ or $\pi(a) \in C_2^\mathcal{I}$. Hence, the rule can be applied such that π satisfies a) to d) w.r.t. the obtained completion system S'.

R$\exists$ There exists an abstract node a such that $\exists R.C \in \mathcal{L}(a)$. By a), this implies $\pi(a) \in (\exists R.C)^\mathcal{I}$ and hence, there exists an $s \in \Delta_\mathcal{I}$ such that $(\pi(a), s) \in R^\mathcal{I}$ and $s \in C^\mathcal{I}$.

- First assume that either R is a role or R is a feature and a does not have an R-neighbor in S. The R$\exists$ rule generates a new abstract node b with $\mathcal{L}(b) = \{C\}$ such that b is an R-successor of a yielding a new completion system S'. Define π' as $\pi \cup \{b \mapsto s\}$. Obviously, π' satisfies a), c), and d) w.r.t. S'. By definition of general R-neighbors, π' satisfies b) w.r.t. S'.
- Now assume that R is an abstract feature and a does already have an R-neighbor b in S. Then, the R$\exists$ rule consistently renames b to some new name c (c.f. the "+" operation) and sets $\mathcal{L}(c) := \mathcal{L}(c) \cup \{C\}$. Define π' as $\pi \cup \{c \mapsto \pi(b)\}$. Since b) holds for π w.r.t. S and by definition of π', we have

$(\pi'(a), \pi'(c)) \in R^{\mathcal{I}}$. Since abstract features are interpreted as functions, we have $\pi'(c) = s$ implying $\pi'(c) \in C^{\mathcal{I}}$. Hence, π' satisfies a) to d) w.r.t. the obtained completion system S'.

R$\forall$ There exist abstract nodes a and b such that $\forall R.C \in \mathcal{L}(a)$, b is a general R-neighbor of a, and $C \notin \mathcal{L}(b)$. By a), b), and semantics, this implies $\pi(a) \in (\forall R.C)^{\mathcal{I}}$, $(\pi(a), \pi(b)) \in R^{\mathcal{I}}$, and $\pi(b) \in C^{\mathcal{I}}$. The rule application adds C to $\mathcal{L}(b)$. Obviously, π satisfies a) to d) w.r.t. the obtained completion system S'.

Rc There exists an abstract node a such that $\exists u_1, \ldots, u_n.P \in \mathcal{L}(a)$ with $u_i = f_1^{(i)} \cdots f_{k_i}^{(i)} g_i$ for $1 \leq i \leq n$. By a), this implies $\pi(a) \in (\exists u_1, \ldots, u_n.P)^{\mathcal{I}}$. Hence, there exist $s_j^{(i)} \in \Delta_{\mathcal{I}}$ for $1 \leq i \leq n$ and $1 \leq j \leq k_i$ and $\alpha_1, \ldots, \alpha_n \in \Delta_{\mathcal{D}}$ such that

- $(\pi(a), s_1^{(i)}) \in (f_1^{(i)})^{\mathcal{I}}$ for $1 \leq i \leq n$,
- $(s_{j-1}^{(i)}, s_j^{(i)}) \in (f_j^{(i)})^{\mathcal{I}}$ for $1 \leq i \leq n$ and $1 < j \leq k_i$,
- $g_i^{\mathcal{I}}(s_{k_i}^{(i)}) = \alpha_i$ for $1 \leq i \leq n$, and
- $(\alpha_1, \ldots, \alpha_n) \in P^{\mathcal{D}}$.

After the application of the R$\exists$ rule, there exist abstract nodes $b_j^{(i)}$ for $1 \leq i \leq n$ and $1 \leq j \leq k_i$ and concrete nodes $x_1, \ldots, x_n$ such that

- $b_1^{(i)}$ is $f_1^{(i)}$-neighbor of a for $1 \leq i \leq n$,
- $b_j^{(i)}$ is $f_j^{(i)}$-neighbor of $b_{j-1}^{(i)}$ for $1 \leq i \leq n$ and $1 < j \leq k_i$,
- x_i is g_i-successor of $b_{k_i}^{(i)}$ for $1 \leq i \leq n$, and
- $(x_1, \ldots, x_n) \in \mathcal{P}(P)$.

We call the completion system obtained by rule application S'. Define π' by extending π as follows: (i) for $1 \leq i \leq n$ and $1 \leq j \leq k_i$, set $\pi'(b_j^{(i)}) := s_j^{(i)}$; (ii) for $1 \leq i \leq n$, set $\pi(x_i) := \alpha_i$.[10] We need to show that π' satisfies a) to d) w.r.t. the new completion system S'. First, we show the following:

If, during the rule application, an abstract object c is renamed to $b_j^{(i)}$ (resp. a concrete object y to x_i), then we have $\pi'(b_j^{(i)}) = \pi(c)$ (resp. $\pi'(x_i) = \pi(y)$). $(*)$

For assume that an object b is renamed to $b_j^{(i)}$. This implies that there exists an object d such that b is $f_j^{(i)}$-neighbor of d in S (d is either a or $b_{j-1}^{(i)}$). Since (i) s satisfies b) w.r.t. S, (ii) $f^{\mathcal{I}}(\pi(d)) = \pi(s_j^{(i)})$, and (iii) features are interpreted as functions, we have $\pi(b) = s_j^{(i)}$. By definition of $\mathcal{I}$, it follows that $\pi(b) = \pi'(b_j^{(i)})$ (the case with y and x_i is analogous).

[10]Note that existing objects may be renamed due to the use of the "+" operation. We assume that, if the "+" operation renamed an object a to b, then the object name a is never "reintroduced" afterwards (and similar for concrete objects). Hence, π' really is an extension of π.

Since the rule application adds no new concepts to node labels, $(*)$ implies that π' satisfies a) w.r.t. S'. Similarly, b) and d) are immediate consequences of $(*)$ and the definition of $\mathcal{I}$. (note that the rule application may generate new virtual R-successor relationships for some abstract nodes a and b and a complex role R). Property c) is satisfied by π' by definition.

R$\mathcal{R}$ There exist abstract nodes a, b such that b is R-neighbor of a with $R = \exists(u_1, \ldots, u_n), (v_1, \ldots, v_m).P$, i.e., either b is an R-successor of a or a is an R^--successor of b. In any case, b) yields $(\pi(a), \pi(b)) \in R^{\mathcal{I}}$. We proceed analogous to the Rc case.

Rch There exist abstract nodes a, b and a predicate role

$$\exists(u_1, \ldots, u_n), (v_1, \ldots, v_m).P \in \mathit{roles}(D)$$

such that $x_i = \mathit{neighb}(a, u_i)$ for $1 \leq i \leq n$ and $y_i = \mathit{neighb}(b, v_i)$ for $1 \leq i \leq m$. The rule application adds $(x_1, \ldots, x_n, y_1, \ldots, y_n)$ either to $\mathcal{P}(P)$ or to $\mathcal{P}(\overline{P})$. By semantics, we have either

$$(\pi(x_1), \ldots, \pi(x_n), \pi(y_1), \ldots, \pi(y_m)) \in P^{\mathcal{D}}$$

or

$$(\pi(x_1), \ldots, \pi(x_n), \pi(y_1), \ldots, \pi(y_m)) \in \overline{P}^{\mathcal{D}}.$$

Hence, the rule Rch can be applied such that π satisfies a) to d) w.r.t. the obtained completion system S'.

It remains to show that the lemma is a consequence of the above claim. Let $S_D = (\mathbf{T}_D, \mathcal{P}_\emptyset)$ be the initial completion system for D and let a_0 be the node in $\mathbf{T}_D$. Set $\pi(a_0)$ to s for an $s \in D^{\mathcal{I}}$. Obviously, π satisfies a) to d) and hence S_D is $\mathcal{I}$-compatible. By the claim, the completion rules can be applied such that only $\mathcal{I}$-compatible completion systems are obtained. By Lemma 37, every sequence of rule applications terminates yielding a complete completion system. Hence, we can obtain a complete and $\mathcal{I}$-compatible completion system $S = (\mathbf{T}, \mathcal{S})$ by rule application. It remains to show that this implies the clash-freeness of S. Let π be a mapping for S satisfying a) to d).

1. S does not contain a clash of the form $\{A, \neg A\} \subseteq \mathcal{L}(a)$ since, together with a), this would imply $\pi(a) \in A^{\mathcal{I}} \cap (\neg A)^{\mathcal{I}}$ which is impossible.

2. It needs to be shown that, whenever $g\uparrow \in \mathcal{L}(a)$, then there exists no g-successor x of a. Assume to the contrary that there exists an abstract object a, a concrete object x, and a concrete feature g such that $g\uparrow \in \mathcal{L}(a)$ and x is g-successor of a in $\mathbf{T}$. By a), we have $\pi(a) \in (g\uparrow)^{\mathcal{I}}$. By c), we have $g^{\mathcal{I}}(\pi(a)) = \pi(x)$ which is a contradiction.

3. It remains to show that S is concrete domain satisfiable, i.e., that the predicate conjunction $\zeta_{\mathcal{P}}$ is satisfiable. However, using d), it is straightforward to show that the "concrete part" of π is a solution for $\zeta_{\mathcal{P}}$.

❑

Satisfiability w.r.t. TBoxes can be reduced to satisfiability without TBoxes by using *unfolding* [21]. Unfolding a concept C w.r.t. a TBox $\mathcal{T}$ means iteratively replacing concept names in C by their definitions given in $\mathcal{T}$ (unfolding obviously terminates since $\mathcal{T}$ is acyclic). This yields a concept C' which is satisfiable w.r.t. $\mathcal{T}$ iff C is satisfiable w.r.t. $\mathcal{T}$. Together with Lemmas 37, 38, and 39 and the fact that subsumption can be reduced to satisfiability, this gives the following result.

Theorem 40. *Satisfiability and subsumption of $\mathcal{ALCRPI}(\mathcal{D})$-concepts w.r.t. TBoxes are decidable.*

The complexity of the presented algorithm is analyzed in the next section.

4.2 Acyclic TBoxes and Complexity

In this section, we modify the algorithm introduced in the previous section to directly take into account TBoxes (instead of using unfolding) and then analyze the complexity of the modified algorithm. The modification technique we employ was introduced in [19], where it was used to prove that many PSPACE tableau algorithms for deciding concept satisfiability (e.g., for $\mathcal{ALC}$-concepts) can be modified to decide satisfiability of concepts w.r.t. TBoxes such that their PSPACE complexity is preserved. First, the TBox has to be converted to a certain normal form.

Definition 41 (Simple TBoxes). 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 constructor (i.e., it is of the form $\neg A$, $A_1 \sqcap A_2$, $A_1 \sqcup A_2$, $\exists R.A$, $\forall R.A$, $\exists u_1, \ldots, u_n.P$, or $g\uparrow$, where A, A_1, and A_2 are concept names).
- 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 is proved in [19].

Lemma 42. *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, any model for $\mathcal{T}$ can be extended to a model for $\mathcal{T}'$.*

This notion of equivalence is necessary since the translation to simple form may remove concept names and add additional ones. A short comment on what is meant by "extended" is appropriate. Let $\mathcal{T}$ be a TBox, $\mathcal{T}'$ the result of converting it to simple form, and $\mathcal{I}$ be a model for $\mathcal{T}$. We can construct a model for $\mathcal{T}'$ from $\mathcal{I}$ by setting $A^{\mathcal{I}}$ to an appropriate value for all concept names A that have been introduced in the conversion of $\mathcal{T}$ to $\mathcal{T}'$. Defining a model for $\mathcal{T}$ from a model of $\mathcal{T}'$ works similar (additionally interprete all variables that have been eliminated during the conversion of $\mathcal{T}$ to $\mathcal{T}'$).

We now modify the *sat* algorithm from Section 4.1 to decide the satisfiability of concept names A w.r.t. simple TBoxes $\mathcal{T}$. Using the modified algorithm, it is obviously also possible to decide the satisfiability of arbitrary 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 (the concept name is *not* eliminated during conversion, see [19]) and start the algorithm with $(A, \mathcal{T}')$ where $\mathcal{T}'$ is the newly obtained TBox. The modified algorithm works on completion trees of a restricted form since node labels may only contain concept names.

Definition 43 (Modified Completion Algorithm). Let A be a concept name and $\mathcal{T}$ be a simple TBox. Making use of the existing *sat* algorithm, the algorithm *tbsat* is defined as follows.

1. Modify the completion rules of *sat* as follows: In the premise of each completion rule, substitute "$C \in \mathcal{L}(a)$" by "$A \in \mathcal{L}(a)$ and $A \doteq C \in \mathcal{T}$" and analogously for "$C \notin \mathcal{L}(a)$". E.g., in the conjunction rule, "$C_1 \sqcap C_2 \in \mathcal{L}(a)$" is replaced by "$A \in \mathcal{L}(a)$ and $A \doteq C_1 \sqcap C_2 \in \mathcal{T}$".

2. Start the *sat* algorithm with the initial completion system $S_A = (\mathbf{T}_A, \mathcal{P}_\emptyset)$ as defined in Section 4.1. Use the modified rules for the *sat* run.

In the following, we investigate the soundness, completeness, termination, and complexity of the modified algorithm. To do this, we need to extend the notion of size and of subconcepts to TBoxes: For a TBox $\mathcal{T}$, $|\mathcal{T}|$ denotes the *size* of $\mathcal{T}$ and is defined as

$$|\mathcal{T}| = \sum_{A \doteq C \in \mathcal{T}} |C|.$$

Furthermore, $sub(\mathcal{T})$ denotes the set of subconcepts used in $\mathcal{T}$ and is defined as

$$sub(\mathcal{T}) = \bigcup_{A \doteq C \in \mathcal{T}} sub(C).$$

We argue that the *tbsat* algorithm started with input $A, \mathcal{T}$ performs exactly the same steps as the *sat* algorithm started on the concept C which is the result of unfolding A w.r.t. $\mathcal{T}$. Because of this, we give a precise definition of the notion unfolding. In the following, we generally assume that, if $A, \mathcal{T}$ is an input to *tbsat*, then $A \in sub(\mathcal{T})$. This can be done w.l.o.g. since, if $A \notin sub(\mathcal{T})$, $\mathcal{T}$ can be extended by a new concept definition $A' \doteq A$, where $A' \neq A$ and $A' \notin sub(\mathcal{T})$.

Definition 44 (Unfolding). Let $\mathcal{T}$ be a TBox. A concept name A is called *defined* in $\mathcal{T}$ if A appears on the left-hand side of a concept definition in $\mathcal{T}$ and *undefined* otherwise. A concept C is called *unfolded* w.r.t. $\mathcal{T}$ iff every concept name in C is undefined in $\mathcal{T}$. Given a concept C and a TBox $\mathcal{T}$, C can be converted to a concept C' which is (i) unfolded w.r.t. $\mathcal{T}$ and (ii) satisfiable w.r.t. $\mathcal{T}$ iff C is satisfiable w.r.t. $\mathcal{T}$ by using the following *unfolding algorithm*:

define procedure *unfold*($\mathcal{T}$)
  while C contains a concept name A defined in $\mathcal{T}$ **do**
    Let $A \doteq E \in \mathcal{T}$.
    Replace each occurrence of A in C with E.
  return C

In Section 4.1, we introduced several measures on concpets for proving termination. The following lemma clarifies the relation between these measures and unfolding.

Lemma 45. *Let A be a concept name and $\mathcal{T}$ be a simple TBox. If C is the result of unfolding A w.r.t. $\mathcal{T}$, then*

1. $\mathit{nf}(C) \leq |\mathcal{T}|$,
2. $\mathit{rex}(C) \leq |\mathcal{T}|$,
3. $\mathit{rd}(C) \leq |\mathcal{T}|$, *and*
4. $|\mathit{sub}(C)| \leq |\mathcal{T}|$.

Proof Point 1 is trivial and Property 2 is an immediate consequence of Property 4. Hence, we concentrate on the proof of Properties 3 and 4. For Property 3, 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 $\mathcal{T}$.

Now for Property 5. The property is proved by defining an injection I from $\mathit{sub}(C)$ to $\mathit{sub}(\mathcal{T})$. The existence of such an injection implies Property 5 since, obviously, $|\mathit{sub}(\mathcal{T})| \leq |\mathcal{T}|$. Assume that the concepts in $\mathit{sub}(\mathcal{T})$ are ordered by a total order $\prec$. For a concept set $\Psi \subseteq \mathit{sub}(\mathcal{T})$, $\min(\Psi)$ denotes the concept in Ψ which is minimal w.r.t. $\prec$. Define a function I from $\mathit{sub}(C)$ to $\mathit{sub}(\mathcal{T})$ as follows:

$$I(E) := \min\{F \in \mathit{sub}(\mathcal{T}) \mid \mathit{unfold}(F, \mathcal{T}) = E\}.$$

We show that I is total and injective.

- Let k be the number of steps the **while** loop in the unfolding algorithm makes to compute C and let C_i ($0 \leq i \leq k$) denote the concept C after the i'th loop, i.e., $C_0 = A$ and $C_k = C$. To prove totality, we establish the following claim:

  Claim: For all $1 \leq i \leq k$, and for all $E \in \mathit{sub}(C_i)$, there exists an $F \in \mathit{sub}(\mathcal{T})$ such that $\mathit{unfold}(E, \mathcal{T}) = \mathit{unfold}(F, \mathcal{T})$.

  The claim implies totality since, for all concepts $E \in \mathit{sub}(C) = \mathit{sub}(C_k)$, we have $\mathit{unfold}(E, \mathcal{T}) = E$. The proof of the claim is by induction over i. For $i = 0$, the claim trivially holds since $C_0 = A$ and we assume that $A \in \mathit{sub}(\mathcal{T})$. Now for the induction step. Assume that, in the the i'th step, a concept name A' has been replaced by a concept F. Let $E \in \mathit{sub}(C_{i+1}) \setminus \mathit{sub}(C_i)$. Then we have one of the following two cases:

  - $E \in \mathit{sub}(F)$. This implies $E \in \mathit{sub}(\mathcal{T})$, and, hence, E satisfies the claim.
  - $E \notin \mathit{sub}(F)$. Then there exists an E' in $\mathit{sub}(C_i)$ such that E can be obtained from E' by substituting an occurrence of A' in E' by F. Obviously, $\mathit{unfold}(E, \mathcal{T}) = \mathit{unfold}(E', \mathcal{T})$. Since $E' \in \mathit{sub}(C_i)$ satisfies the claim by induction hypothesis, $E \in \mathit{sub}(C_{i+1})$ does also satisfy the claim.

- Assume that I is not injective, i.e., there exist two concepts $E, E' \in$ *sub(C)* with $E \neq E'$ such that $I(E) = I(E') = F$. By definition of I, this implies $\mathit{unfold}(F, \mathcal{T}) = E$ and $\mathit{unfold}(F, \mathcal{T}) = E'$. Since $E \neq E'$ and unfolding is deterministic, this is obviously impossible.

❑

We may now establish correctness and termination of the modified algorithm.

Proposition 46. *Let satisfiability of $\mathcal{D}$ be in* NP *and $(A, \mathcal{T})$ be the input to the tbsat algorithm. Then tbsat terminates after $\mathcal{O}(2^{d|\mathcal{T}|})$ rule applications returning "satisfiable" if A is satisfiable w.r.t. $\mathcal{T}$ and "unsatisfiable" otherwise, where d is a constant.*

Proof Let C be the result of unfolding A w.r.t. $\mathcal{T}$. C is in NNF since $\mathcal{T}$ is in simple form. A *run* of the *tbsat* (resp. *sat*) algorithm on $(A, \mathcal{T})$ (resp. on C) is a sequence of completion rules as applied by the algorithm if started with input $(A, \mathcal{T})$ (resp. with input C). By induction over the number of rule applications, it is straightforward to show that the set of runs of *tbsat* on $(A, \mathcal{T})$ is identical to the set of runs of *sat* on C: 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. Let $K = (\mathit{nf}(C) + \mathit{rex}(C))^{3*\mathit{rd}(C)}$. By Proposition 37, the algorithm terminates after at most $\mathcal{O}(|\mathit{sub}(C)| * K + K^2)$ rule applications. By Lemma 45, this implies that *tbsat* terminates after

$$\mathcal{O}(|\mathcal{T}| * (2|\mathcal{T}|)^{3|\mathcal{T}|} + (2|\mathcal{T}|)^{6|\mathcal{T}|})$$

rule applications which obviously implies the bound given in the lemma. Furthermore, soundness and completeness are immediate consequences of the equivalence of run sets.

❑

Finally, the upper bound for satisfiability and subsumption of $\mathcal{ALCRPI}(\mathcal{D})$-concepts can be given.

Theorem 47. *If satisfiability of the concrete domain $\mathcal{D}$ is in* NP, *satisfiability and subsumption of $\mathcal{ALCRPI}(\mathcal{D})$-concepts w.r.t. TBoxes can be decided in nondeterministic exponential time.*

Proof By Proposition 46, *tbsat* decides satisfiability of $\mathcal{ALCRPI}(\mathcal{D})$-concepts w.r.t. TBoxes and terminates after exponentially many rule applications. During its run, *tbsat* constructs a completion system $S = (\mathbf{T}, \mathcal{P})$. After each rule application, the predicate conjunction $\zeta_{\mathcal{P}}$ induced by $\mathcal{P}$ has to be tested for satisfiability. Since the satisfiability test for finite predicate conjunctions is in NP, it remains to show that the size of $\zeta_{\mathcal{P}}$ is at most exponential in the size of the input TBox $\mathcal{T}$. This is, however, obvious since each rule application adds at most one tuple to $\mathcal{P}$. ❑

Since the set of runs *tbsat* may perform on an input $A, \mathcal{T}$ is identical to the set of runs *sat* may perform on the result C of unfolding A w.r.t. $\mathcal{T}$, there seems to exist an alternative way to obtain Theorem 47: Conjecturing that

- unfolding a concept C w.r.t. a (not necessarily simple) TBox $\mathcal{T}$ can be done in time exponential in $|C| + |\mathcal{T}|$, and
- the result from Lemma 45 can be generalized in an appropriate way to the unfolding of (possibly complex) concepts w.r.t. (not-necessarily simple) TBoxes,

we claim that the same complexity result can be proved by using unfolding as a preprocessing step to the *sat* algorithm (without defining simple TBoxes). This is somewhat surprising since unfolding is usually believed to be "harmful" w.r.t. complexity and it is well-known that unfolding may lead to an exponential blow-up in concept size. In our case, unfolding is nevertheless "harmless" since it is not the concept size which is crucial for the complexity of the presented algorithm, but the measures given in Lemma 45. However, we prefer the use of simple TBoxes since, in our opinion, it is far more elegant and more closely related to the techniques used in implementations of DL systems (see, e.g., [3]).

5 Undecidability of $\mathcal{ALCIF}$

The description logic $\mathcal{ALCF}(\mathcal{D})$ is the extension of $\mathcal{ALC}(\mathcal{D})$ with so-called feature agreements and feature disagreements. In [20], it is proved that satisfiability of $\mathcal{ALCF}(\mathcal{D})$-concepts is PSPACE-complete. The algorithm used to establish the upper bound shows that it is natural to consider concrete domains in combination with feature (dis)agreements since the algorithmic treatment is very similar (see also [13]). It is hence also natural to consider the description logic $\mathcal{ALCRPIF}(\mathcal{D})$ which is the extension of $\mathcal{ALCRPI}(\mathcal{D})$ with feature (dis)agreements. However, in this section, we show that concept satisfiability is already undecidable for the fragment $\mathcal{ALCIF}$, i.e., for $\mathcal{ALC}$ with inverse roles and feature (dis)agreements.

Definition 48 (Feature (dis)agreement). Let $v_1 = f_1^{(1)} \cdots f_n^{(1)}$ and $v_2 = f_1^{(2)} \cdots f_m^{(2)}$ be sequences of abstract features. A *feature agreement* is an expression of the form $v_1{\downarrow}v_2$. A *feature disagreement* is an expression of the form $v_1{\uparrow}v_2$. The semantics of feature agreements and disagreements is defined as follows:

$$(v_1{\downarrow}v_2)^{\mathcal{I}} := \{a \in \Delta_{\mathcal{I}} \mid \exists b \in \Delta_{\mathcal{I}}.v_1^{\mathcal{I}}(a) = b \wedge v_2^{\mathcal{I}}(a) = b\}$$
$$(v_1{\uparrow}v_2)^{\mathcal{I}} := \{a \in \Delta_{\mathcal{I}} \mid \exists b_1, b_2 \in \Delta_{\mathcal{I}}.v_1^{\mathcal{I}}(a) = b_1 \wedge v_2^{\mathcal{I}}(a) = b_2 \wedge b_1 \neq b_2\}$$

The undecidability of $\mathcal{ALCIF}$ if proved by a reduction of the well-known, undecidable domino problem (see, e.g., [5] and [18]). 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 the first quadrant of the plane without holes or overlapping, such that adjacent tiles have identical colors on their touching edge (rotation of the tiles is not allowed).

Definition 49 (Domino System). Let $\mathcal{D} = (D, H, V)$ be a *domino system*, where D is a finite set of *tile types* and $H, V \subseteq D \times D$. A mapping $\tau : \mathbb{N}^2 \to D$ is a *solution* of $\mathcal{D}$ if

$$
\begin{aligned}
\mathit{Grid} &= \exists f^{-}.\top \ \sqcap \ \forall f^{-}.xy{\downarrow}yx \ \sqcap \ \forall f^{-}.(f{\downarrow}xf \sqcap f{\downarrow}yf \sqcap f{\downarrow}xyf) \\
\mathit{Tiling} &= (\bigsqcup_{d \in \mathcal{D}} D_d) \sqcap \bigsqcap_{d \in D} \ \bigsqcap_{d' \in \mathcal{D} \setminus \{d\}} \neg(D_d \sqcap D_{d'}) \\
& \quad \bigsqcap_{d \in D} (D_d \to \exists x. \bigsqcup_{(d,d') \in H} D_{d'}) \\
& \quad \bigsqcap_{d \in D} (D_d \to \exists y. \bigsqcup_{(d,d') \in V} D_{d'}) \\
C_{\mathcal{D}} &= \mathit{Grid} \ \sqcap \ \forall f^{-}.\mathit{Tiling}
\end{aligned}
$$

Figure 14: The $\mathcal{ALCIF}$ reduction concept $C_{\mathcal{D}}$.

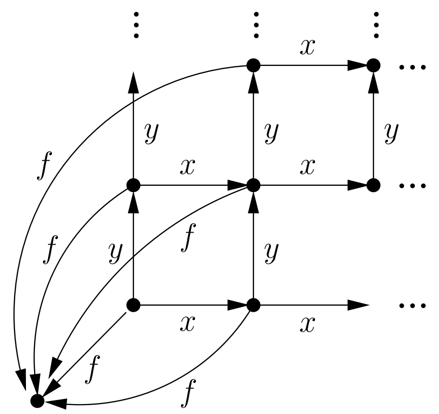

Figure 15: Clipping from a model of $C_{\mathcal{D}}$.

- if $\tau(x, y) = d$ and $\tau(x + 1, y) = d'$ then $(d, d') \in H$, and
- if $\tau(x, y) = d$ and $\tau(x, y + 1) = d'$ then $(d, d') \in V$.

In the following, we reduce the domino problem to satisfiability of $\mathcal{ALCIF}$-concepts. Given a domino system $\mathcal{D}$, the reduction concept $C_{\mathcal{D}}$ is such that (i) models of $C_{\mathcal{D}}$ have the form of a two-side infinite grid, (ii) every node of the grid is an instance of exactly one of the concept names D_d with $d \in D$ (representing tile types), and (iii) the horizontal and vertical conditions V and H are satisfied. The reduction concept can be found in Figure 14 and a sample $C_{\mathcal{D}}$ model can be found in Figure 15. Again, the equalities in the figure are used as an abbreviation and are not intended to denote concept definitions. The symbols x, y, and f denote (abstract) features. In the reduction, the *Grid* concept generates the grid and the *Tiling* concept ensures that the condition listed as (ii) and (iii) are satisfied. We now formally proof correctness.

Lemma 50. *$C_{\mathcal{D}}$ is satisfiable iff $\mathcal{D}$ has a solution τ.*

Proof Assume that $C_{\mathcal{D}}$ has a model $\mathcal{I} = (\Delta_{\mathcal{I}}, \cdot^{\mathcal{I}})$. We define a solution τ for $\mathcal{D}$. Let

$a \in C_{\mathcal{D}}^{\mathcal{I}}$ and let $b \in (f^{-})^{\mathcal{I}}(a)$ (such a and b exist due to the first conjunct of th *Grid* concept). Define the function π from $\mathbb{N}^2$ to $\Delta_{\mathcal{I}}$ inductively as follows.

1. $\pi(0, 0) = b$
2. if $\pi(i, j) = c$ and $x^{\mathcal{I}}(c) = d$, then $\pi(i + 1, j) = d$
3. if $\pi(i, j) = c$ and $y^{\mathcal{I}}(c) = d$, then $\pi(i, j + 1) = d$

The *Grid* concept ensures that this function is total, i.e., that $x^{\mathcal{I}}$ and $y^{\mathcal{I}}$ are always defined. Finally, we define $\tau(i, j)$ as the $d \in D$ for which $\pi(i, j) \in D_d^{\mathcal{I}}$. Note that, due to the first line of *Tiling*, there exists exactly one such d for each $\pi(i, j)$. It is straightforward to check that τ is well-defined and a solution for $\mathcal{D}$.

Conversely, assume that τ is a solution for $\mathcal{D}$. We define a model $\mathcal{I}$ for $C_{\mathcal{D}}$ as follows:

- $\Delta_{\mathcal{I}} = \mathbb{N}^2 \cup \{\lambda\}$
- $x^{\mathcal{I}} = \{((i, j), (i + 1, j)) \mid i, j \in \mathbb{N}\}$
- $y^{\mathcal{I}} = \{((i, j), (i, j + 1)) \mid i, j \in \mathbb{N}\}$
- $f^{\mathcal{I}} = \{((i, j), \lambda) \mid i, j \in \mathbb{N}\}$
- $D_d^{\mathcal{I}} = \tau^{-1}(d)$ for all $d \in D$

Again, it is straightforward to verify that $\mathcal{I}$ is a model for $\mathcal{D}$. ❑

The following theorem is an immediate consequence of Lemma 50 and the undecidability of the domino problem.

Theorem 51. *Satisfiability of $\mathcal{ALCIF}$-concepts is undecidable.*

6 Conclusion

In this paper, we investigate the complexity of various extensions of the Description Logic $\mathcal{ALC}(\mathcal{D})$. The lower bounds are established using a NExpTime-complete variant of the Post Correspondence Problem together with a (rather natural) concrete domain $\mathcal{P}$ for which reasoning can be done in PTime. More precisely, we prove the following problems to be NExpTime-hard:

1. satisfiability of $\mathcal{ALC}(\mathcal{P})$-concepts w.r.t. TBoxes,
2. satisfiability of $\mathcal{ALCI}(\mathcal{P})$-concepts, and
3. satisfiability of $\mathcal{ALCRP}(\mathcal{P})$-concepts.

As a corresponding upper bound, we show that, if reasoning with a concrete domain $\mathcal{D}$ is in NP, then satisfiability and subsumption of $\mathcal{ALCRPI}(\mathcal{D})$-concepts w.r.t. TBoxes is in NExpTime. Finally, we prove that $\mathcal{ALCRPI}(\mathcal{D})$ cannot be extended by feature (dis)agreements without loosing decidability since the satisfiability of $\mathcal{ALCIF}$-concepts is already undecidable.

As future work, it would be interesting to extend the obtained logics by further constructors such as transitive roles [24] and qualifying number restrictions [12]. There are at least two approaches: Since reasoning with $\mathcal{ALCF}(\mathcal{D})$ is known to be in PSpace [20], one could define extensions of $\mathcal{ALCF}(\mathcal{D})$ trying to obtain an expressive logic with concrete domains 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 for "interesting" concrete domains and that feature (dis)agreements cannot be included without loosing decidability.

Acknowledgments

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 J. Mylopoulos and R. Reiter, editors, *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence IJCAI-91*, pages 452–457, Sydney, Australia, August 24–30, 1991. Morgan Kaufmann Publ. Inc., San Mateo, CA, 1991.

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

[3] 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. DFKI Research Report RR-93-03, Saarbrücken.

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

[5] R. Berger. The undecidability of the dominoe problem. *Memoirs of the American Mathematical Society*, 66, 1966.

[6] E. Börger, E. Grädel, and Y. Gurevich. *The Classical Decision Problem.* Perspectives in Mathematical Logic. Springer-Verlag, Berlin, 1997.

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

[8] F. Donini, M. Lenzerini, D. Nardi, and W. Nutt. The complexity of concept languages. DFKI Research Report RR-95-07, Deutsches Forschungszentrum für Künstliche Intelligenz, Kaiserslautern, 1995.

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

[10] V. Haarslev, C. Lutz, and R. Möller. Defined topological relations in description logics. In A. Cohn, L. Schubert, and S.C.Shapiro, editors, *Principles of Knowledge Representation and Reasoning – Proc. of the Sixth International Conference KR'98*, pages 112–124, Trento, Italy, June 2–5, 1998. Morgan Kaufmann Publ. Inc., San Francicso, CA, 1998.

[11] 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.

[12] B. Hollunder and F. Baader. Qualifying number restrictions in concept languages. In J. Allen, R. Fikes, and E. Sandewall, editors, *Principles of Knowledge Representation and Reasoning – Proc. of the Second International Conference KR'91*, pages 335–346, Cambridge, Mass., April 22–25, 1991. Morgan Kaufmann Publ. Inc., San Mateo, CA, 1991. DFKI Research Report RR-91-03, Kaiserslautern.

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

[14] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, Reading, Mass., 1979.

[15] 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.

[16] I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for expressive description logics. In H. Ganzinger, D. McAllester, and A. Voronkov, editors, *Proceedings of the 6th International Conference on Logic for Programming and Automated Reasoning (LPAR'99)*, number 1705 in Lecture Notes in Artificial Intelligence, pages 161–180. Springer-Verlag, Sept. 1999.

[17] 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.

[18] D. E. Knuth. *The Art of Computer Programming*, volume 3. Addison-Wesley, Reading, Mass., 1973.

[19] C. Lutz. Complexity of terminological reasoning revisited. In H. Ganzinger, D. McAllester, and A. Voronkov, editors, *Proceedings of the 6th International Conference on Logic for Programming and Automated Reasoning (LPAR'99)*, number 1705 in Lecture Notes in Artificial Intelligence, pages 181–200. Springer-Verlag, Sept. 1999.

[20] C. Lutz. Reasoning with concrete domains. In T. Dean, editor, *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence IJCAI-99*, pages 90–95, Stockholm, Sweden, July 31 – August 6, 1999. Morgan-Kaufmann Publishers.

[21] B. Nebel. Terminological reasoning is inherently intractable. *Artificial Intelligence*, 43:235–249, 1990.

[22] B. Nebel. Terminological cycles: Semantics and computational properties. In J. F. Sowa, editor, *Principles of Semantic Networks: Explorations in the Representation of Knowledge*, pages 331–361. Morgan Kaufmann Publishers, San Mateo (CA), USA, 1991.

[23] E. M. Post. A variant of a recursively unsolvable problem. *Bull. Am. Math. Soc.*, 52:264–268, 1946.

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

[25] U. Sattler. *Terminological knowledge representation systems in a process engineering application.* PhD thesis, LuFG Theoretical Computer Science, RWTH-Aachen, 1998.

[26] K. Schild. Terminological cycles and the propositional μ-calculus. In P. T. Jon Doyle, Erik Sandewall, editor, *Proceedings of the 4th International Conference on Principles of Knowledge Representation and Reasoning*, pages 509–520, Bonn, FRG, May 1994. Morgan Kaufmann.