LTCS–Report

Keys, Nominals, and Concrete Domains

Carsten Lutz, Carlos Areces, Ian Horrocks, Ulrike Sattler

LTCS-Report 02-04

Keys, Nominals, and Concrete Domains

Carsten Lutz, Carlos Areces, Ian Horrocks, Ulrike Sattler

Abstract

Many description logics (DLs) combine knowledge representation on an abstract, logical level with an interface to "concrete" domains such as numbers and strings with built-in predicates such as $<$, $+$, and prefix-of. These hybrid DLs have turned out to be quite useful for reasoning about conceptual models of information systems, and as the basis for expressive ontology languages. We propose to further extend such DLs with *key constraints* that allow the expression of statements like "US citizens are uniquely identified by their social security number". Based on this idea, we introduce a number of natural description logics and perform a detailed analysis of their decidability and computational complexity. It turns out that naive extensions with key constraints easily lead to undecidability, whereas more careful extensions yield NExpTime-complete DLs for a variety of useful concrete domains.

1 Motivation

Description logics (DLs) are a family of formalisms that allow the representation of and reasoning about conceptual knowledge in a structured and semantically well-understood manner [8, 2]. The central entities for representing such knowledge are *concepts*, which are constructed from atomic *concept names* (unary predicates) and *role names* (binary relations) by means of the concept and role constructors offered by a particular DL. For example, in the basic propositionally closed description logic $\mathcal{ALC}$, we can describe a company that has part-time employees but only full-time managers using the concept

$$\mathsf{Company} \sqcap \exists \mathsf{employee}.\mathsf{Parttime} \sqcap \forall \mathsf{employee}.(\neg \mathsf{Manager} \sqcup \neg \mathsf{Parttime}).$$

In this example, all uppercase words denote concept names while the lowercase employee denotes a role name.

Rather than being viewed only as conceptual entities in a knowledge base, concepts can, more generally, be understood as the central notion in various kinds of class-centered formalisms. In the last decade, this observation has given rise to various new and exciting applications of description logics such as reasoning about database conceptual models expressed in entity-relationship diagrams or object-oriented schemas and reasoning about ontologies for use in the semantic web, see [18, 16] and [6, 29, 30], respectively. These new applications have, in turn, stimulated research in description logics since the expressive power of existing DLs was insufficient for the new tasks. One important extension of "classical" description logics concerns so-called concrete domains: assume, e.g., that we want to continue our example from above by equipping companies with a founding year and employees with a hiring year. Then, we may want to describe companies that were founded before 1970 and state that the hiring year of employees is not prior to the founding year of the employing company. To do this, we obviously need a way to talk about natural numbers (such as 1970) and comparisons between natural numbers.

Nowadays, the standard approach to integrate numbers and other datatypes into description logics is to extend DLs with so-called *concrete domains* as first proposed by Baader and Hanschke in [3], see also the survey [38]. More precisely, a concrete domain $\mathcal{D}$ consists of a set (such as the natural numbers) and predicates which are associated with a *fixed* extension over this set[1] (such as the unary $=_0$, the binary $<$, and the ternary $+$). The integration of concrete domains into, say, the description logic $\mathcal{ALC}$ is achieved by adding

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

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

$$\mathsf{Employee} \sqcap \exists \mathsf{employer}.(\exists \mathsf{foundingyear}.{<_{1970}}) \sqcap \exists \mathsf{hiringyear}, \mathsf{employer}\ \mathsf{foundingyear}.{\geq}$$

describes an employee who is employed by a company founded before 1970 and whose hiring year is not prior to the company's founding year. Here, the term inside parenthesis and the third conjunct are instances of the concrete domain concept constructor (not to be confused with the existential value restriction as in $\exists\mathsf{employee}.\mathsf{Parttime}$), $\mathsf{employer}$ is an abstract feature, and $\mathsf{foundingyear}$ and $\mathsf{hiringyear}$ are concrete features.

Concrete domains can be considered rather important in the "modern" applications of DLs mentioned above:

– the standard way of using description logics for reasoning about conceptual database models is to translate a given model into a DL representation and then use a description logic reasoning procedure for detecting inconsistencies and inferring consequences of the information provided explicitly in the model such as additional, implicit containments between entities/classes [18]. Since most databases store "concrete" data like numbers and strings, constraints concerning such data are usually part of the conceptual model and should thus also be captured by the description logic used for reasoning. Indeed, the above example concepts can be viewed as the DL encoding of constraints from a database about companies and their employees. As discussed in [41], description logics with concrete domains are well-suited for conceptual modelling applications involving concrete datatypes.

[1] This fixed extension is why these predicates are often called "built-in".

– in the construction of ontologies for the semantic web, so-called *concrete datatypes* play a prominent role [30]. Say, for example, that we want to construct an ontology which can be used for describing car dealers' web pages and web services. In such an ontology, concrete datatypes such as prices, manufacturing years, and names of car models will doubtlessly be very important. To formulate this ontology using a DL, we thus need a way to represent these concrete datatypes. Consequently, almost all DLs that have been proposed as an ontology language for the semantic web are equipped with some form of concrete domain [20, 30, 19].

In this paper, we propose to further extend the expressive power of description logics with concrete domains in a way that is useful both for knowledge representation and the two applications sketched above. Let us describe the basic idea, which is to use concrete features for defining "key constraints", using three examples:

1. Suppose that, in a knowledge representation application, we represent nationalities by concept names such as US and German and, for US citizens, we store the social security number using a concrete feature ssn. Then it would be natural to state that US citizens are uniquely identified by their social security number, i.e. any two distinct instances of

$$\mathsf{Human} \sqcap \exists \mathsf{nationality}.\mathsf{US}$$

must have different values for the ssn feature. In our extension of DLs with concrete domains, this can be expressed by using the *key definition*

$$(\mathsf{ssn} \text{ keyfor } \mathsf{Human} \sqcap \exists \mathsf{nationality}.\mathsf{US}).$$

2. Returning to our database about companies and employees, it could be useful to equip every employee with (i) a concrete feature branch storing the branch-ID in which she is working and (ii) a concrete feature id storing her personnel-ID. It would then be natural to enforce that even though personnel-IDs are not unique, the branch-ID together with the personnel-ID uniquely identifies employees. We can do this by using the *n-ary* key definition

$$(\mathsf{branch}, \mathsf{id} \text{ keyfor } \mathsf{Employee}).$$

3. In the car dealers' ontology, we may assume that cars as well as manufacturers are equipped with identification numbers and that every car is uniquely identified by the combination of its own identification number and its manufacturers one. To express this, we could employ an n-ary key definition referring to *sequences* of features:

$$(\mathsf{id}, \mathsf{manufacturer}\ \mathsf{id} \text{ keyfor } \mathsf{Car}).$$

More formally, we propose to extend DLs to provide for concrete domains with *key boxes*, which are sets of key definitions of the form

$$(u_1, \ldots, u_n \text{ keyfor } C),$$

where the u_i are sequences $f_1 \cdots f_n g$ of n abstract features $f_1, \ldots, f_n$ followed by a single concrete feature g, and C is a concept. As the above examples illustrate, the idea of key constraints is a very natural one. Since, moreover, keys play an important role in databases and, as mentioned above, reasoning about database conceptual models is an important, challenging application of description logics, several approaches to extend description logics with keys have already been investigated [15, 17, 35]. What distinguishes our approach from existing ones, however, is the idea to use concrete domains for constructing key constraints, rather than defining keys on an abstract, logical level.

The goal of this paper is *to provide a comprehensive analysis of the effects on decidability and computational complexity of adding key boxes to description logics with concrete domains.* To this end, we extend the two description logics with concrete domains $\mathcal{ALC}(\mathcal{D})$ and $\mathcal{SHOQ}(\mathcal{D})$ with key boxes, in this way obtaining $\mathcal{ALCK}(\mathcal{D})$ and $\mathcal{SHOQK}(\mathcal{D})$, respectively. While $\mathcal{ALC}(\mathcal{D})$ can be viewed as the basic DL with concrete domains and has already been discussed above, $\mathcal{SHOQ}(\mathcal{D})$ was proposed as an ontology language in [31]. It provides a wealth of expressive possibilities such as general concept inclusion axioms (GCIs), transitive roles, role hierarchies, nominals, and qualifying number restrictions. Moreover, it offers a restricted variant of the concrete domain constructor that disallows the use of *sequences* of features in order to avoid undecidability of reasoning.

The main outcome of our investigations is that key constraints can have a dramatic impact on the decidability and complexity of reasoning: for example, whereas satisfiability of $\mathcal{ALC}(\mathcal{D})$-concepts is known to be PSpace-complete [40], we are able to show that satisfiability of $\mathcal{ALCK}(\mathcal{D})$-concepts w.r.t. key boxes is, in general, undecidable. Decidability can be recovered if we restrict the concepts used in key boxes to Boolean combinations of concept names, thus obtaining *Boolean key boxes.* Interestingly, satisfiability of $\mathcal{ALCK}(\mathcal{D})$-concepts w.r.t. Boolean key boxes is still NExpTime-complete even for very simple concrete domains. In the case of $\mathcal{SHOQ}(\mathcal{D})$ and $\mathcal{SHOQK}(\mathcal{D})$, the leap in complexity is somewhat less dramatic since $\mathcal{SHOQ}(\mathcal{D})$-concept satisfiability is already ExpTime-complete: again, the addition of key boxes results in NExpTime-complete reasoning problems (more details are given below).

It is interesting to note that there exists a close connection between key definitions and so-called *nominals*, i.e. concept names that can have *at most one* instance, such as Pope. Nominals are a standard means of expressivity in description logics and sometimes appear in disguise as the "one-of" operator [14, 30]. It is not hard to see that key boxes can "simulate" nominals: if, for example, we use a concrete domain based on the natural numbers and providing unary predicates $=_n$ for equality with $n \in \mathbb{N}$, then the key definition (g keyfor $\top$), where $\top$ stands for logical truth, obviously makes the concept $\exists g.{=_3}$ behave like a nominal. For this reason, we also consider $\mathcal{ALCO}(\mathcal{D})$, the extension of $\mathcal{ALC}(\mathcal{D})$ with nominals, and $\mathcal{ALCOK}(\mathcal{D})$, the extension of $\mathcal{ALCK}(\mathcal{D})$ with nominals.[2] Our main result concerning nominals is that, although in general being of lower expressive power than key boxes, nominals already lead to NExpTime-hardness of reasoning if combined with concrete domains: there exist concrete domains $\mathcal{D}$ such that

[2]Note that the logic $\mathcal{SHOQ}(\mathcal{D})$ already provides for nominals.

$\mathcal{ALCO}(\mathcal{D})$-concept satisfiability is NExpTime-complete. We should like to stress that all NExpTime-hardness results obtained in this paper are in accordance with the observation made in [39], namely that the PSpace-upper bound for reasoning with $\mathcal{ALC}(\mathcal{D})$ is not robust w.r.t. extensions of the logic: there exist several "seemingly harmless" extensions of $\mathcal{ALC}(\mathcal{D})$ (for example acyclic TBoxes and inverse roles) which make the complexity of reasoning leap from PSpace-completeness to NExpTime-completeness for many natural concrete domains.

The remainder of this paper is organized as follows:

In Section 2, we formally introduce concrete domains, key boxes, and the description logic $\mathcal{ALCOK}(\mathcal{D})$ together with its fragments $\mathcal{ALCK}(\mathcal{D})$ and $\mathcal{ALCO}(\mathcal{D})$. We also define Boolean key boxes and so-called *path-free* key boxes which prohibit the use of sequences of features inside key definitions. We also introduce *unary key boxes* and *n-ary key boxes* in analogy to the n-ary key definitions used in the examples above.

Section 3 is devoted to establishing lower bounds for description logics with concrete domains, key boxes, and nominals. In Section 3.1, we use a reduction of the Post Correspondence Problem to prove that $\mathcal{ALCK}(\mathcal{D})$-concept satisfiability w.r.t. (non-Boolean) key boxes is undecidable if the concrete domain $\mathcal{D}$ provides for the natural numbers, a unary predicate for equality with zero, binary equality and inequality, and a binary incrementation predicate. We then shift our attention towards Boolean key boxes since, in Section 4, we show that this restriction recovers decidability. In Section 3.2, we introduce a NExpTime-complete variant of the domino problem and three concrete domains that are useful for the reduction of this problem to concept satisfiability in DLs providing for Boolean key boxes or nominals. In Section 3.3, we use these concrete domains to prove that $\mathcal{ALCK}(\mathcal{D})$-concept satisfiability w.r.t. Boolean and path-free key boxes is NExpTime-hard if $\mathcal{D}$ provides two unary predicates denoting disjoint singleton sets. We then strengthen this result to unary key boxes, but, to compensate for the weaker key box formalism, we use more expressive concrete domains. For example, it suffices that the concrete domain $\mathcal{D}$ provides for the natural numbers, a unary predicate $=_n$ for each $n \in \mathbb{N}$, and ternary addition. In Section 3.4, we prove that $\mathcal{ALCO}(\mathcal{D})$-concept satisfiability without reference to key boxes is already NExpTime-hard. For this result, the strongest requirements on the concrete domain are adopted: we additionally need predicates such as multiplication and exponentiation. However, we are able to show that there still exist concrete domains that are computationally very simple (PTime) if considered in isolation, but lead to NExpTime-hardness if used with the DL $\mathcal{ALCO}(\mathcal{D})$.

The purpose of Section 4 is to develop reasoning procedures for description logics with key boxes and to prove upper complexity bounds matching the NExpTime lower bounds established in the previous section. We start in Section 4.1 with a tableau algorithm that is capable of deciding $\mathcal{ALCOK}(\mathcal{D})$-concept satisfiability w.r.t. Boolean key boxes if the concrete domain $\mathcal{D}$ is *key-admissible.* Intuitively, a concrete domain $\mathcal{D}$ is key admissible if there exists an algorithm that takes a finite conjunction c of predicates from $\mathcal{D}$ over some set of variables, decides whether this conjunction is satisfiable, and additionally returns information on which variables must take the same values in solutions of c. We have chosen a tableau algorithm since this type of reasoning procedure has the potential to be implemented in efficient reasoners and has been shown to behave well in practice [33, 23]. The algorithm provides us with the following upper bound: $\mathcal{ALCOK}(\mathcal{D})$-concept satisfiability w.r.t. Boolean key boxes is in NExpTime if $\mathcal{D}$ is key-admissible and the algorithm mentioned in the explanation of "key-admissible" runs in non-deterministic polynomial time. In Section 4.2, we devise a tableau algorithm for $\mathcal{SHOQK}(\mathcal{D})$-concept satisfiability w.r.t. path-free key boxes which might involve *non-Boolean* concepts. The restriction to Boolean concepts in key boxes was necessary for $\mathcal{ALCOK}(\mathcal{D})$ in order to avoid undecidability. For $\mathcal{SHOQK}(\mathcal{D})$, this restriction is not necessary since $\mathcal{SHOQK}(\mathcal{D})$'s concrete domain constructor is weaker than the one provided by $\mathcal{ALCOK}(\mathcal{D})$: it does not admit the use of *sequences* of features as arguments. As a by-product of the correctness proof of the algorithm, we obtain a bounded model property for $\mathcal{SHOQK}(\mathcal{D})$, which implies that $\mathcal{SHOQK}(\mathcal{D})$-concept satisfiability w.r.t. path-free key boxes is in NExpTime if $\mathcal{D}$ is key-admissible and the corresponding algorithm runs in non-deterministic polynomial time.

In Section 5, we summarize the results obtained and give an outlook to possible future research.

2 Description Logics with Concrete Domains

In the following, we introduce the description logic $\mathcal{ALCOK}(\mathcal{D})$. Let us start with defining concrete domains:

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 and $\Phi_\mathcal{D}$ a set of predicate names. Each predicate name $P \in \Phi_\mathcal{D}$ is associated with an arity n and an n-ary predicate $P^\mathcal{D} \subseteq \Delta^n_\mathcal{D}$. $\Diamond$

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

Definition 2 ($\mathcal{ALCOK}(\mathcal{D})$ Syntax). Let $\mathsf{N_C}$, $\mathsf{N_O}$, $\mathsf{N_R}$, $\mathsf{N_{cF}}$ be pairwise disjoint and countably infinite sets of *concept names*, *nominals*, *role names*, and *concrete features*. Furthermore, we assume that $\mathsf{N_R}$ contains a countably infinite subset $\mathsf{N_{aF}}$ of *abstract features*. A *path* u is a composition $f_1 \cdots f_n g$ of n abstract features $f_1, \dots, f_n$ $(n \geq 0)$ and a concrete feature g. Let $\mathcal{D}$ be a concrete domain. The set of $\mathcal{ALCOK}(\mathcal{D})$-concepts is the smallest set such that

- every concept name and every nominal is a concept, and
- if C and D are concepts, R is a role name, g is a concrete feature, $u_1, \dots, u_n$ are paths, and $P \in \Phi_\mathcal{D}$ is a predicate of arity n, then the following expressions are also concepts:

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

A *key definition* is an expression

$$(u_1, \dots, u_k \text{ keyfor } C),$$

where $u_1, \dots, u_k$ $(k \geq 1)$ are paths and C is a concept. A finite set of key definitions is called *key box.* $\Diamond$

As usual, we use $\top$ as abbreviation for an arbitrary propositional tautology, $\bot$ as abbreviation for $\neg\top$, $C \rightarrow D$ as abbreviation for $\neg C \sqcup D$, and $C \leftrightarrow D$ as abbreviation for $(C \rightarrow D) \sqcap (D \rightarrow C)$. Throughout this paper, we will also consider several fragments of the description logic $\mathcal{ALCOK}(\mathcal{D})$. The DL $\mathcal{ALCO}(\mathcal{D})$ is obtained from $\mathcal{ALCOK}(\mathcal{D})$ by admitting only empty key boxes. In particular, the set of $\mathcal{ALCO}(\mathcal{D})$-concepts is just the set of $\mathcal{ALCOK}(\mathcal{D})$-concepts. Furthermore, by disallowing the use of nominals, we obtain the fragment $\mathcal{ALC}(\mathcal{D})$ of $\mathcal{ALCO}(\mathcal{D})$ and $\mathcal{ALCK}(\mathcal{D})$ of $\mathcal{ALCOK}(\mathcal{D})$.

The description logic $\mathcal{ALCOK}(\mathcal{D})$ is equipped with a Tarski-style set-theoretic semantics. Along with the semantics, we introduce the two standard inference problems: concept satisfiability and concept subsumption.

Definition 3 ($\mathcal{ALCOK}(\mathcal{D})$ Semantics). An *interpretation* $\mathcal{I}$ is a pair $(\Delta_\mathcal{I}, \cdot^\mathcal{I})$, where $\Delta_\mathcal{I}$ is a non-empty set, called the *domain*, and $\cdot^\mathcal{I}$ is the *interpretation function*. The interpretation function maps

- each concept name C to a subset $C^\mathcal{I}$ of $\Delta_\mathcal{I}$,
- each nominal N to a singleton subset $N^\mathcal{I}$ 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 $u^\mathcal{I}(d)$ is defined as $g^\mathcal{I}(f_n^\mathcal{I} \cdots (f_1^\mathcal{I}(d)) \cdots)$. The interpretation function is extended to arbitrary concepts as follows:

$$\begin{aligned}
(\neg C)^\mathcal{I} &:= \Delta_\mathcal{I} \setminus C^\mathcal{I} \\
(C \sqcap D)^\mathcal{I} &:= C^\mathcal{I} \cap D^\mathcal{I} \\
(C \sqcup D)^\mathcal{I} &:= C^\mathcal{I} \cup D^\mathcal{I} \\
(\exists R.C)^\mathcal{I} &:= \{d \in \Delta_\mathcal{I} \mid \text{There is } e \in \Delta_\mathcal{I} \text{ with } (d,e) \in R^\mathcal{I} \text{ and } e \in C^\mathcal{I}\} \\
(\forall R.C)^\mathcal{I} &:= \{d \in \Delta_\mathcal{I} \mid \text{For all } e \in \Delta_\mathcal{I}, \text{ if } (d,e) \in R^\mathcal{I}, \text{ then } e \in C^\mathcal{I}\} \\
(\exists u_1, \ldots, u_n.P)^\mathcal{I} &:= \{d \in \Delta_\mathcal{I} \mid \exists x_1, \ldots, x_n \in \Delta_\mathcal{D} : u_i^\mathcal{I}(d) = x_i \text{ and } (x_1, \ldots, x_n) \in P^\mathcal{D}\} \\
(g\uparrow)^\mathcal{I} &:= \{d \in \Delta_\mathcal{I} \mid g^\mathcal{I}(d) \text{ undefined}\}.
\end{aligned}$$

Let $\mathcal{I}$ be an interpretation. Then $\mathcal{I}$ is a *model* of a concept C iff $C^\mathcal{I} \neq \emptyset$. Moreover, $\mathcal{I}$ *satisfies* a key definition $(u_1, \ldots, u_n \text{ keyfor } C)$ if, for any $a, b \in C^\mathcal{I}$,

$$\text{if, for } 1 \leq i \leq n,\ u_i^\mathcal{I}(a) = u_i^\mathcal{I}(b), \text{ then } a = b.$$

$\mathcal{I}$ is a *model* of a key box $\mathcal{K}$ iff $\mathcal{I}$ satisfies all key definitions in $\mathcal{K}$. A concept C is *satisfiable w.r.t. a key box* $\mathcal{K}$ iff C and $\mathcal{K}$ have a common model. C is *subsumed by* a concept D *w.r.t. a key box* $\mathcal{K}$ (written $C \sqsubseteq_\mathcal{K} D$) iff $C^\mathcal{I} \subseteq D^\mathcal{I}$ for all models $\mathcal{I}$ of $\mathcal{K}$. $\diamondsuit$

It is well-known that, in description logics providing for negation, subsumption can be reduced to (un)satisfiability and vice versa: $C \sqsubseteq_{\mathcal{K}} D$ iff $C \sqcap \neg D$ is unsatisfiable w.r.t. $\mathcal{K}$ and C is satisfiable w.r.t. $\mathcal{K}$ iff $C \not\sqsubseteq_{\mathcal{K}} \bot$. This allows us to concentrate on concept satisfiability when devising complexity bounds for reasoning with description logics: lower and upper complexity bounds for concept satisfiability imply corresponding bounds for concept subsumption—only for the complementary complexity class.

If decision procedures for description logics with concrete domains are to be devised without committing to a *particular* concrete domain, then a well-defined interface between the decision procedure and a concrete domain reasoner is needed. Usually, the concrete domain is required to be *admissible* [3, 37, 38]:

Definition 4 ($\mathcal{D}$-conjunction, admissibility). Let $\mathcal{D}$ be a concrete domain and V a set of variables. A *$\mathcal{D}$-conjunction* is a (finite) predicate conjunction of the form

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

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

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

We refer to the satisfiability of $\mathcal{D}$-conjunctions as *$\mathcal{D}$-satisfiability.* $\Diamond$

As we shall see, it sometimes makes a considerable difference w.r.t. complexity and decidability to restrict key boxes in various ways, for example to admit only the concept $\top$ on the right-hand side of key definitions or to disallow paths of length greater than one. Therefore, we introduce some useful notions.

Definition 5 (Boolean, Path-free, Simple). A key box $\mathcal{K}$ is called

- *Boolean* if all concepts appearing in (key definitions in) $\mathcal{K}$ are Boolean combinations of concept names;
- *path-free* if all key definitions in $\mathcal{K}$ are of the form ($g_1, \ldots, g_n$ keyfor C), where $g_1, \ldots, g_n \in \mathsf{N_{cF}}$;
- *simple* if it is both path-free and Boolean;
- a *unary key box* if all key definitions in $\mathcal{K}$ are *unary key definitions*, i.e. of the form (u keyfor C).

A concept C is called *path-free* if, in all its subconcepts of the form $\exists u_1, \ldots, u_n.P$, $u_1, \ldots, u_n$ are concrete features. ◊

To emphasize that a key box must *not* necessarily be Boolean or path-free, we sometimes call such a key box *general*. Similarly, to emphasize that a key box is not necessarily a unary key box, we sometimes call such a key box *n-ary key box*.

3 Lower Bounds

In this section, we prove lower bounds for description logics with concrete domains which provide for key boxes and/or nominals. In Section 3.1, we start with showing that the satisfiability of $\mathcal{ALCK}(\mathcal{D})$-concepts w.r.t. (general) key boxes is undecidable for many interesting concrete domains. This discouraging result is relativized by the fact that, in Section 4.1, we shall prove that the restriction to Boolean key boxes recovers decidability. It is thus interesting to look for lower complexity bounds that apply under this restriction. In preparation for this, we introduce in Section 3.2 a NExpTime-complete variant of the domino problem and three concrete domains that are well-suited for reductions of this problem. In Section 3.3, we then prove that satisfiability of path-free $\mathcal{ALCK}(\mathcal{D})$-concepts w.r.t. simple key boxes is NExpTime-hard for a large class of concrete domains $\mathcal{D}$ and that, for many concrete domains, this does hold even if we restrict ourselves to unary key boxes. Finally, we consider the description logic $\mathcal{ALCO}(\mathcal{D})$ in Section 3.4 and identify several concrete domains such that $\mathcal{ALCO}(\mathcal{D})$-concept satisfiability (without key boxes!) is NExpTime-hard. As we will explain, key boxes and nominals are closely related: key boxes can express nominals, but are more powerful. Intuitively, key boxes can be used to define concepts that behaves like nominals—but the number of "nominals" defined in this way cannot be bounded in advance by a simple syntactic test.

3.1 Undecidability of $\mathcal{ALCK}(\mathcal{D})$ with General Key Boxes

We prove that satisfiability of $\mathcal{ALCK}(\mathcal{D})$-concepts w.r.t. key boxes is undecidable for a large class of concrete domains—if we allow complex $\mathcal{ALCK}(\mathcal{D})$-concepts to occur in key definitions. The proof is by a reduction of the well-known undecidable Post Correspondence Problem [43, 27].

Definition 6 (PCP). An *instance* P of the *Post Correspondence Problem* is given by a finite, non-empty list $(\ell_1, r_1), \ldots, (\ell_k, r_k)$ of pairs of words over some alphabet Σ. A sequence of integers $i_1, \ldots, i_m$, with $m \geq 1$, is called a *solution* for P iff

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

The *Post Correspondence Problem (PCP)* is to decide, for a given instance P, whether P has a solution. ◊

For reducing the PCP, we need an appropriate concrete domain. It is obviously natural to use a concrete domain based on words and concatenation. We will later see that

$$\begin{aligned}
\mathsf{Step} &:= \bigsqcap_{1\leq i\leq k} \exists f_i.(\neg A \sqcap \exists g.{=_\epsilon} \sqcap \exists \ell, r.{\neq}) \\
&\quad \sqcap \bigsqcap_{1\leq i\leq k} (\exists \ell, f_i\ell.\mathsf{conc}_{\ell_i} \sqcap \exists r, f_i r.\mathsf{conc}_{r_i}) \\
C_P &:= \exists \ell.{=_\epsilon} \sqcap \exists r.{=_\epsilon} \\
&\quad \sqcap\ \exists R.(A \sqcap \exists g.{=_\epsilon} \sqcap \neg\mathsf{Step}) \\
&\quad \sqcap \mathsf{Step} \\
\mathcal{K}_P &:= \{g \text{ keyfor } \neg\mathsf{Step}\}
\end{aligned}$$

Figure 1: The $\mathcal{ALCK}(\mathsf{W})$ reduction concept C_P and key box $\mathcal{K}_P$.

the results obtained for this concrete domain carry over to other, more natural concrete domains based on numbers and arithmetics. The following concrete domain was introduced in [39].

Definition 7 (Concrete domain W). Let Σ be an alphabet. The concrete domain W is defined by setting $\Delta_\mathsf{W} := \Sigma^*$ and defining Φ_W as the smallest set containing the following predicates:

- unary predicates word and nword with $\mathsf{word}^\mathsf{W} = \Delta_\mathsf{W}$ and $\mathsf{nword}^\mathsf{W} = \emptyset$,
- unary predicates $=_\epsilon$ and $\neq_\epsilon$ with $=_\epsilon^\mathsf{W} = \{\epsilon\}$ and $\neq_\epsilon^\mathsf{W} = \Sigma^+$,
- a binary equality predicate $=$ and a binary inequality predicate $\neq$ with the obvious interpretation, and
- for each $w \in \Sigma^+$, two binary predicates conc_w and nconc_w with

$$\mathsf{conc}_w^\mathsf{W} = \{(u,v) \mid v = uw\} \text{ and } \mathsf{nconc}_w^\mathsf{W} = \{(u,v) \mid v \neq uw\}.$$

$\diamondsuit$

It is readily checked that W satisfies Properties 1 and 2 of admissibility. In [39], the complexity of W-satisfiability is investigated.

Theorem 8. W-*satisfiability is in* PTIME.

Thus, W is admissible. This is important since our aim is to demonstrate that the undecidability of $\mathcal{ALCK}(\mathsf{W})$-concept satisfiability is due to the presence of keys, and not due to the undecidability of W-satisfiability.

We can now discuss the reduction of the PCP. A given instance $(\ell_1, r_1), \ldots, (\ell_k, r_k)$ is translated into an $\mathcal{ALCK}(\mathcal{D})$-concept and key box as shown in Figure 1. In this figure, $f_1, \ldots, f_k$ denote abstract features while g, ℓ, and r denote concrete features. The definition of the concept Step just serves as an abbreviation. The idea behind the reduction is that a common model of C_P and $\mathcal{K}_P$ encodes *all* potential solutions

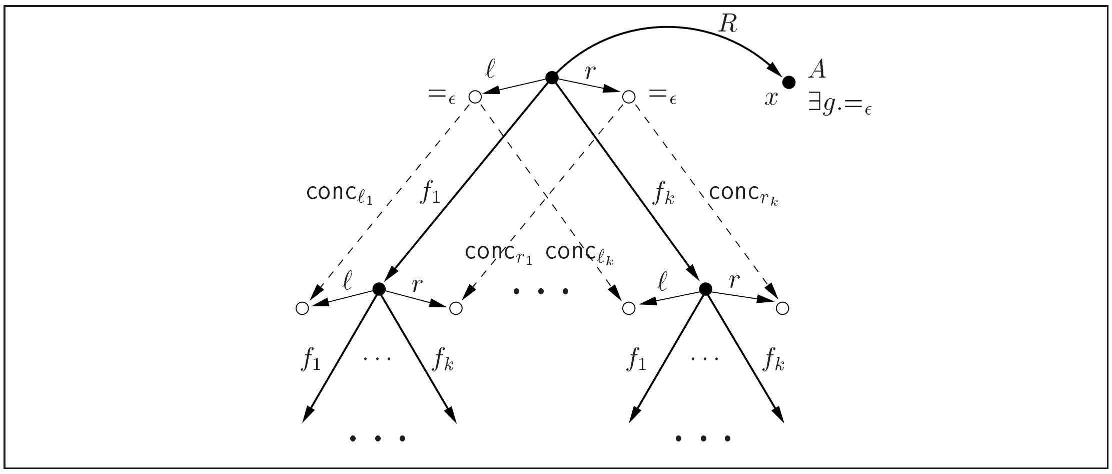

Figure 2: An example model of C_P and $\mathcal{K}_P$.

(i.e., sequences that can be completed to a solution) for the PCP P and, moreover, the existence of such a model guarantees that no potential solution is indeed a solution. Models of C_P and $\mathcal{K}_P$, such as the one displayed in Figure 2, have the form of an infinite k-ary tree whose root is connected to an "extra node" x via the role R. Intuitively, each node of the tree represents one sequence of indices $i_1, \ldots, i_n$, its ℓ-successor represents the corresponding left concatenation $\ell_{i_1} \cdots \ell_{i_n}$, and its r-successor the corresponding right concatenation $r_{i_1} \cdots r_{i_n}$. To enforce the existence of the infinite tree, we employ the key box $\mathcal{K}_P$: consider for example the root node's f_1-successor in Figure 2—let us call this node y. Due to Line 3 of C_P and Line 1 of Step, we have $y \in (\exists g.=_\epsilon)^\mathcal{I}$. Due to Line 2 of C_P, we also have $x \in (\exists g.=_\epsilon)^\mathcal{I}$ and $x \in (\neg\mathsf{Step})^\mathcal{I}$, where x is the "extra node" mentioned above. In view of the key box $\mathcal{K}_P$, this implies that either (i) $x = y$ or (ii) $y \in \mathsf{Step}^\mathcal{I}$. It is easy to see that (i) is impossible since Line 2 of C_P and Line 1 of Step imply that $x \in A^\mathcal{I}$ and $y \in (\neg A)^\mathcal{I}$. Hence $y \in \mathsf{Step}^\mathcal{I}$ and, by Line 2 of Step, y has the appropriate f_i-successors for $1 \leq i \leq n$. In the same way, the construction of the tree can be continued ad infinitum. The second line in the definition of Step enforces that $\ell^\mathcal{I}(z) = \ell_{i_1} \cdots \ell_{i_n}$ and $r^\mathcal{I}(z) = r_{i_1} \cdots r_{i_n}$ for z an $f_{i_1} \cdots f_{i_n}$-successor of the root node. Finally, the concept $\exists \ell, r.\neq$ in Line 1 of Step implies that $\ell^\mathcal{I}(z) \neq r^\mathcal{I}(z)$ holds at all nodes z of the tree (except for the root), which implies that no potential solution is a solution.

Since the size of C_P and $\mathcal{K}_P$ is clearly polynomial in k and the key box $\mathcal{K}_P$ is a unary key box, we obtain the following proposition.

Proposition 9. *The satisfiability of $\mathcal{ALCK}(\mathsf{W})$-concepts w.r.t. (general) unary key boxes is undecidable.*

To emphasize that this undecidability result was obtained using a very simple concrete domain, let us combine Theorem 8 with Proposition 9.

Theorem 10. *There exists a concrete domain $\mathcal{D}$ such that $\mathcal{D}$-satisfiability is in* PTIME *and satisfiability of $\mathcal{ALCK}(\mathcal{D})$-concepts w.r.t. (general) unary key boxes is undecidable.*

On first sight, the concrete domain W might look artificial and one may question the relevance of lower bounds that have been obtained using W. However, it is straightforward to encode words as natural numbers and to define concatenation of words as rather simple operations on the natural numbers [5]: words $w \neq \epsilon$ over the alphabet Σ of cardinality $\#\Sigma$ can be interpreted as numbers written at base $\#\Sigma + 1$ where the symbol that is the "0 digit" does never occur. Hence, we can use the corresponding natural number (at base 10) to represent a word w and the number 0 to represent the empty word. The concatenation of two words v and w can then be expressed as $vw = v \cdot (\#\Sigma + 1)^{|w|} + w$, where $|w|$ denotes the length of the word w. Moreover, exponentiation can be expressed as multiple multiplications, multiplication as multiple additions, and addition as multiple incrementation (see [39] for details). This observation gives rise to the following theorem:

Theorem 11. *Let $\mathcal{D}$ be a concrete domain such that $\mathbb{N} \subseteq \Delta_{\mathcal{D}}$, $\Phi_{\mathcal{D}}$ contains a unary predicate $=_0$ with $(=_0)^{\mathcal{D}} = \{0\}$, binary equality and inequality predicates, and a binary predicate incr with $\mathsf{incr}^{\mathcal{D}} \cap \{(n,x) \mid n \in \mathbb{N}$ and $x \in \Delta_{\mathcal{D}}\} = \{(k, k+1) \mid k \in \mathbb{N}\}$. Then satisfiability of $\mathcal{ALCK}(\mathcal{D})$-concepts w.r.t. (general) unary key boxes is undecidable.*

Since subsumption can be reduced to satisfiability as described in Section 2, we obtain corresponding undecidability results for concept subsumption.

3.2 Domino Problems and Concrete Domains

In this section, we introduce a NExpTime-complete variant of the well-known, undecidable domino problem [11, 36], and then define three concrete domains D_1, D_2, and D_3. These concrete domains will be used in Sections 3.3 and 3.4 to establish lower bounds for reasoning with $\mathcal{ALCK}(\mathcal{D})$ and Boolean key boxes, and for reasoning with $\mathcal{ALCO}(\mathcal{D})$.

In general, a domino problem is given by a finite set of *tile types*. All tile types are of the same size, each type having a square shape and colored edges. An unlimited number of tiles of each type is available. In the NExpTime-hard variant of the domino problem that we use, the task is to tile a $2^{n+1} \times 2^{n+1}$-torus (i.e., a $2^{n+1} \times 2^{n+1}$-rectangle whose borders are "glued" together) where neighboring edges have the same color.

Definition 12 (Domino System). A *domino system* $\mathfrak{D}$ is a triple (T, H, V), where $T \subset \mathbb{N}$ is a finite set of *tile types* and $H, V \subseteq T \times T$ represent the horizontal and vertical matching conditions. Let $\mathfrak{D}$ be a domino system and $a = a_0, \ldots, a_{n-1}$ an *initial condition*, i.e. an n-tuple of tiles. A mapping $\tau : \{0, \ldots, 2^{n+1}\} \times \{0, \ldots, 2^{n+1}\} \to T$ is a *solution* for $\mathfrak{D}$ and a iff, for all $x, y < 2^{n+1}$, the following holds:

– if $\tau(x, y) = t$ and $\tau(x \oplus_{2^{n+1}} 1, y) = t'$, then $(t, t') \in H$

– if $\tau(x, y) = t$ and $\tau(x, y \oplus_{2^{n+1}} 1) = t'$, then $(t, t') \in V$

– $\tau(i, 0) = a_i$ for $i < n$.

where $\oplus_i$ denotes addition modulo i. $\Diamond$

As shown in, e.g., Corollary 4.15 of [37], it follows from results in [13] that the above variant of the domino problem is NExpTime-complete.

We now define the concrete domain D_1 which will be used in the reduction of the NExpTime-complete domino problem to $\mathcal{ALCK}(\mathsf{D}_1)$-concept satisfiability w.r.t. Boolean key boxes.

Definition 13 (Concrete Domain D_1). The concrete domain D_1 is defined by setting $\Delta_{\mathsf{D}_1} := \{0, 1\}$ and Φ_{D_1} to the (smallest) set containing the following predicates:

– a unary predicate $\top_{\mathsf{D}_1}$ with $(\top_{\mathsf{D}_1})^{\mathsf{D}_1} = \Delta_{\mathsf{D}_1}$ and a unary predicate $\bot_{\mathsf{D}_1}$ with $(\bot_{\mathsf{D}_1})^{\mathsf{D}_1} = \emptyset$;

– unary predicates $=_0$ and $=_1$ with $(=_i)^{\mathsf{D}_1} = \{i\}$.

$\Diamond$

It is readily checked that the concrete domain D_1 is admissible and that D_1-satisfiability is in PTime.

Proposition 14. D_1-*satisfiability is in* PTime.

The second concrete domain D_2 will be used for a reduction of the NExpTime-complete domino problem to $\mathcal{ALCK}(\mathsf{D}_2)$-concept satisfiability w.r.t. Boolean *unary* key boxes. For this reduction we need to "store" vectors of bits in single concrete domain elements.

Definition 15 (Concrete Domain D_2). For every $n \in \mathbb{N}$, a function $v : \{0, \dots, n-1\} \to \{0, 1\}$ is called a *bit vector* of dimension n. We use BV_n to denote the set of all bit vectors of dimension n. The concrete domain D_2 is defined by setting $\Delta_{\mathsf{D}_2} := \bigcup_{i>0} \mathsf{BV}_i$ and Φ_{D_2} to the (smallest) set containing the following predicates:

– a unary predicate $\top_{\mathsf{D}_2}$ with $(\top_{\mathsf{D}_2})^{\mathsf{D}_2} = \Delta_{\mathsf{D}_2}$ and a unary predicate $\bot_{\mathsf{D}_2}$ with $(\bot_{\mathsf{D}_2})^{\mathsf{D}_2} = \emptyset$;

– for every $k, i \in \mathbb{N}$ with $i < k$, unary predicates $\mathsf{bit0}_k^i$ and $\mathsf{bit1}_k^i$ with

$$(\mathsf{bit}n_k^i)^{\mathsf{D}_2} = \{v \in \Delta_{\mathsf{D}_2} \mid v \in \mathsf{BV}_k \text{ and } v(i) = n\}.$$

Moreover, unary predicates $\overline{\mathsf{bit0}_k^i}$ and $\overline{\mathsf{bit1}_k^i}$ with $(\overline{\mathsf{bit}n_k^i})^{\mathsf{D}_2} = \Delta_{\mathsf{D}_2} \setminus (\mathsf{bit}n_k^i)^{\mathsf{D}_2}$.

$\Diamond$

It is obvious that D_2 satisfies Conditions 1 and 2 of admissibility, and, as we will see now, an algorithm for D_2-satisfiability is easily devised.

Proposition 16. D_2-*satisfiability is in* PTime.

Proof. Let c be a D_2-conjunction. We show that c is satisfiable iff none of the following conditions applies:

1. c contains a conjunct $\bot_{\mathsf{D}_2}(x)$;
2. c contains conjuncts $\mathsf{bit0}^i_k(x)$ and $\mathsf{bit1}^i_k(x)$;
3. c contains conjuncts $\mathsf{bit}n^i_k(x)$ and $\mathsf{bit}m^j_\ell(x)$ with $k \neq \ell$;
4. c contains conjuncts $\mathsf{bit}n^i_k(x)$ and $\overline{\mathsf{bit}n^i_k}(x)$;
5. c contains conjuncts $\mathsf{bit}n^i_k(x)$, $\overline{\mathsf{bit0}^j_k}(x)$, and $\overline{\mathsf{bit1}^j_k}(x)$.

It is easily seen that c is unsatisfiable if one of the conditions applies. Assume now that Conditions 1 to 5 do *not* apply to c and let X be the set of variables used in c. For each $x \in X$, set $t(x) = k$ if $\mathsf{bit}n^i_k(x) \in c$ for some $n, i \in \mathbb{N}$.[3] If $\mathsf{bit}n^i_k(x) \notin c$ for all $n, i, k \in \mathbb{N}$, then set $t(x) = r$ for some r not appearing as an index $\cdot_r$ to a predicate in c. The mapping t is well-defined since c is finite, Condition 3 does not apply, and the only predicates available are $\mathsf{bit}n^i_k(\cdot)$, $\bot_{\mathsf{D}_2}(\cdot)$, and $\top_{\mathsf{D}_2}(\cdot)$. We define a solution δ for c as follows: for each $x \in X$, set $\delta(x)$ to the bit vector $v \in \mathsf{BV}_{t(x)}$ in which the i'th bit is 1 if $\mathsf{bit1}^i_{t(x)}(x) \in c$ or $\overline{\mathsf{bit0}^i_{t(x)}}(x) \in c$, and 0 otherwise. It remains to prove that δ is indeed a solution for c:

- Let $\mathsf{bit0}^i_k(x) \in c$. Then $t(x) = k$ and thus $\delta(x) \in \mathsf{BV}_k$. Since Condition 2 does not apply, we have $\mathsf{bit1}^i_k(x) \notin c$. Moreover, non-applicability of Condition 4 implies $\overline{\mathsf{bit0}^i_k}(x) \notin c$. By definition of δ, the i'th bit of $\delta(x)$ is thus 0.
- Let $\mathsf{bit1}^i_k(x) \in c$. Then $t(x) = k$ and $\delta(x) \in \mathsf{BV}_k$. By definition of δ, the i'th bit of $\delta(x)$ is 1.
- Let $\overline{\mathsf{bit0}^i_k}(x) \in c$. If $t(x) \neq k$, then $\delta(x) \notin \mathsf{BV}_k$. Thus $\delta(x) \in (\overline{\mathsf{bit0}^i_k})^{\mathsf{D}_2}$ and we are done. If $t(x) = k$, then the i'th bit of $\delta(x)$ is 1 by definition of δ and thus again $\delta(x) \in (\overline{\mathsf{bit0}^i_k})^{\mathsf{D}_2}$.
- Let $\overline{\mathsf{bit1}^i_k}(x) \in c$. If $t(x) \neq k$, then $\delta(x) \notin \mathsf{BV}_k$ and we are done. If $t(x) = k$, then $\mathsf{bit}n^j_k(x) \in c$ for some $n, j \in \mathbb{N}$. Since Condition 5 does not apply, we thus have $\overline{\mathsf{bit0}^i_k}(x) \notin c$. Moreover, non-applicability of Condition 4 yields $\mathsf{bit1}^i_k(x) \notin c$. Thus, by definition of δ, the i'th bit of $\delta(x)$ is 0.

It is obvious that the listed properties can be checked in polynomial time. ❑

[3]We use "$P(x) \in c$" as an abbreviation for "$P(x)$ is a conjunct in c".

The last concrete domain D_3 is used in the reduction of the NExpTime-complete domino problem to $\mathcal{ALCO}(\mathsf{D}_3)$-concept satisfiability. In this reduction, the concrete domain D_3 serves two main purposes: firstly, we store the *whole* $2^{n+1} \times 2^{n+1}$-torus in a *single* element of Δ_{D_3}. Secondly, positions in the torus are addressed using elements of Δ_{D_3}. Thus, the set Δ_{D_3} contains two different types of elements: "domino arrays" for representing tori and "vectors" for addressing positions in domino arrays. Intuitively, vectors of length $n+1$ can be understood as bit vectors representing the binary coding of numbers between 0 and $2^{n+1}-1$, i.e. x-positions and y-positions in the torus. Domino arrays are then functions mapping pairs of vectors to natural numbers (representing tile types). However, as is discussed below, it is advisable to define D_3 in a slightly more general way by admitting vectors of natural numbers rather than bit vectors.

Definition 17 (Concrete Domain D_3). For every $k \in \mathbb{N}$, a function $v : \{0, \ldots, k-1\} \to \mathbb{N}$ is called a *vector* of dimension k. We use VE_k to denote the set of all vectors of dimension k. For every $k \in \mathbb{N}$, a function $k : \mathsf{VE}_k \times \mathsf{VE}_k \to \mathbb{N}$ is called a *domino array* of dimension k. We use DA_k to denote the set of all domino arrays of dimension k. The concrete domain D_3 is defined by setting $\Delta_{\mathsf{D}_3} := \bigcup_{i>0} \mathsf{VE}_i \cup \bigcup_{i>0} \mathsf{DA}_i$ and Φ_{D_3} to the (smallest) set containing the following predicates:

- unary predicates $\top_{\mathsf{D}_3}$ with $(\top_{\mathsf{D}_3})^{\mathsf{D}_3} = \Delta_{\mathsf{D}_3}$ and $\bot_{\mathsf{D}_3}$ with $(\bot_{\mathsf{D}_3})^{\mathsf{D}_3} = \emptyset$;
- for every $k, i \in \mathbb{N}$ with $i < k$, unary predicates $\mathsf{pos0}^i_k$ and $\mathsf{pos1}^i_k$ with

  $$(\mathsf{pos}n^i_k)^{\mathsf{D}_3} = \{v \in \Delta_{\mathsf{D}_3} \mid v \in \mathsf{VE}_k \text{ and } v(i) = n\}$$

  and unary predicates $\overline{\mathsf{pos0}^i_k}$ and $\overline{\mathsf{pos1}^i_k}$ with $(\overline{\mathsf{pos}n^i_k})^{\mathsf{D}_3} = \Delta_{\mathsf{D}_3} \setminus (\mathsf{pos}n^i_k)^{\mathsf{D}_3}$.
- for every $k, i \in \mathbb{N}$, a predicate tile^i_k of arity 3 with

  $$(\mathsf{tile}^i_k)^{\mathsf{D}_3} = \{(v_x, v_y, d) \mid v_x, v_y \in \mathsf{VE}_k,\ d \in \mathsf{DA}_k, \text{ and } d(v_x, v_y) = i\}$$

  and a predicate $\overline{\mathsf{tile}^i_k}$ of arity 3 with $(\overline{\mathsf{tile}^i_k})^{\mathsf{D}_3} = (\Delta_{\mathsf{D}_3})^3 \setminus (\mathsf{tile}^i_k)^{\mathsf{D}_3}$.

$\diamondsuit$

The reason for using vectors of natural numbers rather than bit vectors in the definition of D_3 is that we want D_3-satisfiability to be of low complexity, preferably in PTime: consider the D_3-conjunction

$$\begin{array}{l}\mathsf{pos0}^0_2(x) \wedge \mathsf{pos0}^0_2(y) \wedge \mathsf{pos0}^0_2(z) \wedge \\ \mathsf{pos0}^0_2(v) \wedge \mathsf{pos0}^1_2(v) \wedge \\ \mathsf{tile}^7_2(x, v, d) \wedge \mathsf{tile}^8_2(y, v, d) \wedge \mathsf{tile}^9_2(z, v, d).\end{array}$$

If we use bit vectors rather than vectors of natural numbers, then at least two out of the three variables x, y, and z must take the same value and thus the above conjunction is unsatisfiable. It seems unlikely that this kind of inconsistency can be detected in polynomial time. This problem is eliminated by using vectors of natural numbers in the definition of D_3 (but enforcing them to be bit vectors in the reduction): in this case, the above conjunction is clearly satisfiable. The following proposition is proved in [39]:

Proposition 18. *D_3-satisfiability is in* PTime.

3.3 NExpTime-hardness of $\mathcal{ALCK}(\mathcal{D})$ with Boolean Key Boxes

In this section, we prove two NExpTime-lower bounds for $\mathcal{ALCK}(\mathcal{D})$-concept satisfiability w.r.t. Boolean key boxes by reducing the NExpTime-complete domino problem introduced in the previous section. The first reduction uses the very simple concrete domain D_1, but depends on n-ary key definitions. The second reduction uses the slightly more complex (and more unnatural) concrete domain D_2, but only needs unary key definitions. As we will see, the two reductions yield different, incomparable results.

We first reduce the NExpTime-complete domino problem to $\mathcal{ALCK}(\mathsf{D}_1)$-concept satisfiability w.r.t. Boolean key boxes admitting n-ary key definitions. Each domino system $\mathfrak{D} = (T, H, V)$ with initial condition $a = a_0, \dots, a_{n-1}$ is translated into an $\mathcal{ALCK}(\mathsf{D}_1)$-concept $C_{\mathfrak{D},a}$ as displayed in Figure 3. Names such as TreeX and TreeY are used as abbreviations which should not be confused with so-called TBoxes (see Section 4.2 for the definition of TBoxes). We use $\forall R^i.C$ as an abbreviation for the n-fold nesting $\forall R. \cdots \forall R.C$. The names xpos_i and ypos_i used in the figure denote concrete features. In the definition of the Init concept, for each $n \in \mathbb{N}$, $\mathsf{bit}_i(n)$ is supposed to denote the i'th bit of the binary representation of n. We claim that $C_{\mathfrak{D},a}$ is satisfiable w.r.t. the key box

$$\{(\mathsf{xpos}_0, \dots, \mathsf{xpos}_n, \mathsf{ypos}_0, \dots, \mathsf{ypos}_n \text{ keyfor } \top)\}$$

iff there exists a solution for $\mathfrak{D}$ and a. To substantiate this claim, let us walk through the reduction and explain the various parts of the concept $C_{\mathfrak{D},a}$. The first step towards understanding the structure of models of $C_{\mathfrak{D},a}$ (which is the key to understanding the reduction itself) is to note that the purpose of the first line of $C_{\mathfrak{D},a}$ is to enforce a tree structure of depth $2(n+1)$, whose leaves correspond to positions in the $2^{n+1} \times 2^{n+1}$-torus. More precisely, the TreeX concept guarantees that, in every model $\mathcal{I}$ of $C_{\mathfrak{D},a}$, there exists a binary tree of depth $n+1$. Moreover, the DistX_k concepts (there exists one for each $k \in \{0, \dots, n\}$) ensure that the leaves of this tree are binarily numbered (from 0 to $2^{n+1} - 1$) by the concept names $X_0, \dots, X_n$. More precisely, for a domain object $d \in \Delta_\mathcal{I}$, set

$$\mathsf{xpsn}(d) = \Sigma_{i=0}^{n} \alpha_i(d) * 2^i \quad \text{where} \quad \alpha_i(d) = \begin{cases} 1 & \text{if } d \in X_i^\mathcal{I} \\ 0 & \text{otherwise.} \end{cases}$$

The TreeX and DistX concepts ensure that there exist leaves of the tree $d_0, \dots, d_{2^{n+1}-1}$ such that $\mathsf{xpsn}(d_i) = i$. Intuitively, this numbering represents the horizontal positions in the $2^{n+1} \times 2^{n+1}$-torus. The vertical positions are coded in a similar way by the $Y_0, \dots, Y_n$ concept names. More specifically, the concepts TreeY, DistX, and DistY ensure that every d_i ($i \leq 2^{n+1} - 1$) is the root of another tree, in which (i) every node has the same "$X_0, \dots, X_n$-configuration" as its root node, and (ii) the leaves are numbered binarily using the concept names $Y_0, \dots, Y_n$ (note that the TreeY concept appears in $C_{\mathfrak{D},a}$ inside a $\forall R^{n+1}$ value restriction). Define

$$\mathsf{ypsn}(d) = \Sigma_{i=0}^{n} \beta_i(d) * 2^i \quad \text{where} \quad \beta_i(d) = \begin{cases} 1 & \text{if } d \in Y_i^\mathcal{I} \\ 0 & \text{otherwise.} \end{cases}$$

$$
\begin{aligned}
\mathsf{TreeX} &:= \exists R.X_0 \sqcap \exists R.\neg X_0 \sqcap \bigsqcap_{i=1..n} \forall R^i.(\mathsf{DistX}_{i-1} \sqcap \exists R.X_i \sqcap \exists R.\neg X_i)\\
\mathsf{TreeY} &:= \mathsf{DistX}_n \sqcap \exists R.Y_0 \sqcap \exists R.\neg Y_0 \sqcap\\
&\quad \bigsqcap_{i=1..n} \forall R^i.(\mathsf{DistY}_{i-1} \sqcap \mathsf{DistX}_n \sqcap \exists R.Y_i \sqcap \exists R.\neg Y_i)\\
\mathsf{DistX}_k &:= \bigsqcap_{i=0..k} ((X_i \to \forall R.X_i) \sqcap (\neg X_i \to \forall R.\neg X_i))\\
\mathsf{DistY}_k &:= \bigsqcap_{i=0..k} ((Y_i \to \forall R.Y_i) \sqcap (\neg Y_i \to \forall R.\neg Y_i))\\
\mathsf{TransXPos} &:= \bigsqcap_{i=0..n} (X_i \to \exists \mathsf{xpos}_i.=_1) \sqcap (\neg X_i \to \exists \mathsf{xpos}_i.=_0)\\
\mathsf{TransYPos} &:= \bigsqcap_{i=0..n} (Y_i \to \exists \mathsf{ypos}_i.=_1) \sqcap (\neg Y_i \to \exists \mathsf{ypos}_i.=_0)\\
\mathsf{Succs} &:= \exists R_x.(\mathsf{TransXPos} \sqcap \mathsf{TransYPos}) \sqcap \exists R_y.(\mathsf{TransXPos} \sqcap \mathsf{TransYPos})\\
\mathsf{XSuccOk} &:= \bigsqcap_{i=0..n} \big((Y_i \to \forall R_x.Y_i) \sqcap (\neg Y_i \to \forall R_x.\neg Y_i)\big)\\
&\quad \bigsqcap_{k=0..n} \big(\bigsqcap_{j=0..k} X_j\big) \to \big((X_k \to \forall R_x.\neg X_k) \sqcap (\neg X_k \to \forall R_x.X_k)\big)\\
&\quad \bigsqcap_{k=0..n} \big(\bigsqcup_{j=0..k} \neg X_j\big) \to \big((X_k \to \forall R_x.X_k) \sqcap (\neg X_k \to \forall R_x.\neg X_k)\big)\\
\mathsf{YSuccOk} &:= \bigsqcap_{i=0..n} \big((X_i \to \forall R_y.X_i) \sqcap (\neg X_i \to \forall R_y.\neg X_i)\big)\\
&\quad \bigsqcap_{k=0..n} \big(\bigsqcap_{j=0..k} Y_j\big) \to \big((Y_k \to \forall R_y.\neg Y_k) \sqcap (\neg Y_k \to \forall R_y.Y_k)\big)\\
&\quad \bigsqcap_{k=0..n} \big(\bigsqcup_{j=0..k} \neg Y_j\big) \to \big((Y_k \to \forall R_y.Y_k) \sqcap (\neg Y_k \to \forall R_y.\neg Y_k)\big)\\
\mathsf{Label} &:= \bigsqcup_{i\in T} D_i \;\sqcap \bigsqcap_{i,j\in T, i\neq j} \neg(D_i \sqcap D_j)\\
\mathsf{CheckMatch} &:= \bigsqcup_{(i,j)\in H} (D_i \sqcap \forall R_x.D_j) \;\sqcap \bigsqcup_{(i,j)\in V} (D_i \sqcap \forall R_y.D_j)\\
\mathsf{Init} &:= \bigsqcap_{i=0..n-1} \Big(\big(\bigsqcap_{j=0..n,\mathsf{bit}_j(i)=0} \neg X_j \;\sqcap \bigsqcap_{j=0..n,\mathsf{bit}_j(i)=1} X_j \;\sqcap \bigsqcap_{j=0..n} \neg Y_j\big) \to D_{a_i}\Big)\\
C_{\mathfrak{D},a} &:= \mathsf{TreeX} \sqcap \forall R^{n+1}.\mathsf{TreeY}\\
&\quad \sqcap \forall R^{2(n+1)}.(\mathsf{TransXPos} \sqcap \mathsf{TransYPos} \sqcap \mathsf{Succs} \sqcap \mathsf{XSuccOk} \sqcap \mathsf{YSuccOk})\\
&\quad \sqcap \forall R^{2(n+1)}.(\mathsf{Label} \sqcap \mathsf{CheckMatch} \sqcap \mathsf{Init})
\end{aligned}
$$

Figure 3: The $\mathcal{ALCK}(\mathsf{D}_1)$ reduction concept $C_{\mathfrak{D},a}$.

In the set of leave nodes of all the trees enforced by the TreeY concept, there exists an[4] object $e_{i,j} \in \Delta_\mathcal{I}$ for each $i, j < 2^{n+1}$ such that $\mathsf{xpsn}(e_{i,j}) = i$ and $\mathsf{ypsn}(e_{i,j}) = j$, i.e., each $e_{i,j}$ represents the position (i, j) in the $2^{n+1} \times 2^{n+1}$-torus.

The next step is to translate the individual bits of the numbering of the $e_{i,j}$-concepts,

[4]So far, we do not care if there is more than one such object.

which are up to now represented by concept names, into concrete domain values. This is done by the $\mathsf{TransXPos}$ and $\mathsf{TransYPos}$ concepts which ensure that, for all $\ell \leq n$, we have $\mathsf{xpos}_\ell^\mathcal{I}(e_{i,j}) = 0$ if $e_{i,j} \in \neg X_\ell$, $\mathsf{xpos}_\ell^\mathcal{I}(e_{i,j}) = 1$ if $e_{i,j} \in X_\ell$, and similar for ypos_ℓ and Y_ℓ. Since $\mathcal{I}$ is a model for the key box

$$\{(\mathsf{xpos}_0, \dots, \mathsf{xpos}_n, \mathsf{ypos}_0, \dots, \mathsf{ypos}_n \text{ keyfor } \top)\},$$

grid positions are *uniquely* represented by domain elements from $(\mathsf{TransXPos} \sqcap \mathsf{TransYPos})^\mathcal{I}$, i.e., if $d, e \in (\mathsf{TransXPos} \sqcap \mathsf{TransYPos})^\mathcal{I}$ such that $\mathsf{xpsn}(d) = \mathsf{xpsn}(e)$ and $\mathsf{ypsn}(d) = \mathsf{yxpsn}(e)$, then $d = e$. This fact is used in the concepts Succs, $\mathsf{XSuccOk}$, and $\mathsf{YSuccOk}$ to enforce that, for the two roles R_x and R_y and each $i, j \leq n$, the following holds:

$$\begin{array}{rcl} R_x^\mathcal{I} \cap (\{e_{i,j}\} \times \Delta_\mathcal{I}) & = & \{(e_{i,j}, e_{(i \oplus_{2^{n+1}} 1),j}\} \\ R_y^\mathcal{I} \cap (\{e_{i,j}\} \times \Delta_\mathcal{I}) & = & \{(e_{i,j}, e_{i,(j \oplus_{2^{n+1}} 1)}\}. \end{array} \tag{$*$}$$

The Succs concept ensures that, for each $e_{i,j}$, there exists an R_x-successor and an R_y-successor, and that both are in $(\mathsf{TransXPos} \sqcap \mathsf{TransYPos})^\mathcal{I}$. Let d be an R_x-successor of $e_{i,j}$. Then the $\mathsf{XSuccOk}$ concept ensures that $\mathsf{xpsn}(d) = i \oplus_{2^{n+1}} 1$ and $\mathsf{ypsn}(d) = j$. Before we explain how it does this, let us note that, since all $e_{i,j}$ are in $(\mathsf{TransXPos} \sqcap \mathsf{TransYPos})^\mathcal{I}$ and the grid positions are uniquely represented by elements of $(\mathsf{TransXPos} \sqcap \mathsf{TransYPos})^\mathcal{I}$, this implies $d = e_{(i \oplus_{2^{n+1}} 1),j}$ which shows that the upper line of $(*)$ does indeed hold.

Let us now consider the $\mathsf{XSuccOk}$ concept in some more detail. It is essentially the DL-formulation of the well-known propositional formula

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

which encodes incrementation modulo 2^{n+1}, i.e., if t is the number (binarily) encoded by the propositional variables $x_0, \dots, x_n$ and t' is the number encoded by the propositional variables $x'_0, \dots, x'_n$, then we have $t' = t + 1$ modulo 2^{n+1}, c.f. [13]. Taking into account the $\forall R_x$ quantifiers in $\mathsf{XSuccOk}$, it is readily checked that this concept has just the desired effect: to ensure that, for every R_x-successor d of $e_{i,j}$, we have $\mathsf{xpsn}(x) = \mathsf{xpsn}(e_{(i \oplus_{2^{n+1}} 1),j}) = i \oplus_{2^{n+1}} 1$. The explanation of $\mathsf{YSuccOk}$ and how it enforces the lower line of $(*)$ is omitted since it is analogous to the $\mathsf{XSuccOk}$ case.

It remains to ensure that every grid position is labeled with precisely one tile and that the initial condition as well as the horizontal and vertical matching conditions are satisfied. The tiles are represented by concept names D_i (where i is from the set of tiles T) and the described tasks are accomplished in the standard way by the concepts Label, Init, and $\mathsf{CheckMatch}$.

It is interesting to note that the reduction concept is path-free and the key box is simple, i.e., path-free and Boolean. Path-freeness of concepts is often used to tame the complexity of description logics with concrete domains (although it largely sacrifices their expressive power) [38, 7, 24, 31]. For example, if $\mathcal{ALC}(\mathcal{D})$ is augmented with so-called general TBoxes, then reasoning with arbitrary concepts is undecidable while reasoning with path-free concepts is ExpTime-complete if $\mathcal{D}$ is admissible and

$\mathcal{D}$-satisfiability is in ExpTime [37]. This "taming approach" does not work in the presence of key boxes since, as we have just seen, both reasoning with arbitrary and with path-free $\mathcal{ALCK}(\mathcal{D})$-concepts is (under some natural assumptions) NExpTime-hard.

Since the size of $C_{\mathfrak{D},a}$ and of the used key box is clearly polynomial in n, we obtain the following proposition.

Proposition 19. *The satisfiability of path-free $\mathcal{ALCK}(\mathsf{D}_1)$-concepts w.r.t. simple key boxes is* NExpTime*-hard.*

In [40], it is proved that (non path-free) $\mathcal{ALC}(\mathcal{D})$-concept satisfiability is PSpace-complete if $\mathcal{D}$-satisfiability is in PSpace. In particular, it thus follows from Proposition 14 that $\mathcal{ALC}(\mathsf{D}_1)$-concept satisfiability is PSpace-complete. Thus, there is a rather dramatic increase of complexity if key boxes are added to $\mathcal{ALC}(\mathsf{D}_1)$. To stress that this increase is due to the key boxes themselves and not to the complexity of D_1-satisfiability, we reformulate Proposition 19:

Theorem 20. *There exists a concrete domain $\mathcal{D}$ such that $\mathcal{D}$-satisfiability is in* PTime *and satisfiability of path-free $\mathcal{ALCK}(\mathcal{D})$-concepts w.r.t. simple key boxes is* NExpTime*-hard.*

Since concept satisfiability can be reduced to concept non-subsumption as noted in Section 2, we obtain a corresponding co-NExpTime-hardness bound for the subsumption of path-free $\mathcal{ALCK}(\mathcal{D})$-concepts w.r.t. simple key boxes.

Although, due to its very low expressivity, the concrete domain D_1 itself is not very natural for knowledge representation, it is a fragment of many concrete domains that have been proposed in the literature [5, 23, 38, 40]. Indeed, the presented reduction strategy can be adapted to quite many "standard" concrete domains. Let us formulate a (very weak) condition that a concrete domain must satisfy in order for the presented reduction strategy to be applicable.

Theorem 21. *Let $\mathcal{D}$ be a concrete domain. If there exist $a, b \in \Delta_\mathcal{D}$ and $P_1, P_2 \in \Phi_\mathcal{D}$ such that $P_1^\mathcal{D} = \{a\}$ and $P_2^\mathcal{D} = \{b\}$, then the satisfiability of path-free $\mathcal{ALCK}(\mathcal{D})$-concepts w.r.t. simple key boxes is* NExpTime*-hard.*

Again, a corresponding co-NExpTime-hardness result for concept subsumption is easily obtained.

We now present the second NExpTime-hardness result for $\mathcal{ALCK}(\mathcal{D})$-concept satisfiability. This time, we reduce the NExpTime-complete domino problem to the satisfiability of path-free $\mathcal{ALCK}(\mathsf{D}_2)$-concepts w.r.t. simple *unary* key boxes. The reduction is very similar to the previous one and we only discuss the differences. In the first reduction, we represented the individual bits of grid positions by individual concrete features xpos_i and ypos_i. This approach led to a n-ary key box. To replace it by a unary key box, in the second reduction, we represent entire positions in the torus (i, j) by the bit vectors provided by the concrete domain D_2. The modified reduction concept $C_{\mathfrak{D},a}$ can be found in Figure 4, where bv denotes a concrete feature and the concepts TreeX, TreeY, DistX_k, DistY_k, $\mathsf{XSuccOk}$, $\mathsf{YSuccOk}$, Label, $\mathsf{CheckMatch}$, and Init are defined as in

$$\begin{aligned}
\mathsf{Succs2} &:= \exists R_x.\mathsf{TransPos} \sqcap \exists R_y.\mathsf{TransPos} \\
\mathsf{TransPos} &:= \bigsqcap_{i=0..n} \big((X_i \rightarrow \exists \mathsf{bv}.\mathsf{bit1}^i_{2(n+1)}) \sqcap \neg X_i \rightarrow \exists \mathsf{bv}.\mathsf{bit0}^i_{2(n+1)})\big) \sqcap \\
&\quad \bigsqcap_{i=0..n} \big((Y_i \rightarrow \exists \mathsf{bv}.\mathsf{bit1}^{n+i+1}_{2(n+1)}) \sqcap \neg Y_i \rightarrow \exists \mathsf{bv}.\mathsf{bit0}^{n+i+1}_{2(n+1)})\big) \\
C_{\mathfrak{D},a} &:= \mathsf{TreeX} \sqcap \forall R^{n+1}.\mathsf{TreeY} \\
&\quad \sqcap \forall R^{2(n+1)}.(\mathsf{TransPos} \sqcap \mathsf{Succs2} \sqcap \mathsf{XSuccOk} \sqcap \mathsf{YSuccOk}) \\
&\quad \sqcap \forall R^{2(n+1)}.(\mathsf{Label} \sqcap \mathsf{CheckMatch} \sqcap \mathsf{Init})
\end{aligned}$$

Figure 4: The $\mathcal{ALCK}(\mathsf{D}_2)$ reduction concept $C_{\mathfrak{D},a}$.

Figure 3. The translation of the position in the torus encoded by $X_0, \ldots, X_n, Y_0, \ldots, Y_n$ into a bit vector is done by the TransPos concept in a straightforward manner. Given what was said about the first reduction, it is not hard to see that $C_{\mathfrak{D},a}$ is satisfiable w.r.t. the key box

$$\{(\mathsf{bv} \text{ keyfor } \top)\}$$

iff there exists a solution for $\mathfrak{D}$ and a. We thus obtain the following proposition.

Proposition 22. *The satisfiability of path-free $\mathcal{ALCK}(\mathsf{D}_2)$-concepts w.r.t. simple unary key boxes is* NExpTime*-hard.*

Again, we relate the NExpTime lower bound to the complexity of D_2-complexity, which is determined in Theorem 16.

Theorem 23. *There exists a concrete domain $\mathcal{D}$ such that $\mathcal{D}$-satisfiability is in* PTime *and the satisfiability of path-free $\mathcal{ALCK}(\mathcal{D})$-concepts w.r.t. simple unary key boxes is* NExpTime*-hard.*

As for the previous lower bounds, we obtain a corresponding co-NExpTime-hardness bound for concept subsumption.

Since the elements of Δ_{D_2} are bit vectors, the concrete domain D_2 cannot be considered a natural choice for many application areas. But, in the reduction, D_2 can be replaced by several natural concrete domains. The central observation is that we use bit vectors only to injectively translate sequences of bits into values of the concrete domain, i.e., we translate sequences of $2(n+1)$ bits (represented by the concept names $X_0, \ldots, X_n$ and $Y_0, \ldots, Y_n$) into elements of Δ_{D_2} such that, for *distinct* sequences, the results of the translation are also distinct. Due to this restricted use of bit vectors, there are several ways to replace them by natural numbers. For example, we could define a new TransPos concept such that

$$s^{\mathcal{I}}_{2n+1}(d) = \mathsf{xpsn}(d) + 2^{n+1}\mathsf{ypsn}(d)$$

as follows:

$$\begin{aligned}\mathsf{TransPos}' := {} & \exists \mathsf{zero}.{=_0} \sqcap (\neg X_0 \rightarrow \exists s_0.{=_0}) \sqcap (X_0 \rightarrow \exists s_0.{=_1}) \\ & \sqcap \bigsqcap_{i=1..n} \Big(\exists t_i.{=_{2^i}} \sqcap \big(\neg X_i \rightarrow \exists (s_{i-1}, \mathsf{zero}, s_i).{+}\big) \sqcap \big(X_i \rightarrow \exists (s_{i-1}, t_i, s_i).{+}\big)\Big) \\ & \sqcap \bigsqcap_{i=0..n} \Big(\exists t_{n+i+1}.{=_{2^{n+i+1}}} \sqcap \big(\neg Y_i \rightarrow \exists (s_{n+i}, \mathsf{zero}, s_{n+i+1}).{+}\big) \sqcap \\ & \qquad \big(Y_i \rightarrow \exists (s_{n+i}, t_{n+i+1}, s_{n+i+1}).{+}\big)\Big)\end{aligned}$$

where zero and the s_i and t_i denote concrete features, $=_k$ (with $k \in \mathbb{N}$) denotes a unary predicate with the obvious extension, and $+$ denotes a ternary addition predicate. It is easy to check that, whenever two objects $d, e \in \mathsf{TransPos}^{\mathcal{I}}$ do *not* agree on the interpretation of the $X_0, \ldots, X_n, Y_0, \ldots, Y_n$, then $s_{2n+1}^{\mathcal{I}}(d) \neq s_{2n+1}^{\mathcal{I}}(e)$, and thus the key box

$$\{(s_{2n+1} \text{ keyfor } \top)\}$$

can be used for the reduction. The size of TransPos′ is obviously polynomial in n if the numbers k appearing in $=_k$ predicates are coded in binary. We thus obtain the following theorem:

Theorem 24. *Let $\mathcal{D}$ be a concrete domain such that*

1. $\mathbb{N} \subseteq \Delta_{\mathcal{D}}$,
2. *$\Phi_{\mathcal{D}}$ contains a predicate $=_k$ with $(=_k)^{\mathcal{D}} = \{n\}$ for each $k \in \mathbb{N}$ where the size of (the representation of) $=_k$ is logarithmic in k, and*
3. *$\Phi_{\mathcal{D}}$ contains a predicate $+$ with $(+)^{\mathcal{D}} \cap \{(k_1, k_2, x) \mid k_1, k_2 \in \mathbb{N}$ and $x \in \Delta_{\mathcal{D}}\} = \{(k_1, k_2, k_1 + k_2) \mid k_1, k_2 \in \mathbb{N}\}$.*

Then the satisfiability of path-free $\mathcal{ALCK}(\mathcal{D})$-concepts w.r.t. simple unary key boxes is NExpTime*-hard.*

For example, this theorem yields NExpTime-lower bounds for $\mathcal{ALCK}(\mathcal{D})$ instantiated with the concrete domains proposed in [5, 23, 38, 40]. An alternative to adding the addition predicate is to use multiplication to injectively translate sequences of bits into natural numbers. More precisely, fix a sequence of distinct prime numbers $p_0, \ldots, p_{2n+1}$ and define another version of TransPos as follows:

$$\begin{aligned}\mathsf{TransPos}'' := {} & \exists \mathsf{one}.{=_1} \sqcap (\neg X_0 \rightarrow \exists s_0.{=_0}) \sqcap (X_0 \rightarrow \exists s_0.{=_1}) \\ & \sqcap \bigsqcap_{i=1..n} \Big(\exists t_i.{=_{p_i}} \sqcap \big(\neg X_i \rightarrow \exists (s_{i-1}, \mathsf{one}, s_i).{*}\big) \sqcap \big(X_i \rightarrow \exists (s_{i-1}, t_i, s_i).{*}\big)\Big) \\ & \sqcap \bigsqcap_{i=0..n} \Big(\exists t_{n+i+1}.{=_{p_{n+i+1}}} \sqcap \big(\neg Y_i \rightarrow \exists (s_{n+i}, \mathsf{one}, s_{n+i+1}).{*}\big) \sqcap \\ & \qquad \big(Y_i \rightarrow \exists (s_{n+i}, t_{n+i+1}, s_{n+i+1}).{*}\big)\Big)\end{aligned}$$

where $*$ is a ternary multiplication predicate. Since the factorization of natural numbers into prime numbers is unique, we can again use the key box

$$\{(s_{2n+1} \text{ keyfor } \top)\}$$

for the reduction. Moreover, it is well-known that the k'th prime is polynomial in k [21], and thus the size of the concept $\mathsf{TransPos}''$ is polynomial in n even if the numbers k in $=_k$ predicates are coded unarily. We thus obtain another theorem concerning quite natural concrete domains:

Theorem 25. *Let $\mathcal{D}$ be a concrete domain such that*

1. $\mathbb{N} \subseteq \Delta_{\mathcal{D}}$,
2. *$\Phi_{\mathcal{D}}$ contains a predicate $=_k$ with $(=_k)^{\mathcal{D}} = \{k\}$ for each $k \in \mathbb{N}$, and*
3. *$\Phi_{\mathcal{D}}$ contains a predicate $*$ with $(*)^{\mathcal{D}} \cap \{(k_1, k_2, x) \mid k_1, k_2 \in \mathbb{N}$ and $x \in \Delta_{\mathcal{D}}\} = \{(k_1, k_2, k_1 \cdot k_2) \mid k_1, k_2 \in \mathbb{N}\}$.*

Then the satisfiability of path-free $\mathcal{ALCK}(\mathcal{D})$-concepts w.r.t. simple unary key boxes is NExpTime-*hard.*

3.4 NExpTime-hardness of $\mathcal{ALCO}(\mathcal{D})$

As we already pointed out in Section 1, the relationship between concrete domain keys and nominals is rather close: the latter can be "simulated" by the former if the concrete domain provides for predicates that can be used to uniquely describe elements of $\Delta_{\mathcal{D}}$. For example, in $\mathcal{ALCO}(\mathsf{D}_1)$ the concept $\exists g.{=_0}$ behaves as a nominal if we use the key definition (g keyfor $\top$). We can even define n nominals using n single concrete feature in unary-key definitions. In the logics $\mathcal{ALCO}(\mathsf{D}_2)$ and $\mathcal{ALCO}(\mathsf{D}_3)$, a single concrete feature and unary key definitions are sufficient to simulate an arbitrary number of nominals: for example, in $\mathcal{ALCK}(\mathsf{D}_2)$ the concept $C = \exists g.\mathsf{bit0}^0_2 \sqcap \exists g.\mathsf{bit1}^1_2$ uniquely describes the bit vector $(0, 1) \in \mathsf{BV}_2 \subseteq \Delta_{\mathsf{D}_2}$, i.e., $a \in C^{\mathcal{I}}$ implies $g^{\mathcal{I}}(a) = (0, 1)$. Obviously, any other bit vector (of any length!) can be described in a similar way. This illustrates that, for most non-trivial concrete domains $\mathcal{D}$, the logic $\mathcal{ALCK}(\mathcal{D})$ is (at least) as expressive as $\mathcal{ALCO}(\mathcal{D})$. Although the converse does not hold, the expressive power of $\mathcal{ALCO}(\mathcal{D})$ is still sufficient to prove NExpTime-hardness of concept satisfiability, provided that a suitable concrete domain $\mathcal{D}$ is used. In this section, we reduce the NExpTime-complete domino-problem to $\mathcal{ALCO}(\mathsf{D}_3)$-concept satisfiability.

Again, let $\mathfrak{D} = (T, H, V)$ be a domino system and $a = a_0, \ldots, a_{n-1}$ an initial condition. Then the reduction concept $C_{\mathfrak{D},a}$ is defined as in Figure 5, where bvx, bvy, bvxs, bvys, and darr denote concrete features, N denotes a nominal, and the concepts TreeX, TreeY, DistX_k, and DistY_k are defined as in Figure 3. As in the previous reductions, we now give a detailed explanation of the reduction strategy to show that $C_{\mathfrak{D},a}$ is satisfiable iff there exists a solution for $\mathfrak{D}$ and a.

Let $\mathcal{I}$ be a model for $C_{\mathfrak{D},a}$. To explain the structure of $\mathcal{I}$, which is the key to understanding the reduction strategy, it is convenient to start with the first line of $C_{\mathfrak{D},a}$. As in the previous reductions, the TreeX and TreeY concepts are used to ensure that $\mathcal{I}$ contains a tree-shaped substructure of depth $n + 1$ whose leaf nodes are the roots of additional trees of depth $n + 1$ such that the set of the leafs of the latter trees correspond to the positions in the $2^{n+1} \times 2^{n+1}$-torus, i.e., for each position, there is

$$
\begin{aligned}
\mathsf{Nominal} &:= \exists f.N \\
\mathsf{XSucc} &:= \bigsqcap_{k=0..n} \big(\bigsqcap_{j=0..k} X_j\big) \rightarrow (X_k \leftrightarrow \neg X'_k) \sqcap \bigsqcap_{k=0..n} \big(\bigsqcup_{j=0..k} \neg X_j\big) \rightarrow (X_k \leftrightarrow X'_k) \\
\mathsf{YSucc} &:= \bigsqcap_{k=0..n} \big(\bigsqcap_{j=0..k} Y_j\big) \rightarrow (Y_k \leftrightarrow \neg Y'_k) \sqcap \bigsqcap_{k=0..n} \big(\bigsqcup_{j=0..k} \neg Y_j\big) \rightarrow (Y_k \leftrightarrow Y'_k) \\
\mathsf{TransXPos} &:= \bigsqcap_{i=0..n} \big((X_i \rightarrow \exists \mathsf{bvx}.\mathsf{pos1}^{i}_{2(n+1)}) \sqcap \neg X_i \rightarrow \exists \mathsf{bvx}.\mathsf{pos0}^{i}_{2(n+1)})\big) \\
&\ \bigsqcap_{i=0..n} \big((Y_i \rightarrow \exists \mathsf{bvx}.\mathsf{pos1}^{n+i+1}_{2(n+1)}) \sqcap \neg Y_i \rightarrow \exists \mathsf{bvx}.\mathsf{pos0}^{n+i+1}_{2(n+1)})\big) \\
\mathsf{TransYPos} &:= \bigsqcap_{i=0..n} \big((X_i \rightarrow \exists \mathsf{bvy}.\mathsf{pos1}^{i}_{2(n+1)}) \sqcap \neg X_i \rightarrow \exists \mathsf{bvy}.\mathsf{pos0}^{i}_{2(n+1)})\big) \\
&\ \bigsqcap_{i=0..n} \big((Y_i \rightarrow \exists \mathsf{bvy}.\mathsf{pos1}^{n+i+1}_{2(n+1)}) \sqcap \neg Y_i \rightarrow \exists \mathsf{bvy}.\mathsf{pos0}^{n+i+1}_{2(n+1)})\big) \\
\mathsf{TransXSucc} &:= \bigsqcap_{i=0..n} \big((X'_i \rightarrow \exists \mathsf{bvxs}.\mathsf{pos1}^{i}_{2(n+1)}) \sqcap \neg X'_i \rightarrow \exists \mathsf{bvxs}.\mathsf{pos0}^{i}_{2(n+1)})\big) \\
&\ \bigsqcap_{i=0..n} \big((Y_i \rightarrow \exists \mathsf{bvxs}.\mathsf{pos1}^{n+i+1}_{2(n+1)}) \sqcap \neg Y_i \rightarrow \exists \mathsf{bvxs}.\mathsf{pos0}^{n+i+1}_{2(n+1)})\big) \\
\mathsf{TransYSucc} &:= \bigsqcap_{i=0..n} \big((X_i \rightarrow \exists \mathsf{bvys}.\mathsf{pos1}^{i}_{2(n+1)}) \sqcap \neg X_i \rightarrow \exists \mathsf{bvys}.\mathsf{pos0}^{i}_{2(n+1)})\big) \\
&\ \bigsqcap_{i=0..n} \big((Y'_i \rightarrow \exists \mathsf{bvys}.\mathsf{pos1}^{n+i+1}_{2(n+1)}) \sqcap \neg Y'_i \rightarrow \exists \mathsf{bvys}.\mathsf{pos0}^{n+i+1}_{2(n+1)})\big) \\
\mathsf{CheckHMatch} &:= \bigsqcup_{i,j \in H} (\exists(\mathsf{bvx}, \mathsf{bvy}, f \circ \mathsf{darr}).\mathsf{tile}^{i}_{2(n+1)} \sqcap \exists(\mathsf{bvxs}, \mathsf{bvy}, f \circ \mathsf{darr}).\mathsf{tile}^{j}_{2(n+1)}) \\
\mathsf{CheckVMatch} &:= \bigsqcup_{i,j \in V} (\exists(\mathsf{bvx}, \mathsf{bvy}, f \circ \mathsf{darr}).\mathsf{tile}^{i}_{2(n+1)} \sqcap \exists(\mathsf{bvx}, \mathsf{bvys}, f \circ \mathsf{darr}).\mathsf{tile}^{j}_{2(n+1)}) \\
\mathsf{Init2} &:= \bigsqcap_{i=0..n-1} \Big(\big(\bigsqcap_{j=0..n, \mathsf{bit}_j(i)=0} \neg X_j \sqcap \bigsqcap_{j=0..n, \mathsf{bit}_j(i)=1} X_j \sqcap \bigsqcap_{j=0..n} \neg Y_j\big) \\
&\qquad\qquad \rightarrow \exists(\mathsf{bvx}, \mathsf{bvy}, f \circ \mathsf{darr}).\mathsf{tile}^{a_i}_{2(n+1)}\Big) \\
C_{\mathfrak{D},a} &:= \mathsf{TreeX} \sqcap \forall R^{n+1}.\mathsf{TreeY} \sqcap \forall R^{2(n+1)}.\mathsf{Nominal} \\
&\ \sqcap \forall R^{2(n+1)}.(\mathsf{TransXPos} \sqcap \mathsf{TransYPos}) \\
&\ \sqcap \forall R^{2(n+1)}.(\mathsf{XSucc} \sqcap \mathsf{YSucc} \sqcap \mathsf{TransXSucc} \sqcap \mathsf{TransYSucc}) \\
&\ \sqcap \forall R^{2(n+1)}.(\mathsf{Init2} \sqcap \mathsf{CheckHMatch} \sqcap \mathsf{CheckVMatch})
\end{aligned}
$$

Figure 5: The $\mathcal{ALCO}(\mathsf{D}_3)$ reduction concept $C_{\mathfrak{D},a}$.

a leaf node representing it. The torus positions are binarily encoded by the concept names $X_0, \ldots, X_n$ and $Y_0, \ldots, Y_n$ and we use $e_{i,j}$ to refer to the leaf with $\mathsf{xpsn}(e_{i,j}) = i$ and $\mathsf{ypsn}(e_{i,j}) = j$ (c.f. Section 3.3).

As in the previous reduction, the numbers coded by $X_0, \ldots, X_n$ and $Y_0, \ldots, Y_n$ have to be translated into concrete domain values, which is done by the TransXPos and TransYPos concepts. Note that, in contrast to the $\mathcal{ALCK}(\mathsf{D}_2)$-reduction, the x-position and the y-position are not stored in the same bit vector, but rather in the two distinct ones bvx and bvy. Also in contrast to the previous reduction, the actual tiling of the

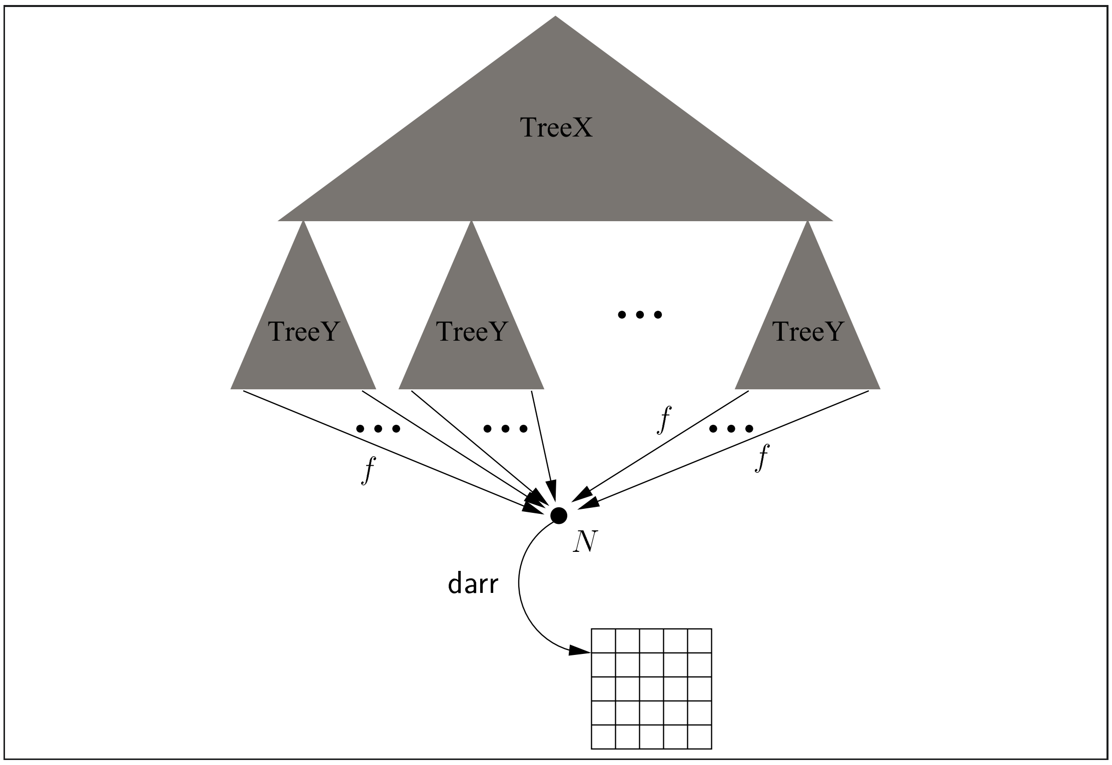

Figure 6: The structure of models of $C_{\mathfrak{D},a}$.

torus is *not* represented by the leaf nodes $e_{i,j}$, but rather by a domino array: the last conjunct in the first line of $C_{\mathfrak{D},a}$ ensures that every leaf $e_{i,j}$ is connected via the abstract feature f to the (unique) element w of $\Delta_{\mathcal{I}}$ that is in the extension of the nominal N. The domain element w is associated with a domino array via the concrete feature darr (as we shall see later, this is guaranteed by the CheckHMatch and CheckVMatch concepts). This domino array represents the tiling of the $2^{n+1} \times 2^{n+1}$-torus. Summing up, the structure of $\mathcal{I}$ is roughly as shown in Figure 6.

If the tiling is stored in a domino array, what is the purpose of the leaf node $e_{i,j}$? They are needed to enforce the initial condition and the horizontal and vertical matching condition. Let us discuss the horizontal matching condition (the vertical matching condition is enforced analogously): the XSucc concept is the DL reformulation of the propositional logic formula for incrementation modulo 2^{n+1} discussed on Page 18 and ensures that, for each $e_{i,j}$, the concept names $X'_0, \ldots, X'_n$ encode the number $i \oplus_{2^{n+1}} 1$, i.e., the horizontal position of $e_{i,j}$'s horizontal neighbor. Whereas the horizontal and vertical position of $e_{i,j}$ are stored in bvx$(e_{i,j})$ and bvy$(e_{i,j})$, we store the horizontal position $i + 1$ of $e_{i,j}$s horizontal successor in bvxs$(e_{i,j})$ (whose vertical position is j). This translation from $X'_0, \ldots, X'_n$ into bvxs is realized by the concept TransXSucc.

Finally, CheckHMatch verifies that the tiles of the positions (i, j) and $(i \oplus_{2^{n+1}} 1, j)$, which are both stored in the domino array, are compatible with the horizontal matching condition. Note that CheckHMatch also ensures that the domain element w (with $\{w\} = N^{\mathcal{I}}$) has a domino array attached via the concrete feature darr and that, for each position of the torus, the (unique!) tile stored in the domino array is from the set T. The initial condition is ensured via the Init2 concept in a similar way. We (again) use $\mathsf{bit}_j(i)$ to denote the j'th bit of the binary encoding of the natural number i.

Using the above considerations, the correctness of the reduction is readily checked. Moreover, the size of $C_{\mathfrak{D},a}$ is at most polynomial in n. Note that $C_{\mathfrak{D},a}$ is *not* path-free: paths of length two appear in the concepts CheckHMatch, CheckVMatch, and Label2. Summing up, the reduction described yields the following result:

Proposition 26. *The satisfiability of $\mathcal{ALCO}(\mathsf{D}_3)$-concepts is* NExpTime*-hard.*

Again, we relate the NExpTime lower bound to the complexity of D_3-complexity, which is determined in Theorem 18.

Theorem 27. *There exists a concrete domain $\mathcal{D}$ such that $\mathcal{D}$-satisfiability is in* PTime *and the satisfiability of $\mathcal{ALCO}(\mathcal{D})$-concepts is* NExpTime*-hard.*

Note that the reduction uses only a *single nominal* N. Hence, a single nominal is sufficient for the above hardness result. This is a dramatic increase of complexity since it was shown in [40] that satisfiability of $\mathcal{ALC}(\mathcal{D})$-concepts (i.e., without nominals and key boxes) is PSpace-complete even for concrete domains with $\mathcal{D}$-satisfiability being in PSpace. Let us once more note that we obtain a corresponding co-NExpTime-hardness bound for concept subsumption.

As in previous sections, we note that D_3 can be replaced by more natural concrete domains in the NExpTime-hardness proof presented. The idea is to represent the whole domino array by a single natural number and then to use arithmetic operations to access the individual positions: a natural number k can be viewed as a domino array by partitioning its binary representation into $2^{n+1} \cdot 2^{n+1} = 2^{2(n+1)}$ "sections" of length $\lceil \log(\#T) \rceil$, where $\#T$ denotes the cardinality of T. Each such section describes the tile of a single position in the torus. To access the sections, we need ternary predicates div for integer division and mod for computing the remainder of a division. More precisely, we replace TransXPos and TransYPos by the TransPos′ concept from Section 3.3 to translate the two numbers encoded by $X_1, \ldots, X_n$ and $Y_1, \ldots, Y_n$ into a single natural number that is then stored in the concrete feature s_{2n+1}. We can then devise a new concept $\mathsf{Tile}[i]$ (for each $i \in T$) enforcing that the position identified by the feature s_{2n+1} is labeled with tile i:

$$\begin{aligned}\mathsf{Tile}[i] := {} & \exists r.{=_{\lceil \log(\#T) \rceil}} \sqcap \exists s_{2n+1}, r, r'.{*} \sqcap \exists r', r''.\mathsf{2^x} \\ \sqcap {} & \exists \mathsf{one}.{=_1} \sqcap \exists r, \mathsf{one}, t.{+} \sqcap \exists t, t'.\mathsf{2^x} \\ \sqcap {} & \exists f \circ \mathsf{torus}, r'', u.\mathsf{div} \sqcap \exists u, t'', \mathsf{tile}.\mathsf{mod} \\ \sqcap {} & \exists \mathsf{tile}.{=_i}\end{aligned}$$

Here, r, r', r'', t, t', u, one, torus, and tile are concrete features and $\mathsf{2^x}$ is a binary predicate expressing exponentiation with basis 2. The torus feature is the counterpart of the darr feature in the original reduction, i.e., it stores the natural number that represents the torus. Somewhat more succinctly, the $\mathsf{Tile}[i]$ concept states that, if $N^{\mathcal{I}} = \{w\}$, then all elements e in its extension satisfy the equation

$$(\mathsf{torus}^{\mathcal{I}}(w) \text{ div } 2^{s^{\mathcal{I}}_{2n+1}(e) \cdot \lceil \log(\#T) \rceil}) \mod 2^{\lceil \log(\#T) \rceil} + 1 = i.$$

We can use the $\mathsf{Tile}[i]$ concept in the obvious way inside the $\mathsf{CheckHMatch}$, $\mathsf{CheckVMatch}$, and $\mathsf{Init2}$ concepts. The size of the resulting reduction concept is polynomial in n if the numbers k appearing in $=_k$ predicates are coded in binary. We thus obtain the following theorem:

Theorem 28. *Let $\mathcal{D}$ be a concrete domain such that $\mathbb{N} \subseteq \Delta_\mathcal{D}$ and $\Phi_\mathcal{D}$ contains the following predicates:*

1. *unary $=_k$ with $(=_k)^\mathcal{D} = \{n\}$ for each $k \in \mathbb{N}$ such that the size of (the representation of) $=_k$ is logarithmic in k, and*
2. *ternary $+$ with $(+)^\mathcal{D} \cap \{(k_1, k_2, x) \mid k_1, k_2 \in \mathbb{N}$ and $x \in \Delta_\mathcal{D}\} = \{(k_1, k_2, k_1 + k_2) \mid k_1, k_2 \in \mathbb{N}\}$.*
3. *ternary $*$ with $(*)^\mathcal{D} \cap \{(k_1, k_2, x) \mid k_1, k_2 \in \mathbb{N}$ and $x \in \Delta_\mathcal{D}\} = \{(k_1, k_2, k_1 \cdot k_2) \mid k_1, k_2 \in \mathbb{N}\}$.*
4. *binary $\mathsf{2^x}$ with $(\mathsf{2^x})^\mathcal{D} \cap \{(k, x) \mid k \in \mathbb{N}$ and $x \in \Delta_\mathcal{D}\} = \{(k, 2^k) \mid k \in \mathbb{N}\}$.*
5. *ternary div with $(\mathsf{div})^\mathcal{D} \cap \{(k_1, k_2, x) \mid k_1, k_2 \in \mathbb{N}$ and $x \in \Delta_\mathcal{D}\} = \{(k_1, k_2, k_1 \, div \, k_2) \mid k_1, k_2 \in \mathbb{N}\}$.*
6. *ternary mod with $(\mathsf{mod})^\mathcal{D} \cap \{(k_1, k_2, x) \mid k_1, k_2 \in \mathbb{N}$ and $x \in \Delta_\mathcal{D}\} = \{(k_1, k_2, k_1 \bmod k_2) \mid k_1, k_2 \in \mathbb{N}\}$.*

Then the satisfiability of $\mathcal{ALCO}(\mathcal{D})$-concepts is NExpTime*-hard.*

4 Reasoning Procedures

This section is devoted to developing reasoning procedures for description logics with concrete domains, nominals, and keys. We start with devising a tableau algorithm that decides the satisfiability of $\mathcal{ALCOK}(\mathcal{D})$-concepts w.r.t. Boolean key boxes. This algorithm yields a NExpTime upper complexity bound matching the lower bounds established in Section 3.3. Then we consider the rather powerful description logic $\mathcal{SHOQK}(\mathcal{D})$. This DL, which is an extension of $\mathcal{SHOQ}(\mathcal{D})$ as introduced in [31, 42], provides a wealth of expressive means such as transitive roles, role hierarchies, nominals, and qualifying number restrictions. Moreover, $\mathcal{SHOQK}(\mathcal{D})$ is equipped with a restricted variant of the concrete domain constructor and with key boxes. We develop a tableau algorithm for deciding the satisfiability of $\mathcal{SHOQK}(\mathcal{D})$-concepts w.r.t. key boxes. Due to the restrictedness of $\mathcal{SHOQK}(\mathcal{D})$'s concrete domain constructor, we can even admit general rather than only Boolean key boxes. Again, the algorithm yields a tight NExpTime upper complexity bound.

4.1 A Tableau Algorithm for $\mathcal{ALCOK}(\mathcal{D})$ with Boolean Key Boxes

Tableau algorithms decide the satisfiability of the input concept (in our case w.r.t. the input key box) by attempting to construct a model for it. More precisely, a tableau algorithm starts with an initial data structure induced by the input concept and then repeatedly applies so-called completion rules to it. This rule application can be thought of as attempting to construct a model for the input concept. Finally, either the algorithm will find an obvious contradiction or it will encounter a situation that is contradiction-free and in which no more completions rules are applicable. In the former case, the input concept is unsatisfiable, while it is satisfiable in the latter case.

If the goal is to devise a tableau algorithm for a description logic with concrete domains without committing to a *particular* concrete domain, then an "interface" between the tableau algorithm and a concrete domain reasoner is needed. Usually, it suffices to assume that the concrete domain is admissible, which implies that there exists a procedure that can tell the tableau algorithm whether a given $\mathcal{D}$-conjunction is satisfiable [3, 40, 37]. In the presence of keys, however, this is not enough: we do not only need to know whether a given $\mathcal{D}$-conjunction is satisfiable, but also which variables in it must take the same value in solutions. As an example, consider the concrete domain $\mathsf{N} = (\mathbb{N}, \{<_n \mid n \in \mathbb{N}\})$ and the N-conjunction

$$c = <_2(v_1) \wedge <_2(v_2) \wedge <_2(v_3).$$

Obviously, every solution δ for c satisfies

$$\delta(v_1) = \delta(v_2),\ \delta(v_1) = \delta(v_3),\ \text{or}\ \delta(v_2) = \delta(v_3).$$

This information has to be passed from the concrete domain reasoner to the tableau algorithm since, in the presence of key boxes, it may have an impact on the structure of the constructed model. For example, this information transfer reveals the unsatisfiability of

$$\exists R.A \sqcap \exists R.(\neg A \sqcap B) \sqcap \exists R.(\neg A \sqcap \neg B) \sqcap \forall R.\exists g.<_2 \text{ w.r.t. } (g \text{ keyfor } \top).$$

To formalize this requirement, we strengthen the notion of admissibility into *key-admissibility.* Since the tableau algorithm developed in this section will be non-deterministic, we formulate key-admissibility in a non-deterministic way.

Definition 29 (key-admissible). A concrete domain $\mathcal{D}$ is *key-admissible* iff it satisfies the following properties:

1. $\Phi_\mathcal{D}$ contains a name $\top_\mathcal{D}$ for $\Delta_\mathcal{D}$;
2. $\Phi_\mathcal{D}$ is closed under negation;
3. there exists an algorithm that takes as input a $\mathcal{D}$-conjunction c, returns clash if c is unsatisfiable, and otherwise non-deterministically outputs an equivalence relation $\sim$ on the set of variables V used in c such that there exists a solution δ for c with the following property: for all $v, v' \in V$

$$\delta(v) = \delta(v') \text{ iff } v \sim v'.$$

Equivalence relations as described in Point 3 are henceforth called *concrete equivalences.* We say that *extended $\mathcal{D}$-satisfiability is in NP* if there exists an algorithm as above running in polynomial time. ◊

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

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

Figure 7: The NNF rewrite rules.

Note that this property is much less esoteric than it seems: any concrete domain that is admissible and provides for an equality predicate is also key-admissible. Due to admissibility, the presence of an equality predicate implies that an inequality predicate is also available. We can now construct an algorithm for extended $\mathcal{D}$-satisfiability from an algorithm for $\mathcal{D}$-satisfiability: when presented with a predicate conjunction c, we simply "guess" an equivalence relation $\sim$ on the set of variables used in c. Then we decide the (non-extended) satisfiability of the conjunction $c \wedge \bigwedge_{v \sim v'} =(v,v') \wedge \bigwedge_{v \not\sim v'} \neq(v,v')$, return clash if it is unsatisfiable and $\sim$ otherwise. The rather weak condition that an equality predicate should be present is satisfied by almost all concrete domains proposed in the literature, see e.g. [38, 4, 34, 22, 9].

Throughout this chapter, we assume that any concrete domain is equipped with an equality predicate. This can we done w.l.o.g. since any $\mathcal{D}$-conjunction using equality can be translated into an equivalent one without equality by identifying variables according to the stated equalities. This assumption must not be confused with what was discussed in the previous paragraph: even if the concrete domain $\mathcal{D}$ is admissible and its set of predicates is thus closed under negation, this assumption does *not* imply the presence of an inequality predicate.

We need some more prerequisites before we can start the presentation of the tableau algorithm: a concept is in *negation normal form (NNF)* if negation occurs only in front of concept names and nominals. It is easily seen that, if the concrete domain $\mathcal{D}$ is admissible, then every $\mathcal{ALCOK}(\mathcal{D})$-concept can be converted into an equivalent one in NNF by exhaustively applying the rewrite rules displayed in Figure 7. We use $\dot\neg C$ to denote the result of converting $\neg C$ to NNF. A key box is in NNF if all concepts occurring in key definitions are in NNF. In what follows, we generally assume input concepts and key boxes to be in NNF. Let C be an $\mathcal{ALCOK}(\mathcal{D})$-concept and $\mathcal{K}$ a key box. We use $\mathsf{sub}(C)$ to denote the set of subconcepts of C (including C itself) and $\mathsf{con}(\mathcal{K})$ to denote the set of concepts appearing on the right-hand side of key definitions in $\mathcal{K}$. For a set of concepts Γ, $\mathsf{sub}(\Gamma)$ denotes the set $\bigcup_{C \in \Gamma} \mathsf{sub}(C)$. Moreover, we write $\mathsf{cl}(C, \mathcal{K})$ as abbreviation for the set

$$\mathsf{sub}(C) \cup \mathsf{sub}(\mathsf{con}(\mathcal{K})) \cup \{\dot\neg D \mid D \in \mathsf{sub}(\mathsf{con}(\mathcal{K}))\}.$$

Let us now start the presentation of the tableau algorithm by introducing the underlying data structure.

Definition 30 (Completion System). Let $\mathsf{O_a}$ and $\mathsf{O_c}$ be disjoint and countably infinite sets of *abstract* and *concrete nodes*. A *completion tree* for an $\mathcal{ALCOK}(\mathcal{D})$-

concept C and a key box $\mathcal{K}$ is a finite, labeled tree $\mathbf{T} = (V_a, V_c, E, \mathcal{L})$ with nodes $V_a \cup V_c$ such that $V_a \subseteq \mathsf{O_a}$, $V_c \subseteq \mathsf{O_c}$, and all nodes from V_c are leaves. The tree is labeled as follows:

1. each node $a \in V_a$ is labeled with a subset $\mathcal{L}(a)$ of $\mathsf{cl}(C, \mathcal{K})$;
2. each edge $(a, b) \in E$ with $a, b \in V_a$ is labeled with a role name $\mathcal{L}(a, b)$ occurring in C or $\mathcal{K}$;
3. each edge $(a, x) \in E$ with $a \in V_a$ and $x \in V_c$ is labeled with a concrete feature $\mathcal{L}(a, x)$ occurring in C or $\mathcal{K}$.

For $a \in V_a$, we use $\mathsf{lev}_{\mathbf{T}}(a)$ to denote the depth on which a occurs in $\mathbf{T}$ (starting with the root node on depth 0). A *completion system* for an $\mathcal{ALCOK}(\mathcal{D})$-concept C and a key box $\mathcal{K}$ is a tuple $(\mathbf{T}, \mathcal{P}, \prec, \sim)$, where

- $\mathbf{T} = (V_a, V_c, E, \mathcal{L})$ is a completion tree for C and $\mathcal{K}$,
- $\mathcal{P}$ is a function mapping each $P \in \Phi_{\mathcal{D}}$ of arity n in C to a subset of V_c^n,
- $\prec$ is a linear ordering of V_a such that $\mathsf{lev}_{\mathbf{T}}(a) \leq \mathsf{lev}_{\mathbf{T}}(b)$ implies $a \prec b$, and
- $\sim$ is an equivalence relation on V_c.

Let $(V_a, V_c, E, \mathcal{L})$ be a completion tree. A node $b \in V_a$ is an *R-successor* of a node $a \in V_a$ if $(a, b) \in E$ and $\mathcal{L}(a, b) = R$, while a node $x \in V_c$ is a *g-successor* of a if $(a, x) \in E$ and $\mathcal{L}(a, x) = g$. For a path u the notion *u-successor* is defined in the obvious way. $\diamondsuit$

Intuitively, the relation $\sim$ records equalities between concrete nodes that have been found during the model construction process. The relation $\sim$ induces an equivalence relation $\approx_{\mathsf{a}}$ on abstract nodes which in turn yields another equivalence relation $\approx_{\mathsf{c}} \supseteq \sim$ on concrete nodes.

Definition 31 ($\approx_{\mathsf{a}}$ and $\approx_{\mathsf{c}}$ Relations). Let $S = (\mathbf{T}, \mathcal{P}, \prec, \sim)$ be a completion system for a concept C and a key box $\mathcal{K}$ with $\mathbf{T} = (V_a, V_c, E, \mathcal{L})$, and let $\approx$ be an equivalence relation on V_a. For each $R \in \mathsf{N_R}$, a node $b \in V_a$ is an *$R/{\approx}$-neighbor* of a node $a \in V_a$ if there exists a node $c \in V_a$ such that $a \approx c$ and b is an R-successor of c. Similarly, for each $g \in \mathsf{N_{cF}}$ a node $x \in V_c$ is a *$g/{\approx}$-neighbor* of a if there exists a node $c \in V_a$ such that $a \approx c$ and x is a g-successor of c. For paths u, the notion *$u/{\approx}$-neighbor* is defined in the obvious way.

We define a sequence of equivalence relations $\approx_{\mathsf{a}}^0 \subseteq \approx_{\mathsf{a}}^1 \subseteq \cdots$ on V_a as follows:

$$\begin{array}{rcl}
\approx_{\mathsf{a}}^0 & = & \{(a,a) \in V_a^2\} \cup \\
 & & \{(a,b) \in V_a^2 \mid \text{there is an } N \in \mathsf{N_O} \text{ such that } N \in \mathcal{L}(a) \cap \mathcal{L}(b)\} \\
\approx_{\mathsf{a}}^{i+1} & = & \approx_{\mathsf{a}}^i \cup \\
 & & \{(a,b) \in V_a^2 \mid \text{there is a } c \in \mathsf{V}_a \text{ and an } f \in \mathsf{N_{aF}} \text{ such that} \\
 & & \qquad a \text{ and } b \text{ are } f/{\approx_{\mathsf{a}}^i}\text{-neighbors of } c\} \cup \\
 & & \{(a,b) \in V_a^2 \mid \text{there is a } (u_1, \ldots, u_n \text{ keyfor } D) \in \mathcal{K}, \\
 & & \qquad u_i/{\approx_{\mathsf{a}}^i}\text{-neighbors } x_i \text{ of } a \text{ for } 1 \leq i \leq n, \text{ and} \\
 & & \qquad u_i/{\approx_{\mathsf{a}}^i}\text{-neighbors } y_i \text{ of } b \text{ for } 1 \leq i \leq n \\
 & & \qquad \text{such that } D \in \mathcal{L}(a) \cap \mathcal{L}(b) \text{ and } x_i \sim y_i \text{ for } 1 \leq i \leq n\}
\end{array}$$

Finally, set $\approx_{\mathsf{a}} = \bigcup_{i \geq 0} \approx^i_{\mathsf{a}}$. Then define

$$\approx_{\mathsf{c}} \quad = \quad \sim \cup \{(x,y) \in V_c^2 \mid \text{ there is an } a \in V_a \text{ and a } g \in \mathsf{N_{cF}} \text{ such that } x \text{ and } y \text{ are } g/\approx_{\mathsf{a}}\text{-neighbors of } a\}.$$

$\Diamond$

Let $\mathcal{D}$ be a key-admissible concrete domain. To decide the satisfiability of an $\mathcal{ALCOK}(\mathcal{D})$-concept C_0 w.r.t. a Boolean key box $\mathcal{K}$ (both in NNF), the tableau algorithm is started with the *initial completion tree*

$$\mathbf{T}_{C_0} = (\{a_0\}, \emptyset, \emptyset, \{a_0 \mapsto \{C_0\}\})$$

in the *initial completion system*

$$S_{C_0} = (\mathbf{T}_{C_0}, \mathcal{P}_\emptyset, \emptyset, \{\}),$$

where $\mathcal{P}_\emptyset$ maps each $P \in \Phi_{\mathcal{D}}$ occurring in C_0 to $\emptyset$. We now introduce an operation that is used by the completion rules to add new nodes to completion trees.

Definition 32 ("+" Operation). An abstract or concrete node is called *fresh* in a completion tree $\mathbf{T}$ if it does not appear in $\mathbf{T}$. Let $S = (\mathbf{T}, \mathcal{P}, \prec, \sim)$ be a completion system with $\mathbf{T} = (V_a, V_c, E, \mathcal{L})$. We use the following notions:

- Let $a \in V_a$, $b \in \mathsf{O_a}$ fresh in $\mathbf{T}$, and $R \in \mathsf{N_R}$. We write $S + aRb$ to denote the completion system S' that can be obtained from S by adding b to V_a and (a,b) to E and setting $\mathcal{L}(a,b) = R$ and $\mathcal{L}(b) = \emptyset$. Moreover, b is inserted into $\prec$ such that $b \prec c$ implies $\mathsf{lev}_{\mathbf{T}}(b) \leq \mathsf{lev}_{\mathbf{T}}(c)$.

- Let $a \in V_a$, $x \in \mathsf{O_c}$ fresh in $\mathbf{T}$ and $g \in \mathsf{N_{cF}}$. We write $S + agx$ to denote the completion system S' that can be obtained from S by adding x to V_c and (a,x) to E and setting $\mathcal{L}(a,x) = g$.

When nesting the $+$ operation, we omit brackets writing, e.g., $S + aR_1b + bR_2c$ for $(S + aR_1b) + bR_2c$. Let $u = f_1 \cdots f_n g$ be a path. With $S + aux$, where $a \in V_a$ and $x \in \mathsf{O_c}$ are fresh in $\mathbf{T}$, we denote the completion system S' that can be obtained from S by taking distinct nodes $b_1, \ldots, b_n \in \mathsf{O_a}$ which are fresh in $\mathbf{T}$ and setting

$$S' \quad := \quad S + af_1b_1 + \cdots + b_{n-1}f_nb_n + b_ngx.$$

$\Diamond$

Strictly speaking, the $S + aRb$ operation is non-deterministic since we did not specify how precisely the node b is inserted into $\prec$. However, since this is *don't care* non-determinism, we will view the "+" operation as being deterministic.

The completion rules can be found in Figure 8. Note that the $\mathsf{R}\sqcup$ and Rch rules are non-deterministic, i.e., they have more than one possible outcome (this is true *don't know* non-determinism). Some further remarks on the completion rules are in order: the upper five rules are well-known from existing tableau algorithms for $\mathcal{ALC}(\mathcal{D})$-concept satisfiability (c.f. for example [37]). Only $\mathsf{R}\forall$ deserves a comment since it considers

R$\sqcap$	if $C_1 \sqcap C_2 \in \mathcal{L}(a)$ and $\{C_1, C_2\} \not\subseteq \mathcal{L}(a)$ 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 $\{C_1, C_2\} \cap \mathcal{L}(a) = \emptyset$ 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 there is no $R/{\approx_\mathsf{a}}$-neighbor b of a such that $C \in \mathcal{L}(b)$, then set $S := S + aRb$ for a fresh $b \in \mathsf{O_a}$ and $\mathcal{L}(b) := \{C\}$
R$\forall$	if $\forall R.C \in \mathcal{L}(a)$, b is an $R/{\approx_\mathsf{a}}$-neighbor of a, and $C \notin \mathcal{L}(b)$ then set $\mathcal{L}(b) := \mathcal{L}(b) \cup \{C\}$
R$\exists$c	if $\exists u_1, \dots, u_n.P \in \mathcal{L}(a)$ and there exist no $x_1, \dots, x_n \in V_c$ such that x_i is u_i-successor of a for $1 \le i \le n$ and $(x_1, \dots, x_n) \in \mathcal{P}(P)$ then set $S := (S + au_1x_1 + \cdots + au_nx_n)$ with $x_1, \dots, x_n \in \mathsf{O_c}$ fresh and $\mathcal{P}(P) := \mathcal{P}(P) \cup \{(x_1, \dots, x_n)\}$
Rch	if $(u_1, \dots, u_n$ keyfor $C) \in \mathcal{K}$ and there exist $x_1, \dots, x_n \in V_c$ such that x_i is $u_i/{\approx_\mathsf{a}}$-neighbor of a for $1 \le i \le n$ and $\{C, \dot\neg C\} \cap \mathcal{L}(a) = \emptyset$ then set $\mathcal{L}(a) := \mathcal{L}(a) \cup \{D\}$ for some $D \in \{C, \dot\neg C\}$
Rp	if $\mathcal{L}(b) \not\subseteq \mathcal{L}(a)$ and $a \in V_a$ is minimal w.r.t. $\prec$ such that $a \approx_\mathsf{a} b$ then set $\mathcal{L}(a) := \mathcal{L}(a) \cup \mathcal{L}(b)$

Figure 8: Completion rules for $\mathcal{ALCOK}(\mathcal{D})$.

$R/{\approx_\mathsf{a}}$-neighbors rather than R-successors as usual. Intuitively, if we have $a \approx_\mathsf{a} b$ for two abstract nodes a and b of the completion tree, then a and b describe the same domain element of the constructed model (and similarly for the $\approx_\mathsf{c}$ relation on concrete nodes). Thus if $a \approx_\mathsf{a} b$ and c is an R-successor of a, then c should also be an R-successor of b. However, since we want the completion tree to be a tree, we do not make the latter successorship explicit. To compensate for this, the R$\forall$ rule talks about $R/{\approx_\mathsf{a}}$-neighbors rather than about R-successors.

The lower two rules are necessary for dealing with key boxes. The Rch rule is a so-called "choose rule" (c.f. [26, 33]): intuitively, it guesses whether or not an abstract node a satisfies C if there exists a key definition $(u_1, \dots, u_n$ keyfor $C) \in \mathcal{K}$ such that there are neighbors of a for all the paths u_i. This is necessary since both possibilities may have ramifications: if a satisfies C, then it must be taken into account in the construction of the relation $\approx_\mathsf{a}$; if a does not satisfy C, then we must deal with the consequences of it satisfying $\dot\neg C$ (imagine e.g. that C is $\top$).

The Rp rule is dealing with equalities between abstract nodes as recorded by the $\approx_\mathsf{a}$ relation: since $a \approx_\mathsf{a} b$ means that a and b describe the same node in the constructed model, their node labels should be identical. It suffices, however, to choose one representative for each equivalence class of $\approx_\mathsf{a}$ and make sure that this representative's node label contains the labels of all its $\approx_\mathsf{a}$-equivalent nodes. As representative, we use the node that is minimal w.r.t. the ordering $\prec$, which has been introduced for solely this reason. The Rp rule does the appropriate copying of node labels.

```
define procedure sat($S$)
  do
    if $S$ contains a clash then
      return unsatisfiable
    $\sim$ := check($\zeta_S$)
    compute $\approx_{\mathsf{a}}$
    compute $\approx_{\mathsf{c}}$
  while $\sim \neq \approx_{\mathsf{c}}$
  if $S$ contains a clash then
    return unsatisfiable
  if $S$ is complete then
    return satisfiable
  $S'$ := the application of a completion rule to $S$
  return sat($S'$)
```

Figure 9: The $\mathcal{ALCOK}(\mathcal{D})$ tableau algorithm.

Let us now formalize what it means for a completion system to contain a contradiction.

Definition 33 (Clash). Let $S = (\mathbf{T}, \mathcal{P}, \prec, \sim)$ be a completion system for a concept C and a key box $\mathcal{K}$ with $\mathbf{T} = (V_a, V_c, \prec, \sim)$. We say that the completion system S is *concrete domain satisfiable* iff the conjunction

$$\zeta_S \quad = \quad \bigwedge_{P \text{ used in } C} \; \bigwedge_{(x_1,\ldots,x_n)\in\mathcal{P}(P)} P(x_1,\ldots,x_n) \wedge \bigwedge_{x \approx_{\mathsf{c}} y} =(x,y)$$

is satisfiable. S is said to contain a *clash* iff

1. there is an $a \in V_a$ and an $A \in \mathsf{N_C}$ such that $\{A, \neg A\} \subseteq \mathcal{L}(a)$,
2. there are $a \in V_a$ and $x \in V_c$ such that $g\uparrow \in \mathcal{L}(a)$ and x is $g/\approx_{\mathsf{a}}$-neighbor of a,
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. $\Diamond$

The tableau algorithm is described in Figure 9 in pseudo-code notation. In this figure, check refers to the algorithm computing a concrete equivalence for a given $\mathcal{D}$-conjunction as described in Definition 29. Let us spend a few words on the while loop. There obviously exist close relationships between the relations $\sim$ and $\approx_{\mathsf{c}}$ and the predicate conjunction ζ_S:

- $\sim \subseteq \approx_{\mathsf{c}}$ (note that both $\approx_{\mathsf{a}}$ and $\approx_{\mathsf{c}}$ depend on $\sim$ and are thus recomputed in each step of the while loop);

- the result of $\mathsf{check}(\zeta_S)$ yields a relation containing $\approx_{\mathsf{c}}$ (and thus also $\sim$).

Using these facts, one may check that, in each step of the while loop, new tuples are added to the $\sim$ relation, but none are deleted (see the proof of Lemma 35 below). The presence of the while loop leads to a tight coupling between the concrete domain reasoner and the tableau algorithm: if the concrete domain reasoner finds that two concrete nodes are equal, the tableau algorithm may use this to deduce (via the computation of $\approx_{\mathsf{a}}$ and $\approx_{\mathsf{c}}$) even more equalities between concrete nodes. These new equalities may then be used by the concrete domain reasoner to find additional ones and so forth.

A similar interplay takes place in the course of several recursion steps: equalities between concrete nodes provided by the concrete domain reasoner may make new rules applicable (for example Rp and $\mathsf{R\exists c}$) which changes $\mathcal{P}$ and thus also ζ_S. This may subsequently lead to the detection of more equalities between concrete nodes by the concrete domain reasoner, and so forth. These considerations show that, in the presence of keys, there exists a close interplay between the concrete domain reasoner and the tableau algorithm which is not needed if keys are not present: in this case, it suffices to apply the concrete domain satisfiability check only once after the completion rules have been exhaustively applied [3].

We now prove termination, soundness, and completeness of the tableau algorithm, starting with termination. We first need to establish a few notions and technical lemmas. Let C be a concept and $\mathcal{K}$ a key box. We use $|C|$ to denote the length of C, i.e. the number of symbols used to write it down, and $|\mathcal{K}|$ to denote $\sum_{(u_1,\dots,u_k \text{ keyfor } C)\in\mathcal{K}} |C|$. The *role depth* of concepts is defined inductively as follows:

$$\begin{aligned}
&\mathsf{rd}(A) = \mathsf{rd}(N) = \mathsf{rd}(g\uparrow) = 0\\
&\mathsf{rd}(\exists u_1,\dots,u_n.P) = \max\{|u_i| \mid 1 \le i \le n\} - 1\\
&\mathsf{rd}(C \sqcap D) = \mathsf{rd}(C \sqcup D) = \max\{\mathsf{rd}(C), \mathsf{rd}(D)\}\\
&\mathsf{rd}(\exists R.C) = \mathsf{rd}(\forall R.C) = \mathsf{rd}(C) + 1
\end{aligned}$$

The following series of lemmas will eventually allow us to prove termination.

Lemma 34. *There is a constant k such that, if the tableau algorithm is started on input $C_0, \mathcal{K}$ and $\mathbf{T} = (V_a, V_c, E, \mathcal{L})$ is a completion tree constructed during the run of the algorithm, then $\#V_a \le 2^{|C_0|^k}$ and $\#V_c \le 2^{|C_0|^k}$.*

Proof. Using induction on the number of rule applications and a case distinction according to the applied rule, it is straightforward to show that

$$C \in \mathcal{L}(a) \text{ implies } \mathsf{rd}(C) \le |C_0| - \mathsf{lev}_{\mathbf{T}}(a) \qquad (*)$$

for all constructed completion trees $\mathbf{T}$. We omit the details but note that, (1) for treating the Rch rule, one needs to employ the fact that $\mathcal{K}$ is Boolean and thus only adds concepts of role depth 0 to node labels, and (2) for treating the Rp rule, we use that $a \prec b$ implies $\mathsf{lev}_{\mathbf{T}}(a) \le \mathsf{lev}_{\mathbf{T}}(b)$.

This implies an upper bound on the depth of constructed completion trees: first, only the $\mathsf{R\exists}$ and $\mathsf{R\exists c}$ rules generate new nodes and an application of both rules to a node $a \in V_a$ implies $\mathcal{L}(a) \neq \emptyset$ and thus $\mathsf{lev}_{\mathbf{T}}(a) \leq |C_0|$ by $(*)$. Second, each new (abstract or concrete) node b generated by an application of these rules to a node $a \in V_a$ clearly satisfies $\mathsf{lev}_{\mathbf{T}}(b) \leq \mathsf{lev}_{\mathbf{T}}(a) + \mathsf{max}(1, \mathsf{mpl}(C_0))$, where $\mathsf{mpl}(C_0)$ denotes the maximum length of paths in C_0 (note that concepts in $\mathcal{K}$ may not contain any paths since it is Boolean). Since $\mathsf{mpl}(C_0) \leq |C_0|$, the above observations imply that the depth of constructed completion trees is bounded by $2 \cdot |C_0|$.

Now for the out-degree. If a node a is generated, then this is due to the application of a rule $\mathsf{R}\exists$ or $\mathsf{R}\exists\mathsf{c}$ and initially a has at most one successor. Let us analyze the number of successors generated by later applications of the rules $\mathsf{R}\exists$ and $\mathsf{R}\exists\mathsf{c}$: these rules can be applied at most once for each concept $\exists R.C$ and $\exists u_1, \ldots, u_n.P$ appearing in a node label. By definition of $\mathsf{cl}(C_0, \mathcal{K})$ and since $\mathcal{K}$ is Boolean, the number of such concepts per node label is bounded by $\#\mathsf{sub}(C_0) \leq |C_0|$. Moreover, each rule application creates at most $|C_0|$ successors. Hence, the out-degree of constructed completion trees is bounded by $|C_0|^2 + 1$. ❑

Lemma 35. *There is a constant k such that, if the tableau algorithm is started with $C_0, \mathcal{K}$, then, in every recursion step, the while loop terminates after at most $2^{|C_0|^k}$ steps.*

Proof. Fix an argument $S = (\mathbf{T}, \mathcal{P}, \prec, \sim)$ with $\mathbf{T} = (V_a, V_c, E, \mathcal{L})$ passed to the sat function, let $\sim_1, \sim_2, \ldots$ be the sequence of concrete equivalences computed in the while loop, and let $\approx_{\mathsf{c}}^1, \approx_{\mathsf{c}}^2, \ldots$ be the corresponding $\approx_{\mathsf{c}}$ relations. We show that

$$\sim_1 \subsetneq \sim_2 \subsetneq \cdots, \qquad (*)$$

which implies Lemma 35: by Lemma 34, there exists a constant k such that $\#V_c \leq 2^{|C_0|^k}$. Hence, we have $\#{\sim_0} \leq 2^{2 \cdot |C_0|^k}$ which, together with $(*)$, implies that the number of steps performed by the while loop is also bounded by $2^{2 \cdot |C_0|^k}$.

Now for the proof of $(*)$. If the while loop reaches the i-th step, then we had $\sim_{i-1} \neq \approx_{\mathsf{c}}^{i-1}$ after step $i-1$. Since $\sim_{i-1} \subseteq \approx_{\mathsf{c}}^{i-1}$ by definition, this implies $\sim_{i-1} \subsetneq \approx_{\mathsf{c}}^{i-1}$. By definition of ζ_S, it is easy to see that $\approx_{\mathsf{c}}^{i-1} \subseteq \sim_i$ for $i \geq 0$. Hence $\sim_{i-1} \subsetneq \sim_i$. ❑

Lemma 36. *There is a constant k such that, if the tableau algorithm is started with $C_0, \mathcal{K}$, then the number of recursion calls is bounded by $2^{(|C_0|+|\mathcal{K}|)^k}$.*

Proof. It obviously suffices to establish an appropriate upper bound on the number of rule applications. The $\mathsf{R}\sqcap$, $\mathsf{R}\sqcup$, $\mathsf{R}\exists$, and $\mathsf{R}\exists\mathsf{c}$ rules can be applied at most once for each concept in a node label. By Lemma 34, the number of nodes is at most exponential in $|C_0| + |\mathcal{K}|$. Since neither nodes nor concepts in node labels are ever deleted, the fact that node labels are subsets of $\mathsf{cl}(C_0, \mathcal{K})$ thus implies that the number of applications of these rules is at most exponential in $|C_0| + |\mathcal{K}|$. The same holds for the rules $\mathsf{R}\forall$ and Rp, which can be applied at most once for every concept $C \in \mathsf{cl}(C_0, \mathcal{K})$ and every pair of (abstract) nodes. Finally, the number of Rch applications is at most exponential in $|C_0| + |\mathcal{K}|$ since this rule can be applied at most once for every abstract node and every key definition in $\mathcal{K}$. ❑

Termination is now an obvious consequence of Lemmas 35 and 36.

Corollary 37 (Termination). *The tableau algorithm terminates on any input.*

Let us now prove soundness of the algorithm.

Lemma 38 (Soundness). *If the tableau algorithm returns* satisfiable, *then the input concept C_0 is satisfiable w.r.t. the input key box $\mathcal{K}$.*

Proof. If the tableau algorithm returns satisfiable, then there exists a complete and clash-free completion system $S = (\mathbf{T}, \mathcal{P}, \prec, \sim)$ for C_0. Let $\mathbf{T} = (V_a, V_c, E, \mathcal{L})$. By definition of the tableau algorithm, there is a completion system $S' = (\mathbf{T}, \mathcal{P}, \prec, \sim')$ such that a call to $\mathsf{check}(\zeta_{S'})$ returned $\sim$. Moreover, we have $\sim = \approx_{\mathsf{c}}$ in S. Thus, there exists a solution δ for $\zeta_{S'}$ such that

$$\delta(x) = \delta(y) \text{ iff } x \approx_{\mathsf{c}} y. \tag{†}$$

Clearly, δ is also a solution for ζ_S: since the second component $\mathcal{P}$ of S and S' is identical, δ is a solution for the first part

$$\bigwedge_{P \text{ used in } C} \; \bigwedge_{(x_1,\dots,x_n) \in \mathcal{P}(P)} P(x_1, \dots, x_n)$$

of ζ_S. Moreover, for each conjunct $=(x, y)$ from the second part of ζ_S, we have $x \approx_{\mathsf{c}} y$ by definition of ζ_S and thus $\delta(x) = \delta(y)$ by (†).

We now use S and δ to construct an interpretation $\mathcal{I}$ by setting

$$\begin{array}{rcl}
\Delta_{\mathcal{I}} & = & \{a \in V_a \mid \text{ there is no } b \in V_a \text{ such that } a \approx_{\mathsf{a}} b \text{ and } b \prec a\} \cup \{w\} \\
A^{\mathcal{I}} & = & \{a \in \Delta_{\mathcal{I}} \mid A \in \mathcal{L}(a)\} \\
N^{\mathcal{I}} & = & \left\{ \begin{array}{ll} \{a \in \Delta_{\mathcal{I}} \mid N \in \mathcal{L}(a)\} & \text{if there is an } a \in \Delta_{\mathcal{I}} \text{ such that } N \in \mathcal{L}(a) \\ \{w\} & \text{otherwise} \end{array} \right. \\
R^{\mathcal{I}} & = & \{(a, b) \in \Delta_{\mathcal{I}} \times \Delta_{\mathcal{I}} \mid \text{ there are } a', b' \in V_a \text{ such that } a \approx_{\mathsf{a}} a',\, b \approx_{\mathsf{a}} b', \text{ and} \\
 & & \qquad b' \text{ is } R\text{-successor of } a'\} \\
g^{\mathcal{I}} & = & \{(a, \delta(x)) \in \Delta_{\mathcal{I}} \times \Delta_{\mathcal{D}} \mid x \text{ is } g/\!\approx_{\mathsf{a}}\text{-neighbor of } a\}
\end{array}$$

for all $A \in \mathsf{N_C}$, $N \in \mathsf{N_O}$, $R \in \mathsf{N_R}$, and $g \in \mathsf{N_{cF}}$. We first show that $\mathcal{I}$ is well-defined:

- $N^{\mathcal{I}}$ is a singleton for each $N \in \mathsf{N_O}$. For assume that there exist $a, b \in \Delta_{\mathcal{I}}$ such that $a \neq b$ and $N \in \mathcal{L}(a) \cap \mathcal{L}(b)$. By definition of $\approx_{\mathsf{a}}$, $N \in \mathcal{L}(a) \cap \mathcal{L}(b)$ implies $a \approx_{\mathsf{a}} b$. This, together with $a, b \in \Delta_{\mathcal{I}}$, yields $a \prec b$ and $b \prec a$, a contradiction.

- $f^{\mathcal{I}}$ is functional for each $f \in \mathsf{N_{aF}}$. For assume that there exist $a, b, c \in \Delta_{\mathcal{I}}$ such that $\{(a, b), (a, c)\} \subseteq f^{\mathcal{I}}$ and $b \neq c$. Then there exist $a_1, a_2, b', c' \in V_a$ such that $a \approx_{\mathsf{a}} a_1 \approx_{\mathsf{a}} a_2$, $b \approx_{\mathsf{a}} b'$, $c \approx_{\mathsf{a}} c'$, b' is an f-successor of a_1, and c' is an f-successor of a_2. By definition of $\approx_{\mathsf{a}}$, we thus have $b' \approx_{\mathsf{a}} c'$ implying $b \approx_{\mathsf{a}} c$. Since $b, c \in \Delta_{\mathcal{I}}$, this yields $b \prec c$ and $c \prec b$, a contradiction.

- $g^{\mathcal{I}}$ is functional for each $g \in \mathsf{N_{cF}}$. For assume that there exist an $a \in \Delta_{\mathcal{I}}$ and $x, y \in \mathsf{V}_c$ such that $\{(a, \delta(x)), (a, \delta(y))\} \subseteq f^{\mathcal{I}}$ and $\delta(x) \neq \delta(y)$. Then x and y are both $g/{\approx_{\mathsf{a}}}$-neighbors of a. By definition of $\approx_{\mathsf{c}}$, we thus have $x \approx_{\mathsf{c}} y$ implying $\delta(x) = \delta(y)$ by (†), a contradiction.

The following claim is central for showing that $\mathcal{I}$ is a model for C_0 and $\mathcal{K}$.

Claim: For all $a \in \Delta_{\mathcal{I}}$ and $C \in \mathsf{cl}(C_0, \mathcal{K})$, if $C \in \mathcal{L}(a)$, then $a \in C^{\mathcal{I}}$.

Since C_0 is in the label of the root node, the claim clearly implies that $\mathcal{I}$ is a model for C_0. Moreover, we can use it to prove that $\mathcal{I}$ satisfies all key definitions $(u_1, \dots, u_n \text{ keyfor } C)$ in $\mathcal{K}$: fix $a, b \in C^{\mathcal{I}}$ such that $u_i^{\mathcal{I}}(a) = u_i^{\mathcal{I}}(b)$ for $1 \leq i \leq n$. Non-applicability of Rch yields $\{C, \dot{\neg} C\} \cap \mathcal{L}(a) \neq \emptyset$. If $\dot{\neg} C \in \mathcal{L}(a)$, then the claim implies $a \in (\dot{\neg} C)^{\mathcal{I}}$ in contradiction to $a \in C^{\mathcal{I}}$. Thus we obtain $C \in \mathcal{L}(a)$. In an analogous way, we can argue that $C \in \mathcal{L}(b)$. Using the construction of $\mathcal{I}$ and the fact that $u_i^{\mathcal{I}}(a)$ and $u_i^{\mathcal{I}}(b)$ are defined for $1 \leq i \leq n$, it is readily checked that a has an $u_i/{\approx_{\mathsf{a}}}$-neighbor x_i and b an $u_i/{\approx_{\mathsf{a}}}$-neighbor y_i for $1 \leq i \leq n$. Moreover, the construction of $\mathcal{I}$ and (†) imply that $x_i \approx_{\mathsf{c}} y_i$ and thus $x_i \sim y_i$ for $1 \leq i \leq n$. The above observations obviously imply that $a \approx_{\mathsf{a}} b$. Since $a, b \in \Delta_{\mathcal{I}}$, we obtain $a \not\prec b$ and $b \not\prec a$ by definition of $\Delta_{\mathcal{I}}$ and thus $a = b$.

It remains to prove the above claim, which can be done by structural induction:

- C is a concept name or a nominal. Easy by construction of $\mathcal{I}$.

- $C = \neg D$. Since $C \in \mathsf{cl}(C_0, \mathcal{K})$, C is in NNF and D is a concept name. Since S is clash-free, $C \in \mathcal{L}(a)$ implies $D \notin \mathcal{L}(a)$. Thus, $a \notin D^{\mathcal{I}}$ by construction of $\mathcal{I}$, which yields $a \in (\neg D)^{\mathcal{I}}$.

- $C = \exists u_1, \dots, u_n.P$. Since the $\mathsf{R\exists c}$ rule is not applicable, there exist $x_1, \dots, x_n \in V_c$ such that x_i is a u_i-successor of a for $1 \leq i \leq n$ and $(x_1, \dots, x_n) \in \mathcal{P}(P)$. Using the construction of $\mathcal{I}$ and induction on the length of paths, the reader may check that this implies $u_i^{\mathcal{I}}(a) = \delta(x_i)$ for $1 \leq i \leq n$ (be careful to deal with abstract nodes that are not in $\Delta_{\mathcal{I}}$ but encountered while following paths). Since $(x_1, \dots, x_n) \in \mathcal{P}(P)$ and δ is a solution for ζ_S, we have $(\delta(x_1), \dots, \delta(x_n)) \in P^{\mathcal{D}}$ and thus $a \in C^{\mathcal{I}}$.

- $C = g\uparrow$. Since S is clash-free, there exists no $x \in V_c$ such that x is $g/{\approx_{\mathsf{a}}}$-neighbor of a. Thus, by construction of $\mathcal{I}$, there is no α such that $(a, \alpha) \in g^{\mathcal{I}}$.

- $C = D \sqcap E$ or $C = D \sqcup E$. Straightforward using completeness and the induction hypothesis.

- $C = \exists R.D$. Since the $\mathsf{R\exists}$ rule is not applicable, a has an $R/{\approx_{\mathsf{a}}}$-neighbor b such that $D \in \mathcal{L}(b)$. Let b' be minimal w.r.t. $\prec$ such that $b \approx_{\mathsf{a}} b'$. By definition of $\mathcal{I}$, we have $(a, b') \in R^{\mathcal{I}}$. Non-applicability of the Rp rule yields $D \in \mathcal{L}(b')$. By induction, we get $b' \in D^{\mathcal{I}}$ and thus $a \in C^{\mathcal{I}}$.

- $C = \forall R.D$. Let $(a, b) \in R^{\mathcal{I}}$. By definition of $\mathcal{I}$, this implies that there exist $a', b' \in V_a$ such that a is minimal w.r.t. $\prec$ and $a \approx_{\mathsf{a}} a'$, b is minimal w.r.t. $\prec$ and $b \approx_{\mathsf{a}} b'$, and b' is an R-successor of a'. Since b' is clearly an $R/{\approx_{\mathsf{a}}}$-neighbor of

a, non-applicability of $\mathsf{R}\forall$ yields $D \in \mathcal{L}(b')$, which implies $D \in \mathcal{L}(b)$ due to non-applicability of Rp. By induction, we get $b \in D^{\mathcal{I}}$. Since this holds independently of the choice of b, we obtain $a \in (\forall R.D)^{\mathcal{I}}$.

❑

Lemma 39 (Completeness). *If the input concept C_0 is satisfiable w.r.t. the input key box $\mathcal{K}$, then the tableau algorithm returns* satisfiable.

Proof. Let $\mathcal{I}$ be a model of C_0 and $\mathcal{K}$. We use $\mathcal{I}$ to "guide" the (non-deterministic parts of) the algorithm such that it constructs a complete and clash-free completion system. A completion system $S = (\mathbf{T}, \mathcal{P}, \prec, \sim)$ with $\mathbf{T} = (V_a, V_c, E, \mathcal{L})$ is called *$\mathcal{I}$-compatible* if there exist mappings $\pi : V_a \to \Delta_{\mathcal{I}}$ and $\tau : V_c \to \Delta_{\mathcal{D}}$ such that

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

(Cb) b is an R-successor of $a \Rightarrow (\pi(a), \pi(b)) \in R^{\mathcal{I}}$

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

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

(Ce) $x \sim y \Rightarrow \tau(x) = \tau(y)$

We first establish the following claim:

Claim 1: If a completion system S is $\mathcal{I}$-compatible, then (i) $a \approx_{\mathsf{a}} b$ implies $\pi(a) = \pi(b)$ and (ii) $x \approx_{\mathsf{c}} y$ implies $\tau(x) = \tau(y)$.

Proof: We show by induction on i that $a \approx_{\mathsf{a}}^i b$ implies $\pi(a) = \pi(b)$ (c.f. Definition 31), which yields (i).

- Start. If $a \approx_{\mathsf{a}}^0 b$, then there exists a nominal N such that $N \in \mathcal{L}(a) \cap \mathcal{L}(b)$. By (Ca) we obtain $\pi(a) \in N^{\mathcal{I}}$ and $\pi(b) \in N^{\mathcal{I}}$, which yields $\pi(a) = \pi(b)$ by definition of the semantics.
- Step. For $a \approx_{\mathsf{a}}^i b$, we distinguish three cases:
  1. If $a \approx_{\mathsf{a}}^{i-1} b$, then $\pi(a) = \pi(b)$ by induction.
  2. There is a $c \in V_a$ and an $f \in \mathsf{N_{aF}}$ such that both a and b are $f/\approx_{\mathsf{a}}^{i-1}$-neighbors of c. Hence, there exist $c_1, c_2 \in V_a$ such that $c \approx_{\mathsf{a}}^{i-1} c_1 \approx_{\mathsf{a}}^{i-1} c_2$, a is an f-successor of c_1, and b is an f-successor of c_2. By induction, we have $\pi(c) = \pi(c_1) = \pi(c_2)$. Thus (Cb) yields $\{(\pi(c), \pi(a)), (\pi(c), \pi(b))\} \subseteq f^{\mathcal{I}}$ which implies $\pi(a) = \pi(b)$ by definition of the semantics.
  3. There exist $(u_1, \ldots, u_n$ keyfor $C) \in \mathcal{K}$, $u_i/\approx_{\mathsf{a}}^{i-1}$-neighbors x_i of a and $u_i/\approx_{\mathsf{a}}^{i-1}$-neighbors y_i of b for $1 \leq i \leq n$ such that $C \in \mathcal{L}(a) \cap \mathcal{L}(b)$ and $x_i \sim y_i$ for $1 \leq i \leq n$. (Ca) yields $a, b \in C^{\mathcal{I}}$. Using induction, (Cb), and (Cc), it is straightforward to show that $u_i^{\mathcal{I}}(\pi(a)) = \tau(x_i)$ and $u_i^{\mathcal{I}}(\pi(b)) = \tau(y_i)$ for $1 \leq i \leq n$. By (Ce), this implies $u_i^{\mathcal{I}}(\pi(a)) = u_i^{\mathcal{I}}(\pi(b))$ for $1 \leq i \leq k$. Since $\mathcal{I}$ is a model of the key box $\mathcal{K}$, this yields $\pi(a) = \pi(b)$ by definition of the semantics.

Now for Part (ii) of Claim 1. If $x \approx_{\mathsf{c}} y$, then either $x \sim y$ or there is an $a \in V_a$ and a $g \in \mathsf{N_{cF}}$ such that both x and y are $g/{\approx_{\mathsf{a}}}$-neighbors of a. In the former case, (Ce) yields $\tau(x) = \tau(y)$. In the latter case, Part (i) of the claim and (Cc) yields $\{(\pi(a), \tau(x)), (\pi(a), \tau(y))\} \subseteq g^{\mathcal{I}}$ which implies $\tau(x) = \tau(y)$. This finishes the proof of Claim 1.

We can now show that the completion rules can be applied such that $\mathcal{I}$-compatibility is preserved.

Claim 2: If a completion system S is $\mathcal{I}$-compatible and a rule R is applicable to S, then R can be applied such that an $\mathcal{I}$-compatible completion system S' is obtained.

Proof: Let S be an $\mathcal{I}$-compatible completion system, π and τ be functions satisfying (Ca) to (Ce), and let R be a completion rule applicable to S. We make a case distinction according to the type of R.

R$\sqcap$ The rule is applied to a concept $C_1 \sqcap C_2 \in \mathcal{L}(a)$. By (Ca), $C_1 \sqcap C_2 \in \mathcal{L}(a)$ implies $\pi(a) \in (C_1 \sqcap C_2)^{\mathcal{I}}$ and hence $\pi(a) \in C_1^{\mathcal{I}}$ and $\pi(a) \in C_2^{\mathcal{I}}$. Since the rule adds C_1 and C_2 to $\mathcal{L}(a)$, it yields a completion system that is $\mathcal{I}$-compatible via the same π and τ.

R$\sqcup$ The rule is applied to $C_1 \sqcup C_2 \in \mathcal{L}(a)$. $C_1 \sqcup C_2 \in \mathcal{L}(a)$ implies $\pi(a) \in C_1^{\mathcal{I}}$ or $\pi(a) \in C_2^{\mathcal{I}}$. Since the rule adds either C_1 or C_2 to $\mathcal{L}(a)$, it can be applied such that it yields a completion system that is $\mathcal{I}$-compatible via the same π and τ.

R$\exists$ The rule is applied to a concept $\exists R.C \in \mathcal{L}(a)$, generates a new R-successor b of a and sets $\mathcal{L}(b) = \{C\}$. By (Ca), we have $\pi(a) \in (\exists R.C)^{\mathcal{I}}$ and, hence, there exists a $d \in \Delta_{\mathcal{I}}$ such that $(\pi(a), d) \in R^{\mathcal{I}}$ and $d \in C^{\mathcal{I}}$. Set $\pi' := \pi \cup \{b \mapsto d\}$. It is readily checked that the resulting completion system is $\mathcal{I}$-compatible via π' and τ.

R$\forall$ The rule is applied to a concept $\forall R.C \in \mathcal{L}(a)$ and adds C to the label $\mathcal{L}(b)$ of an existing $R/{\approx_{\mathsf{a}}}$-neighbor b of a. Hence, there exists an a' such that $a \approx_{\mathsf{a}} a'$ and b is R-successor of a'. By Part (i) of Claim 1, we have $\pi(a) = \pi(a')$. Thus, by (Ca) we have $\pi(a') \in (\forall R.C)^{\mathcal{I}}$ while (Cb) yields $((\pi(a'), \pi(b)) \in R^{\mathcal{I}}$. By definition of the semantics, we obtain $\pi(b) \in C^{\mathcal{I}}$ and thus the resulting completion system is $\mathcal{I}$-compatible via π and τ.

R$\exists$c The rule is applied to a concept $\exists u_1, \dots, u_n.P \in \mathcal{L}(a)$ with $u_i = f_1^{(i)} \cdots f_{k_i}^{(i)} g_i$ for $1 \leq i \leq n$. The rule application generates new abstract nodes $b_j^{(i)}$ and x_j for $1 \leq i \leq n$ and $1 \leq j \leq k_i$ such that

- $b_1^{(i)}$ is an $f_1^{(i)}$-successor of a for $1 \leq i \leq n$,
- $b_j^{(i)}$ is an $f_j^{(i)}$-successor 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
- $(x_1, \dots, x_n) \in \mathcal{P}(P)$.

By (Ca), we have $\pi(a) \in (\exists u_1, \dots, u_n.P)^{\mathcal{I}}$. Hence, there exist $d_j^{(i)} \in \Delta_{\mathcal{I}}$ for $1 \leq i \leq n$ and $1 \leq j \leq k_i$ and $\alpha_1, \dots, \alpha_n \in \Delta_{\mathcal{D}}$ such that

– $(\pi(a), d_1^{(i)}) \in (f_1^{(i)})^\mathcal{I}$ for $1 \le i \le n$,

– $(d_{j-1}^{(i)}, d_j^{(i)}) \in (f_j^{(i)})^\mathcal{I}$ for $1 \le i \le n$ and $1 < j \le k_i$,

– $g_i^\mathcal{I}(d_{k_i}^{(i)}) = \alpha_i$ for $1 \le i \le n$, and

– $(\alpha_1, \dots, \alpha_n) \in P^\mathcal{D}$.

Set

$$\pi' := \pi \bigcup_{1 \le i \le n \text{ and } 1 \le j \le k_i} \{b_j^{(i)} \mapsto d_j^{(i)}\} \text{ and } \tau' := \tau \cup \bigcup_{1 \le i \le n} \{x_i \mapsto \alpha_i\}.$$

The resulting completion system is $\mathcal{I}$-compatible via π' and τ'.

Rch The rule is applied to an abstract node a and a key definition $(u_1, \dots, u_n \text{ keyfor } C) \in \mathcal{K}$ and non-deterministically adds either C or $\dot{\neg} C$. By definition of the semantics, $\pi(a) \in C^\mathcal{I}$ or $\pi(a) \in (\dot{\neg} C)^\mathcal{I}$. Hence, Rch can be applied such that the resulting completion system is $\mathcal{I}$-compatible via π and τ.

Rp The rule is applied to a concept $C \in \mathcal{L}(a)$ and adds C to the label $\mathcal{L}(b)$ of a node b with $a \approx_\mathsf{a} b$. By (Ca), we have $\pi(a) \in C^\mathcal{I}$. Since Claim 1 yields $\pi(a) = \pi(b)$, it is clear that the resulting completion system is $\mathcal{I}$-compatible via π and τ.

Finally, we show that $\mathcal{I}$-compatibility implies clash-freeness.

Claim 3: Every $\mathcal{I}$-compatible completion system is clash-free.

Proof: Let $S = (\mathbf{T}, \mathcal{P}, \prec, \sim)$ be an $\mathcal{I}$-compatible completion system. To show that S is clash-free, we make a case distinction:

- Assume that there exists an $a \in V_a$ such that $\{A, \neg A\} \in \mathcal{L}(a)$ for some concept name A. Due to (Ca), we have $\pi(a) \in A^\mathcal{I} \cap (\neg A)^\mathcal{I}$, a contradiction.
- Assume that there are $a \in V_a$ and $x \in V_c$ such that $g\uparrow \in \mathcal{L}(a)$ and x is $g/\approx_\mathsf{a}$-neighbor of a. Then there exists a $b \in V_a$ such that $a \approx_\mathsf{a} b$ and x is g-successor of b. By Claim 1, $a \approx_\mathsf{a} b$ yields $\pi(a) = \pi(b)$. Thus, $g\uparrow \in \mathcal{L}(a)$ and (Ca) give $\pi(b) \in (g\uparrow)^\mathcal{I}$. We obtain a contradiction since (Cc) yields $(\pi(b), \tau(x)) \in g^\mathcal{I}$.
- Using Properties (Cd) and (Ce) and Part (ii) of Claim 1, it is easy to check that τ is a solution for ζ_S. Thus, S is concrete domain satisfiable.

We can now describe the "guidance" of the tableau algorithm by the model $\mathcal{I}$ in detail: we ensure that, at all times, the considered completion systems are $\mathcal{I}$-compatible. This does obviously hold for the initial completion system

$$S_{C_0} = (\mathbf{T}_{C_0}, \mathcal{P}_\emptyset, \emptyset, \emptyset) \text{ with } \mathbf{T}_{C_0} = (\{a_0\}, \emptyset, \emptyset, \{a_0 \mapsto \{C\}\}).$$

We guide the non-deterministic check function such that, when given a predicate conjunction ζ_S with set of variables $V_c \subseteq \mathsf{O_c}$ as input, it returns the relation $\sim$ defined by setting $x \sim y$ iff $\tau(x) = \tau(y)$ for all $x, y \in V$. The relation $\sim$ is a concrete equivalence since τ is a solution for ζ_S (see above). With this guidance (Ce) is obviously satisfied after each call to check, and the other properties are not affected by such a call. According to Claim 2, we can apply the completion rules such that $\mathcal{I}$-compatibility is preserved. By Corollary 37, the algorithm terminates always, hence also when guided in this way. Since, by Claim 3, we will not find a clash, the algorithm returns satisfiable. ❑

The tableau algorithm yields decidability and a tight upper complexity bound for $\mathcal{ALCOK}(\mathcal{D})$-concept satisfiability w.r.t. key boxes.

Theorem 40. *Let $\mathcal{D}$ be a concrete domain that is key-admissible. If extended $\mathcal{D}$-satisfiability is in* NP*, then $\mathcal{ALCOK}(\mathcal{D})$-concept satisfiability w.r.t. Boolean key boxes is in* NExpTime.

Proof. Corollary 37 and Lemmas 38 and 39 yield decidability of $\mathcal{ALCOK}(\mathcal{D})$-concept satisfiability w.r.t. Boolean key boxes. For complexity, Lemma 36 provides an exponential bound on the number of recursion calls. Hence, it remains to show that each single recursion step needs at most exponential time. By Lemma 35, the while loop terminates after at most exponentially many steps. In each such step, we compute the relations $\approx_\mathsf{a}$ and $\approx_\mathsf{c}$, which are needed for constructing the predicate conjunction ζ_S and for checking termination of the while loop. Since, by Lemma 34, there exists an exponential bound on the number of abstract and concrete nodes in the completion system S, this can obviously be done in exponential time. Moreover, Lemma 34 implies that the size of ζ_S is at most exponential. This together with the fact that extended $\mathcal{D}$-satisfiability is in NP implies that the call to check needs at most exponential time. All remaining tasks (checking for clashes, completeness, and rule applicability) can clearly also be performed in exponential time. ❑

We should note that, in the way it is presented here, the algorithm leaves quite some room for optimizations. One possible optimization concerns the "re-use" of f-successors (for abstract features f): for example, when applying the R$\exists$ rule to a concept $\exists f.C \in \mathcal{L}(a)$, where a already has an f-successor b, we could simply add C to $\mathcal{L}(b)$ instead of adding a new f-successor c and recording that $b \approx_\mathsf{a} c$. Another candidate for optimizations is the check function. Recall that this function takes a predicate conjunction c with set of variables V and non-deterministically returns a concrete equivalence, i.e., a relation $\sim$ such that there exists a solution δ for c with $v_i \sim v_j$ iff $\delta(v_i) = \delta(v_j)$ (see Definition 29). It is not hard to devise an $\mathcal{ALC}(\mathcal{D})$-concept that enforces completion systems to have exponentially many concrete nodes by slightly adapting well-known $\mathcal{ALC}$-concepts that enforce models of exponential size [25]. Hence, the size of input conjunctions c to check can be exponential in the size of the input concept. Now note that, even for trivial $\mathcal{D}$-conjunctions

$$c = \top_\mathcal{D}(v_1) \wedge \cdots \wedge \top_\mathcal{D}(v_k)$$

with $(\top_\mathcal{D})^\mathcal{D} = \Delta_\mathcal{D}$, there exist more than $k!$ (i.e. factorial of k) distinct concrete equivalences $\sim$. Thus, the number of possible outcomes of a call to the check function may be *double exponential* in the size of the input concept. Considering the above example, a natural approach to attack this problem is to require check to return only *minimal* concrete equivalences: intuitively, an equivalence is minimal if only those variables are equivalent whose equality is enforced by the conjunction. More precisely, $\sim$ is called *minimal* if there exists no concrete equivalence $\sim'$ such that (i) $x \sim' y$ implies $x \sim y$ and (ii) there are x, y with $x \sim y$ and $x \not\sim' y$. We conjecture that restricting check in this way does not destroy soundness and completeness of the tableau algorithm. However, although this definitely is a worthwhile optimization, it does not help to overcome the existence of double exponentially many outcomes of check in the worst case—at least not for all concrete domains $\mathcal{D}$: consider the concrete domain N from Page 27 and conjunctions of the form

$$c_i = <_i(v_1) \wedge \cdots \wedge <_i(v_{2i}).$$

It is readily checked that, for each $i \geq 1$, the number of minimal concrete equivalences for c_i is exponential in i. Moreover, it is not hard to devise a concept C_i of size logarithmic in i that enforces completion systems S such that $\zeta_S = c_i$. Hence, there are still double exponentially many outcomes of the check function.

In the example just discussed, the exponential branching of check is clearly due to the discreteness of the natural numbers. Indeed, if we use a dense (and infinite!) structure for defining concrete domains, it seems that the restriction to minimal concrete equivalences can have the desired effect, namely that the number of check's possible outcomes becomes polynomial in the size of its input and thus exponential in the size of the input concept. For example, consider the concrete domain Q, which is defined as follows:

- Δ_{Q} is the set $\mathbb{Q}$ of rational numbers;
- Φ_{Q} provides unary predicates $\top_{\mathsf{Q}}$ and its negation $\bot_{\mathsf{Q}}$, unary predicates $=_q$ and $\neq_q$ for each $q \in \mathbb{Q}$, binary comparison predicates $\{<, \leq, =, \neq, \geq, >\}$, a ternary addition predicate $+$, and its negation $\overline{+}$ (all with the obvious semantics).

It is readily checked that Q is key-admissible (note that it provides for a binary equality predicate) and thus falls into our framework. We conjecture that there exists only one minimal concrete equivalence for every Q-predicate conjunction c: intuitively, it seems possible to (inductively) determine a relation $\sim$ on the set of variables V used in c such that (i) $x \sim y$ implies that $\delta(x) = \delta(y)$ for *every* solution δ for c and (ii) there exists a solution δ for c such that $v \not\sim v'$ implies $\delta(v) \neq \delta(v')$. Clearly, $\sim$ is a minimal concrete equivalence. Moreover, due to (i) it is the only one.

4.2 A Tableau Algorithm for $\mathcal{SHOQK}(\mathcal{D})$

Although $\mathcal{ALCOK}(\mathcal{D})$ is a quite powerful DL, it lacks several expressive means that can be found in most state-of-the-art description logic systems such as FaCT and RACER [28, 33, 23]. In this section, we consider the very expressive description logic $\mathcal{SHOQK}(\mathcal{D})$ that provides for concrete domains, key boxes, and nominals, but also for many other means of expressivity such as transitive roles, role hierarchies, qualifying number restrictions, and general TBoxes. Modulo some details, $\mathcal{SHOQK}(\mathcal{D})$ can be viewed as the extension of the DL $\mathcal{SHOQ}(\mathcal{D})$ with key boxes. $\mathcal{SHOQK}(\mathcal{D})$ was proposed in [31] and extended in [42] as a tool for ontology reasoning in the context of the semantic web [12, 6]. One very important feature of $\mathcal{SHOQK}(\mathcal{D})$ are so-called TBoxes, i.e. concept equations of the form $C \doteq D$ that are used as a "background theory" in reasoning. Since it is well-known that combining general TBoxes and the concrete domain constructor easily leads to undecidability [5, 39], $\mathcal{SHOQK}(\mathcal{D})$ only offers a path-free variant of the concrete domain constructor—i.e. only concrete features are admitted inside this constructor rather than paths of arbitrary length. [24, 31] show that this restriction regains decidability. Path-freeness of the concrete domain constructor obviously renders abstract features unnecessary, and thus this syntactic type is not available in $\mathcal{SHOQK}(\mathcal{D})$. Moreover, in this section we restrict ourselves to path-free key boxes.

4.2.1 The Description Logic $\mathcal{SHOQK}(\mathcal{D})$

Let us now define $\mathcal{SHOQK}(\mathcal{D})$ in a formal way, starting with the syntax.

Definition 41 ($\mathcal{SHOQK}(\mathcal{D})$ Syntax). A *role axiom* is either a *role inclusion*, which is of the form $R \sqsubseteq S$ with $R, S \in \mathsf{N_R}$, or a *transitivity axiom* $\mathsf{Trans}(R)$ where $R \in \mathsf{N_R}$. A *role box* $\mathcal{R}$ is a finite set of role axioms. A role name R is called *simple* if, for $\sqsubseteq^*$ the reflexive-transitive closure of the role inclusions in $\mathcal{R}$, $S \sqsubseteq^* R$ implies $\mathsf{Trans}(S) \notin \mathcal{R}$ for all role names S. Let $\mathcal{D}$ be a concrete domain. The set of $\mathcal{SHOQK}(\mathcal{D})$-concepts is the smallest set such that

- every concept name and every nominal is a concept, and
- if C and D are concepts, R is a role name, S a simple role name, n and k are natural numbers, $g_1, \ldots, g_n$ are concrete feature, and $P \in \Phi_\mathcal{D}$ is a predicate of arity n, then the following expressions are also concepts:

  $\neg C$, $C \sqcap D$, $C \sqcup D$, $\exists R.C$, $\forall R.C$, $(\geqslant k\ S\ C)$, $(\leqslant k\ S\ C)$, $\exists g_1, \ldots, g_n.P$, and $g_1\uparrow$.

A *concept equation* is an expression $C \doteq D$ with C and D concepts. A *TBox* is a finite set of concept equations. $\Diamond$

For $\mathcal{SHOQK}(\mathcal{D})$, we consider key boxes that differ in two aspect from the ones we considered for $\mathcal{ALCOK}(\mathcal{D})$: in the following, we assume key boxes to be path-free, but we admit complex concepts to occur in key definitions. Note that abstract features and paths do no occur in the syntax of $\mathcal{SHOQK}(\mathcal{D})$—as will become clear after the semantics has been defined, the former can be "simulated" by the more general number restrictions $(\leqslant n\ R\ C)$. As usual in description logics of the $\mathcal{SHIQ}/\mathcal{SHOQ}$ family, we require role names in number restrictions to be simple since admitting arbitrary roles yields undecidability of reasoning [33, 31]. If the role box $\mathcal{R}$ is clear from the context, we will usually write $\mathsf{Trans}(R)$ instead of $\mathsf{Trans}(R) \in \mathcal{R}$. We now introduce the semantics of $\mathcal{SHOQK}(\mathcal{D})$ and the relevant reasoning problems.

Definition 42 ($\mathcal{SHOQK}(\mathcal{D})$ Semantics). *Interpretations* $\mathcal{I} = (\Delta_\mathcal{I}, \cdot^\mathcal{I})$ are defined as in Definition 3. The interpretation function $\cdot^\mathcal{I}$ is extended to $\mathcal{SHOQK}(\mathcal{D})$-concepts as follows:

$$\begin{aligned}
(\neg C)^{\mathcal{I}} &:= \Delta_{\mathcal{I}} \setminus C^{\mathcal{I}} \\
(C \sqcap D)^{\mathcal{I}} &:= C^{\mathcal{I}} \cap D^{\mathcal{I}} \\
(C \sqcup D)^{\mathcal{I}} &:= C^{\mathcal{I}} \cup D^{\mathcal{I}} \\
(\exists R.C)^{\mathcal{I}} &:= \{d \in \Delta_{\mathcal{I}} \mid \{e \mid (d,e) \in R^{\mathcal{I}}\} \cap C^{\mathcal{I}} \neq \emptyset\} \\
(\forall R.C)^{\mathcal{I}} &:= \{d \in \Delta_{\mathcal{I}} \mid \{e \mid (d,e) \in R^{\mathcal{I}}\} \subseteq C^{\mathcal{I}}\} \\
(\leqslant k\ R\ C)^{\mathcal{I}} &:= \{d \in \Delta_{\mathcal{I}} \mid \sharp\{e \mid (d,e) \in R^{\mathcal{I}}\} \leq k\} \\
(\geqslant k\ R\ C)^{\mathcal{I}} &:= \{d \in \Delta_{\mathcal{I}} \mid \sharp\{e \mid (d,e) \in R^{\mathcal{I}}\} \geq k\} \\
(\exists g_1, \ldots, g_n.P)^{\mathcal{I}} &:= \{d \in \Delta_{\mathcal{I}} \mid \exists x_1, \ldots, x_n \in \Delta_{\mathcal{D}} : g_i^{\mathcal{I}}(d) = x_i \text{ and } (x_1, \ldots, x_n) \in P^{\mathcal{D}}\} \\
(g\uparrow)^{\mathcal{I}} &:= \{d \in \Delta_{\mathcal{I}} \mid g^{\mathcal{I}}(d) \text{ undefined}\}.
\end{aligned}$$

Let $\mathcal{I}$ be an interpretation. Then $\mathcal{I}$ *satisfies* a concept equation $C \doteq D$ if $C^{\mathcal{I}} = D^{\mathcal{I}}$. $\mathcal{I}$ is a *model* of a TBox $\mathcal{T}$ if $\mathcal{I}$ satisfies all concept equations in $\mathcal{T}$. Similarly, $\mathcal{I}$ satisfies a role inclusion $R \sqsubseteq S$ if $R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$ and a transitivity axiom $\mathsf{Trans}(R)$ if $R^{\mathcal{I}}$ is a transitive relation. $\mathcal{I}$ is a *model* of a role box $\mathcal{R}$ if $\mathcal{I}$ satisfies all role inclusions and transitivity axioms in $\mathcal{R}$.

Let $\mathcal{T}$ be a TBox, $\mathcal{R}$ a role box, and $\mathcal{K}$ a key box. A concept C is *satisfiable w.r.t. $\mathcal{T}$, $\mathcal{R}$, and $\mathcal{K}$* iff C, $\mathcal{T}$, $\mathcal{R}$, and $\mathcal{K}$ have a common model. C is *subsumed by* a concept D *w.r.t. $\mathcal{T}$, $\mathcal{R}$, and $\mathcal{K}$* (written $C \sqsubseteq_{\mathcal{T},\mathcal{R},\mathcal{K}} D$) iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for all common models $\mathcal{I}$ of $\mathcal{T}$, $\mathcal{R}$, and $\mathcal{K}$. ♢

Note that, due to the requirement that role names used inside number restrictions should be simple, existential and universal value restrictions are no syntactic sugar: in contrast to number restrictions, they can be used on *all* roles.

It is well-known that, in many expressive description logics, reasoning *with* TBoxes can be reduced to reasoning without them [44, 31]: in $\mathcal{SHOQK}(\mathcal{D})$, deciding satisfiability of a concept C w.r.t. $\mathcal{T}$, $\mathcal{R}$, and $\mathcal{K}$ is equivalent to deciding satisfiability of the concept

$$C \sqcap \forall R.\Big(\bigsqcap_{D \doteq E \in \mathcal{T}} D \leftrightarrow E\Big) \sqcap \bigsqcap_{\text{nominal } N \text{ used in } C,\, \mathcal{T},\text{ or } \mathcal{K}} \Big(\exists R.(N \sqcap \bigsqcap_{D \doteq E \in \mathcal{T}} D \leftrightarrow E)\Big)$$

w.r.t. $\mathcal{R}'$, $\mathcal{K}$, and the empty TBox, where R is a fresh role not appearing in C, $\mathcal{R}$, and $\mathcal{T}$, and

$$\mathcal{R}' := \mathcal{R} \cup \{\mathsf{Trans}(R)\} \cup \bigcup_{\text{role name } S \text{ used in } C,\, \mathcal{T},\, \mathcal{R},\text{ or } \mathcal{K}} \{S \sqsubseteq R\}.$$

Since subsumption can be reduced to satisfiability as described in Section 2, in the following we will only consider the satisfiability of concepts w.r.t. role boxes and key boxes, but without TBoxes. We will also generally assume role boxes $\mathcal{R}$ to be *acyclic*, i.e. to satisfy the following condition: for each role name R, there are no role names $R_1, \ldots, R_k$ such that $R = R_1 = R_k$ and $R_i \sqsubseteq R_{i+1} \in \mathcal{R}$ for $1 \leq i < k$. It is not hard to see that this is no restriction since cycles can be eliminated: if $R_1, \ldots, R_k$ is a cycle in $\mathcal{R}$, then we have $R_1^{\mathcal{I}} = \cdots = R_k^{\mathcal{I}}$ for all interpretations $\mathcal{I}$. Thus we can simply remove the cycle from $\mathcal{R}$ and replace every occurrence of $R_2, \ldots, R_k$ in C, $\mathcal{R}$, and $\mathcal{K}$ with R_1. Moreover, we have to possibly add $\mathsf{Trans}(R_1)$ if, before the cycle elimination, we had $\mathsf{Trans}(R_i)$ for some i with $1 \leq i \leq n$.

Before we turn our attention towards the construction of a tableau algorithm for $\mathcal{SHOQK}(\mathcal{D})$, let us comment on a few minor differences between $\mathcal{SHOQK}(\mathcal{D})$ as introduced here and the original version of $\mathcal{SHOQ}(\mathcal{D})$ as described in [31]. The main difference is that our logic, like the extension investigated in [42], allows n-ary predicates while Horrocks and Sattler restrict themselves to unary predicates. Moreover, $\mathcal{SHOQ}(\mathcal{D})$ as introduced in [31] uses *concrete roles* rather than concrete features, the difference being that concrete roles are not necessary functional. Due to this non-functionality, the original $\mathcal{SHOQ}(\mathcal{D})$ admits two variants $\exists T.P$ and $\forall T.P$ of the concrete domain constructor (where T is a concrete role and P a unary predicate). In $\mathcal{SHOQK}(\mathcal{D})$, we can simulate the universal variant by writing $\exists g.P \sqcup g\uparrow$ since concrete features g are interpreted as partial functions and, in contrast to Horrocks and Sattler, we have the undefinedness constructor $g\uparrow$ available. Except for the n-ary predicates which provide important additional expressivity, we view these deviations as minor ones since they are easily seen to not affect decidability and complexity of reasoning.

4.2.2 A Tableau for $\mathcal{SHOQK}(\mathcal{D})$

Similar to the tableau algorithm for $\mathcal{ALCOK}(\mathcal{D})$ concept satisfiability, the $\mathcal{SHOQK}(\mathcal{D})$ algorithm will use completion systems based on completion trees as the underlying data structure. However, to simplify dealing with transitive roles and role hierarchies, in the correctness proofs, we will not establish a direct correspondence between the existence of complete and clash-free completion systems and the existence of models, but rather employ an intermediate step involving an abstraction of models called *tableau*. Intuitively, the main difference between completion systems and tableaux is that completion systems constructed by the tableau algorithm are *finite* objects while a tableau is potentially infinite. The main difference between tableaux and interpretations is that, in tableaux, roles declared to be transitive must not necessarily be described by transitive relations.

In this section, we introduce tableaux. Let us start with discussing some preliminaries. As for $\mathcal{ALCOK}(\mathcal{D})$, we assume all concepts and key boxes to be in *negation normal form* (NNF) and use $\dot{\neg} C$ to denote the NNF of $\neg C$. The NNF rewrite rules for $\mathcal{SHOQK}(\mathcal{D})$ can be found in Figure 10. For a concept D, role box $\mathcal{R}$, and key box $\mathcal{K}$, we define

$$\begin{aligned}\mathsf{cl}(D,\mathcal{K}) &:= \mathsf{sub}(D) \cup \mathsf{sub}(\mathsf{con}(\mathcal{K})) \cup \{\dot{\neg} C \mid C \in \mathsf{sub}(D) \cup \mathsf{sub}(\mathsf{con}(\mathcal{K}))\}\\ \mathsf{cl}(D,\mathcal{R},\mathcal{K}) &:= \mathsf{cl}(D,\mathcal{K}) \cup \{\forall R.C \mid R \mathrel{\underline{\sqsubseteq}^*} S \text{ and } \forall S.C \in \mathsf{cl}(D,\mathcal{K})\},\end{aligned}$$

where $\underline{\sqsubseteq}^*$ denotes the reflexive transitive closure of the role inclusions in $\mathcal{R}$. Obviously, the cardinality of $\mathsf{cl}(D,\mathcal{R},\mathcal{K})$ is linear in the size of D, $\mathcal{R}$, and $\mathcal{K}$. In what follows, we write $\mathsf{N}_\mathsf{R}^{D,\mathcal{R},\mathcal{K}}$ to denote the set of role names occurring in D, $\mathcal{R}$, or $\mathcal{K}$, and $\mathsf{N}_{\mathsf{cF}}^{D,\mathcal{K}}$ to denote the sets of concrete features occurring in D or $\mathcal{K}$. We are now ready to define tableaux.

$$\begin{array}{rcl@{\quad}rcl}
\neg(C \sqcap D) & \rightsquigarrow & \neg C \sqcup \neg D & \neg(C \sqcup D) & \rightsquigarrow & \neg C \sqcap \neg D \\
\neg(\exists R.C) & \rightsquigarrow & \forall R.\neg C & \neg(\forall R.C) & \rightsquigarrow & \exists R.\neg C \\
\neg\neg C & \rightsquigarrow & C & \neg(g\uparrow) & \rightsquigarrow & \exists g.\top_{\mathcal{D}}
\end{array}$$

$$\begin{array}{rcl}
\neg(\geqslant n\ R\ C) & \rightsquigarrow & (\leqslant (n-1)\ R\ C) \text{ if } n \geq 1 \\
\neg(\geqslant 0\ R\ C) & \rightsquigarrow & \bot \\
\neg(\leqslant n\ R\ C) & \rightsquigarrow & (\geqslant (n+1)\ R\ C) \\
\neg(\exists g_1, \ldots, g_n.P) & \rightsquigarrow & \exists g_1, \ldots, g_n.\overline{P} \sqcup g_1\uparrow \sqcup \cdots \sqcup g_n\uparrow
\end{array}$$

Figure 10: The $\mathcal{SHOQK}(\mathcal{D})$ NNF rewrite rules.

Definition 43 (Tableau). Let D be a $\mathcal{SHOQK}(\mathcal{D})$-concept in NNF, $\mathcal{R}$ a role box, and $\mathcal{K}$ a path-free key box in NNF. A *tableau* T for D w.r.t. $\mathcal{R}$ and $\mathcal{K}$ is a tuple $(\mathbf{S}_a, \mathbf{S}_c, \mathcal{L}, E, e, \mathcal{P})$ such that

- $\mathbf{S}_a$, $\mathbf{S}_c$ are sets of *abstract* and *concrete individuals*,
- $\mathcal{L} : \mathbf{S}_a \to 2^{\mathsf{cl}(D,\mathcal{R},\mathcal{K})}$ maps each abstract individual to a subset of $\mathsf{cl}(D,\mathcal{R},\mathcal{K})$,
- $E : \mathbf{S}_a \times \mathbf{S}_a \to 2^{\mathsf{N}_\mathsf{R}^{D,\mathcal{R},\mathcal{K}}}$ maps pairs of abstract individuals to sets of roles,
- $e : \mathbf{S}_a \times \mathsf{N}_{\mathsf{cF}}^{D,\mathcal{K}} \to S_c$ maps pairs of abstract individuals and concrete features to concrete individuals,
- $\mathcal{P}$ maps each n-ary concrete predicate occurring in $\mathsf{cl}(D,\mathcal{R},\mathcal{K})$ to a set of n-tuples over $\mathbf{S}_c$,
- there is an abstract individual $s_0 \in \mathbf{S}_a$ such that $D \in \mathcal{L}(s_0)$, and

for all $s, t \in \mathbf{S}_a$, $C, C_1, C_2 \in \mathsf{cl}(D,\mathcal{R},\mathcal{K})$, $R, S \in \mathsf{N}_\mathsf{R}^{D,\mathcal{R},\mathcal{K}}$, and

$$S^T(s, C) := \{t \in \mathbf{S}_a \mid S \in E(s,t) \text{ and } C \in \mathcal{L}(t)\},$$

it holds that:

(T1) if $C \in \mathcal{L}(s)$, then $\neg C \notin \mathcal{L}(s)$,

(T2) if $C_1 \sqcap C_2 \in \mathcal{L}(s)$, then $C_1 \in \mathcal{L}(s)$ and $C_2 \in \mathcal{L}(s)$,

(T3) if $C_1 \sqcup C_2 \in \mathcal{L}(s)$, then $C_1 \in \mathcal{L}(s)$ or $C_2 \in \mathcal{L}(s)$,

(T4) if $R \in E(s,t)$ and $R \sqsubseteq^* S$, then $S \in E(s,t)$,

(T5) if $\forall R.C \in \mathcal{L}(s)$ and $R \in E(s,t)$, then $C \in \mathcal{L}(t)$,

(T6) if $\exists R.C \in \mathcal{L}(s)$, then there is some $t \in \mathbf{S}_a$ such that $R \in E(s,t)$ and $C \in \mathcal{L}(t)$,

(T7) if $\forall S.C \in \mathcal{L}(s)$ and $R \in E(s,t)$ for some $R \sqsubseteq^* S$ with $\mathsf{Trans}(R)$, then $\forall R.C \in \mathcal{L}(t)$,

(T8) if $(\geqslant n\ S\ C) \in \mathcal{L}(s)$, then $\sharp S^T(s,C) \geqslant n$,

(T9) if $(\leqslant n\ S\ C) \in \mathcal{L}(s)$, then $\sharp S^T(s, C) \leqslant n$,

(T10) if either $(\leqslant n\ S\ C) \in \mathcal{L}(s)$ and $S \in E(s,t)$ or $(g_1, \ldots, g_n$ keyfor $C) \in \mathcal{K}$ and $e(t, g_i)$ is defined for all $1 \leq i \leq n$, then $\{C, \dot{\neg} C\} \cap \mathcal{L}(t) \neq \emptyset$,

(T11) if $N \in \mathcal{L}(s) \cap \mathcal{L}(t)$, then $s = t$,

(T12) if $\exists g_1, \ldots, g_n.P \in \mathcal{L}(s)$, then there are $x_1, \ldots, x_n \in \mathbf{S}_c$ with $e(s, g_i) = x_i$ and $(x_1, \ldots, x_n) \in \mathcal{P}(P)$,

(T13) $\bigwedge_{P \text{ used in } D,\mathcal{K}} \bigwedge_{(x_1,\ldots,x_n) \in \mathcal{P}(P)} P(x_1, \ldots, x_n) \wedge \bigwedge_{x \neq y} x \neq y$ is satisfiable,

(T14) if $(g_1, \ldots, g_n$ keyfor $C) \in \mathcal{K}$, $C \in \mathcal{L}(s) \cap \mathcal{L}(t)$, and $e(s, g_i) = e(t, g_i)$ for all $1 \leq i \leq n$, then $s = t$,

(T15) if $g\uparrow \in \mathcal{L}(s)$, then $e(s, g)$ is undefined.

$\Diamond$

Note that the predicate conjunction in **(T13)** uses a binary inequality predicate. In general, we do not require the concrete domain $\mathcal{D}$ to be equipped with such a predicate and thus this predicate conjunction is not necessarily a $\mathcal{D}$-conjunction. However, it is nevertheless "safe" to use **(T13)** in the given form since tableaux are only used in proofs and we do not need a concrete domain reasoner that is capable of deciding the satisfiability of the listed predicate conjunction. We now show that tableaux are an adequate abstraction of models.

Lemma 44. *Let D be a $\mathcal{SHOQK}(\mathcal{D})$-concept in NNF, $\mathcal{R}$ a role box, and $\mathcal{K}$ a key box in NNF. Then D is satisfiable w.r.t. $\mathcal{R}$ and $\mathcal{K}$ iff D has a tableau w.r.t. $\mathcal{R}$ and $\mathcal{K}$.*

Proof. We concentrate on Properties **(T10)** to **(T15)** since **(T1)** to **(T9)** are "standard" and can also be found in tableaux for $\mathcal{SHIQ}$ and $\mathcal{SHOQ}$, see [33, 31]. For the "only-if" direction, we construct a tableau T from a common model $\mathcal{I}$ of D, $\mathcal{R}$, and $\mathcal{K}$ as follows:

$$\begin{array}{rcl}
\mathbf{S}_a & := & \Delta_{\mathcal{I}} \\
\mathbf{S}_c & := & \{x \in \Delta_{\mathcal{D}} \mid g^{\mathcal{I}}(s) = x \text{ for some } s \in \mathbf{S}_a\} \\
\mathcal{L}(s) & := & \{C \in \mathsf{cl}(D, \mathcal{R}, \mathcal{K}) \mid s \in C^{\mathcal{I}}\} \\
E(s,t) & := & \{S \in \mathsf{N}_{\mathsf{R}}^{D,\mathcal{R},\mathcal{K}} \mid (s,t) \in S^{\mathcal{I}}\} \\
e(s,g) & := & g^{\mathcal{I}}(s) \text{ if } g^{\mathcal{I}}(s) \text{ is defined} \\
\mathcal{P}(P) & := & \{(x_1, \ldots, x_n) \in \mathbf{S}_c^n \mid (x_1, \ldots, x_n) \in P^{\mathcal{D}}\}.
\end{array}$$

It can be easily verified that T is a tableau for D w.r.t. $\mathcal{R}$ and $\mathcal{K}$: the proof that T satisfies **(T1)** – **(T9)** is identical to the corresponding cases in [33, 31]; **(T10)** holds by definition of $\mathcal{L}$; **(T11)** by definition of $\mathcal{L}$ and the fact that nominals are interpreted as singleton sets; **(T12)** by definition of $\mathcal{L}$, e, and $\mathcal{P}$ together with the semantics of concepts $\exists g_1, \ldots, g_n.P$; **(T13)** since the identity function on $\mathbf{S}_c$ is clearly a solution for the listed predicate conjunction; **(T14)** by definition of $\mathcal{L}$ and e together with the semantics of key constraints; and finally **(T15)** by definition of $\mathcal{L}$ and e together with the semantics of concepts $g\uparrow$.

For the "if" direction, let $T = (\mathbf{S}_a, \mathbf{S}_c, \mathcal{L}, E, e, \mathcal{P})$ be a tableau for D w.r.t. $\mathcal{R}$ and $\mathcal{K}$ and let δ be a solution for the predicate conjunction in **(T13)**. We construct a model $\mathcal{I}$ for D as follows:

$$\begin{aligned}
\Delta_{\mathcal{I}} &:= \mathbf{S}_a \\
A^{\mathcal{I}} &:= \{s \in \Delta_{\mathcal{I}} \mid A \in \mathcal{L}(s)\} \\
N^{\mathcal{I}} &:= \{s \in \Delta_{\mathcal{I}} \mid N \in \mathcal{L}(s)\} \\
R^{\mathcal{I}} &:= \begin{cases} \bigcup_{\substack{S \,\underline{\circledast}\, R \\ S \neq R}} S^{\mathcal{I}} \cup \{(s,t) \mid R \in E(s,t)\} & \text{for } R \in \mathsf{N_R} \setminus \mathsf{N_{cF}} \text{ with } \textit{not } \mathsf{Trans}(\mathrm{R}) \\ \{(s,t) \mid R \in E(s,t)\}^{+} & \text{for } R \in \mathsf{N_R} \setminus \mathsf{N_{cF}} \text{ with } \mathsf{Trans}(\mathrm{R}) \end{cases} \\
g^{\mathcal{I}}(s) &:= \begin{cases} \delta(x) & \text{if } e(s,g) = x \\ \text{undefined} & \text{if } e(s,g) \text{ is undefined} \end{cases} \quad \text{for } g \in \mathsf{N_{cF}}.
\end{aligned}$$

Due to **(T11)**, the interpretation of nominals is a singleton. Moreover, the interpretation of roles is well-defined since role boxes are acyclic. The following claim is central for proving that $\mathcal{I}$ is indeed a model for C, $\mathcal{R}$, and $\mathcal{K}$:

Claim: For each $D \in \mathsf{cl}(D, \mathcal{R}, \mathcal{K})$, $D \in \mathcal{L}(s)$ implies $s \in D^{\mathcal{I}}$.

Proof: We proceed by induction on the structure of D. For concept names A and nominals N, the claim follows by definition of $A^{\mathcal{I}}$ and $N^{\mathcal{I}}$. For the negation of concept names A and nominals N (note that D is in NNF), we may use the definition of $A^{\mathcal{I}}$ and $N^{\mathcal{I}}$ together with **(T1)**. Concepts D of the form $C_1 \sqcap C_2$ and $C_1 \sqcup C_2$ can be treated using **(T2)** and **(T3)** together with the induction hypothesis. For existential, universal, and number restrictions, the proof is analogous to the one for $\mathcal{SHIQ}$ in [33]. For concepts of the form $D = \exists g_1, \ldots g_n.P \in \mathcal{L}(s)$, $s \in D^{\mathcal{I}}$ is an immediate consequence of **(T12)**, the definition of $g_i^{\mathcal{I}}$, and the fact that $(x_1, \ldots, x_n) \in \mathcal{P}(P)$ implies $(\delta(x_1), \ldots, \delta(x_n)) \in P^{\mathcal{D}}$ by **(T13)**. Finally, for concepts $D = g\uparrow$, $s \in D^{\mathcal{I}}$ is an immediate consequence of the definition of $g^{\mathcal{I}}$ together with **(T15)**. This finishes the proof of the claim.

By definition of tableaux, there exists an $s_0 \in \mathbf{S}_a$ such that $C \in \mathcal{L}(s_0)$. By the claim, $s_0 \in C^{\mathcal{I}}$ and thus $\mathcal{I}$ is a model of C.

Next, we show that $\mathcal{I}$ is a model of $\mathcal{R}$. By definition of $R^{\mathcal{I}}$, it is obvious that $\mathsf{Trans}(R) \in \mathcal{R}$ implies that $R^{\mathcal{I}}$ is a transitive relation. Now let $S \sqsubseteq R \in \mathcal{R}$. If $\mathsf{Trans}(R) \notin \mathcal{R}$, then we have $S^{\mathcal{I}} \subseteq R^{\mathcal{I}}$ by definition of $R^{\mathcal{I}}$. Now let $\mathsf{Trans}(R) \in \mathcal{R}$ and $(s,t) \in S^{\mathcal{I}}$. If $S \in E(s,t)$, then **(T4)** implies $R \in E(s,t)$, and thus $(s,t) \in E^{\mathcal{I}}$. Otherwise, there is an $S' \,\underline{\circledast}\, S$ with $\mathsf{Trans}(S') \in \mathcal{R}$ and $(s,t) \in \{(u,v) \mid S' \in E(u,v)\}^{+}$. Now **(T4)** together with $S' \,\underline{\circledast}\, R$ implies that $\{(u,v) \mid S' \in E(u,v)\} \subseteq \{(u,v) \mid R \in E(u,v)\}$, and thus $\mathsf{Trans}(R) \in \mathcal{R}$ implies that $(s,t) \in R^{\mathcal{I}}$.

It remains to show that $\mathcal{I}$ is a model of $\mathcal{K}$. To this end, let $(g_1, \ldots, g_n \text{ keyfor } D) \in \mathcal{K}$ and $s,t \in D^{\mathcal{I}}$ such that $g_i^{\mathcal{I}}(s) = g_i^{\mathcal{I}}(t)$ for $1 \leq i \leq n$. Since the predicate conjunction in **(T13)** contains explicit inequalities for all distinct concrete individuals, this implies that $e(s,g_i) = e(t,g_i)$ for $1 \leq i \leq n$. **(T10)** implies $\{D, \dot{\neg} D\} \cap \mathcal{L}(s) \neq \emptyset$ and $\{D, \dot{\neg} D\} \cap \mathcal{L}(t) \neq \emptyset$. If $\dot{\neg} D \in \mathcal{L}(s)$, then the claim yields $s \in (\dot{\neg} D)^{\mathcal{I}}$ contradicting $s \in D^{\mathcal{I}}$. Thus we obtain $D \in \mathcal{L}(s)$, and, in a similar way, $D \in \mathcal{L}(t)$. Finally, **(T14)** implies that $s = t$, and thus $\mathcal{I}$ satisfies $\mathcal{K}$. ❏

4.2.3 A Tableau Algorithm for $\mathcal{SHOQK}(\mathcal{D})$

Lemma 44 shows that, in order to decide satisfiability of $\mathcal{SHOQK}(\mathcal{D})$-concepts w.r.t. role and key boxes, we may use a (tableau) algorithm that tries to construct a tableau for the input. In the following, we will describe such an algorithm in detail. As in the previous section, the algorithm works on completion systems. However, in the case of $\mathcal{SHOQK}(\mathcal{D})$, the core component of completion systems is a completion forest rather than a completion tree. The reason for this is that some completion rules remove nodes from the completion system and in this way can disconnect one tree into two subtrees.

Definition 45 (Completion System). Let D be a $\mathcal{SHOQK}(\mathcal{D})$-concept in NNF, $\mathcal{R}$ a role box, and $\mathcal{K}$ a path-free key box in NNF. For each concept $(\geqslant n\ R\ C) \in \mathsf{cl}(D,\mathcal{R},\mathcal{K})$ and $1 \leq i \leq n$, we reserve a concept name A_i^{nRC} not appearing in $\mathsf{cl}(D,\mathcal{R},\mathcal{K})$ and define an *extended closure*

$$\mathsf{cl}^+(D,\mathcal{R},\mathcal{K}) := \mathsf{cl}(D,\mathcal{R},\mathcal{K}) \cup \{A_1^{nRc},\ldots,A_n^{nRc} \mid (\geqslant n\ R\ C) \in \mathsf{cl}(D,\mathcal{R},\mathcal{K})\},$$

Let $\mathsf{O_a}$ and $\mathsf{O_c}$ be disjoint and countably infinite sets of *abstract* and *concrete nodes.* A *completion forest* for D, $\mathcal{R}$, and $\mathcal{K}$ is a structure $\mathbf{F} = (V_a, V_c, E, \mathcal{L})$ such that

- $V_a \subseteq \mathsf{O_a}$, $V_c \subseteq \mathsf{O_c}$,
- there is a node $s_0 \in V_a$ such that $D \in \mathcal{L}(s_0)$,
- $\mathcal{L} : \mathbf{S}_a \to 2^{\mathsf{cl}^+(D,\mathcal{R},\mathcal{K})}$ maps each abstract node to a subset of $\mathsf{cl}^+(D,\mathcal{R},\mathcal{K})$,
- each edge $(a,b) \in E$ with $a,b \in V_a$ is labeled with a non-empty set of role names $\mathcal{L}(a,b)$ occurring in D, $\mathcal{R}$, or $\mathcal{K}$, and
- each edge $(a,x) \in E$ with $a \in V_a$ and $x \in V_c$ is labeled with a concrete feature $\mathcal{L}(a,x)$ occurring in D, $\mathcal{R}$, or $\mathcal{K}$.

A *completion system* for D, $\mathcal{R}$, and $\mathcal{K}$ is a structure $S = (\mathbf{F}, \mathcal{P}, \sim_c, \prec)$ such that

- $\mathbf{F} = (V_a, V_c, E, \mathcal{L})$ is a completion forest for D, $\mathcal{R}$, and $\mathcal{K}$,
- $\mathcal{P}$ maps each n-ary concrete predicate occurring in $\mathsf{cl}(D,\mathcal{R},\mathcal{K})$ to a set of n-tuples in V_c,
- $\sim_\mathsf{c}$ is an equivalence relation on V_c, and
- $\prec$ is a linear ordering on V_a.

A node $t \in V_a$ is called an *R-successor* of a node $s \in V_a$ if, for some R' with $R' \underline{\sqsubseteq}^* R$, we have $R' \in \mathcal{L}(s,t)$. A node $x \in V_c$ is called a *g-successor* of a node $s \in V_a$ if $\mathcal{L}(s,x) = g$. Finally, we write $s \neq t$ if s and t are R-successors of the same node and there is some $A_i^{nRC} \in \mathcal{L}(s)$ and $A_j^{nRC} \in \mathcal{L}(t)$ with $i \neq j$. ◊

Some remarks are in order here. Firstly, in contrast to the $\mathcal{ALCOK}(\mathcal{D})$ case, the relation $\prec$ is no longer required to respect the level of a node. This is due to the fact that (a) we have to enforce termination artificially anyway, and this property of $\prec$ is not used to prove termination, and (b) the level of a node might change anyway since a node might become a root node because some completion rules will remove edges.

Secondly, the relation $\sim_{\mathsf{c}}$ will be returned by the concrete domain solver, and is used to compute a relation $\approx_{\mathsf{a}}$ which is then used by the tableau algorithm. However, we do not need to compute the relation $\approx_{\mathsf{c}}$ from $\approx_{\mathsf{a}}$ since, in contrast to the $\mathcal{ALCOK}(\mathcal{D})$ case, all concepts and key boxes are assumed to be path-free.

Thirdly, the new concept names A_i^{nRC} are introduced to ensure that successors of a node x introduced for some $(\geqslant n\ R\ C) \in \mathcal{L}(x)$ will not be merged later—neither by the completion rules, nor when we construct a tableau. Intuitively, we construct a finite, cyclic tableau for a complete and clash-free completion system, and we re-use nodes: sometimes, instead of having an edge to an R-successor y of x, we will have an edge to "a node z similar to y". Now, if a node z could be "similar" to two R-successor y_1, y_2 of x, we might not have enough R-successors of x to satisfy a restriction $(\geqslant n\ R\ C) \in \mathcal{L}(x)$.

Since $\mathcal{SHOQK}(\mathcal{D})$ provides for transitive roles, we need some cycle-detection mechanism in order to guarantee termination of our algorithm: roughly spoken, if we encounter a node which is "similar" to an already existing one, then this node does not need to be further explored. Speaking in terms of [33, 10], we employ a mechanism called *subset blocking.*

Definition 46 (Blocked). Let $\preceq$ be the reflexive closure of $\prec$. A node $t \in V_a$ is *blocked by* a node $s \in V_a$ if $\mathcal{L}(t) \subseteq \mathcal{L}(s)$, and $s \preceq s'$, for all s' with $\mathcal{L}(t) \subseteq \mathcal{L}(s')$. $\Diamond$

Note that, unlike to what is done, e.g., in [33], the blocking node is not necessarily an ancestor of the blocked node, but can be anywhere in the forest. This modification is used to design a NExpTime algorithm. Moreover, blocked nodes may have unblocked successors.

To decide the satisfiability of an $\mathcal{ALCOK}(\mathcal{D})$-concept D w.r.t. a role box $\mathcal{R}$ and a path-free key box $\mathcal{K}$ (where D and $\mathcal{K}$ are in NNF), the tableau algorithm is started with the *initial completion system*

$$\begin{array}{rcl} S_D &=& (\mathbf{F}_D, \mathcal{P}_\emptyset, \emptyset, \emptyset), \text{ where} \\ \mathbf{F}_D &=& (\{s_0\}, \emptyset, \emptyset, \{s_0 \mapsto \{D\}\}) \text{ and} \\ \mathcal{P}_\emptyset & & \text{maps each } P \in \Phi_\mathcal{D} \text{ occurring in } D \text{ and } \mathcal{K} \text{ to } \emptyset. \end{array}$$

Then the algorithm repeatedly applies completion rules. Before the actual rules are given, we introduce some new notions: we use $S^{\mathbf{F}}(s, C)$ to denote the set

$$\{t \in \mathbf{S}_a \mid t \text{ is an } S\text{-successor of } s \text{ in } \mathbf{F} \text{ and } C \in \mathcal{L}(t)\}.$$

For $s, t \in \mathbf{S}_a$, we write $s \approx_{\mathsf{a}} t$ if one of the following conditions is satisfied:

- $N \in \mathcal{L}(s) \cap \mathcal{L}(t)$ for some nominal N or
- $(g_1 \ldots, g_n \text{ keyfor } C) \in \mathcal{K}$, $C \in \mathcal{L}(s) \cap \mathcal{L}(t)$, there are x_i, y_i such that $g_i \in E(s, x_i) \cap E(t, y_i)$ and $x_i \sim_{\mathsf{c}} y_i$ for $1 \leq i \leq n$.

Intuitively, two abstract nodes related via the $\approx_{\mathsf{a}}$ relation describe the same individual in a tableau and should thus have the same label. Note that $\approx_{\mathsf{a}}$ might change after each rule application. However, as mentioned above, we do not use the $\approx_{\mathsf{a}}$ relation to compute a relation $\approx_{\mathsf{c}}$. Intuitively, we do not need $\approx_{\mathsf{c}}$ since, if $s \approx_{\mathsf{a}} t$, then we only "keep" the one smaller w.r.t. $\prec$ and hence do not care if both s and t have a g-successor for a concrete feature g.

We are now ready to formulate the completion rules, which are given in Figure 11. Some abbreviations are used in the formulation of the $\mathsf{R}\leqslant$ and $\mathsf{R}\exists\mathsf{c}$ rules (written in italics), which have the following meaning:

- To *remove an abstract node s and all its incoming and outgoing edges*, remove s from V_a and each (s,t) and (t,s) from E for all $t \in V_a \cup V_c$.
- *Adding a g-successor* of an abstract node s means doing nothing if there exists a g-successor $x \in V_c$ of s and, otherwise, adding $E(s,x) = g$ for some $x \in V_c$ that does not yet occur in the completion forest.
- To *update the relation* $\sim_{\mathsf{c}}$, the concrete domain solver is asked to decide the satisfiability of the $\mathcal{D}$-conjunction

$$\bigwedge_{\substack{P \text{ used in} D,\mathcal{K} \\ (x_1,\ldots,x_n)\in\mathcal{P}(P)}} P(x_1,\ldots,x_n) \wedge \bigwedge_{x \sim_c y} x = y$$

  and returns, in case that this conjunction is satisfiable, an "updated" concrete equivalence $\sim_{\mathsf{c}}$ as defined in Definition 29.

Some explanation of the rules is in order. The rules $\mathsf{R}\sqcup$, $\mathsf{R}\leqslant$, $\mathsf{R}\exists\mathsf{c}$, and Rch are non-deterministic, i.e., their application has more than one possible outcome. For the $\mathsf{R}\exists\mathsf{c}$ rule, this is true due to the update operation performed on $\sim_{\mathsf{c}}$ using the concrete domain reasoner: as discussed at the end of Section 4.1, computing a concrete equivalence for a given $\mathcal{D}$-conjunction may result in a high degree of non-determinism. Please note that, in contrast to $\mathcal{ALCOK}(\mathcal{D})$, we now only need to call the concrete domain in one rule—and not after each rule application.

Next, the $\mathsf{R}\approx_{\mathsf{a}}$ rule simply takes care that two similar nodes $s \approx_{\mathsf{a}} b$ have the same label, i.e., if $s \prec t$, then s will possibly block t (if t is not blocked by other nodes), and thus $\mathcal{L}(t)$ is added to $\mathcal{L}(s)$.

The $\mathsf{R}\leqslant$ rule removes a surplus R-successor t of a node s with $(\leqslant n\ R\ C) \in \mathcal{L}(s)$. Since the subtree below s is not removed, t's successor are new, additional root nodes. This behavior is the reason why we work on a completion forest. Moreover, no other rule removes nodes or edges and would thus yield "new" root nodes.

As in the previous section, the tableau algorithm stops applying rules if it finds an obvious contradiction (a "clash") or no more completion rules are applicable.

Definition 47 (Clash). Let $S = (\mathbf{F}, \mathcal{P}, \sim_c, \prec)$ be a completion system for D, $\mathcal{R}$, and $\mathcal{K}$ and $\mathbf{F} = (V_a, V_c, E, \mathcal{L})$. Then S is said to contain a *clash* if one of the following conditions applies:

R$\sqcap$	if $C_1 \sqcap C_2 \in \mathcal{L}(s)$, s is not blocked, and $\{C_1, C_2\} \not\subseteq \mathcal{L}(s)$, then $\mathcal{L}(s) := \mathcal{L}(s) \cup \{C_1, C_2\}$
R$\sqcup$	if $C_1 \sqcup C_2 \in \mathcal{L}(s)$, s is not blocked, and $\{C_1, C_2\} \cap \mathcal{L}(s) = \emptyset$, then $\mathcal{L}(s) := \mathcal{L}(s) \cup \{C\}$ for some $C \in \{C_1, C_2\}$
R$\exists$	if $\exists R.C \in \mathcal{L}(s)$, s is not blocked, and s has no R-successor t with $C \in \mathcal{L}(t)$ then create a new node t such that $t' \prec t$ for all $t' \in \mathbf{S}_a$ and set $E(s,t) := \{R\}$ and $\mathcal{L}(t) := \{C\}$
R$\geqslant$	if $(\geqslant n\ S\ C) \in \mathcal{L}(s)$, s is not blocked, and there are no n S-successors $t_1, \ldots, t_n$ of s with $C \in \mathcal{L}(t_i)$ and $t_i \neq t_j$ for $1 \leq i < j \leq n$, then create n new nodes $t_1, \ldots, t_n$ s.t. $t' \prec t_i$ for $1 \leq i \leq n$ and all $t' \in \mathbf{S}_a$, and set $E(s, t_i) := \{S\}$ and $\mathcal{L}(t_i) := \{C, A_i^{nSC}\}$ for $1 \leq i \leq n$
R$\leqslant$	if $(\leqslant n\ S\ C) \in \mathcal{L}(s)$, s is not blocked, s has $n+1$ S-successors $t_0, \ldots, t_n$ with $C \in \mathcal{L}(t_i)$ for $0 \leq i \leq n$, then choose i, j such that $t_i \prec t_j$, set $\mathcal{L}(t_i) := \mathcal{L}(t_i) \cup \mathcal{L}(t_j)$ and *remove t_j and all its incoming and outgoing edges*
R$\exists$c	if $\exists g_1, \ldots, g_n.P \in \mathcal{L}(s)$, s is not blocked, and there are no g_i-successors x_i with $(x_1, \ldots, x_n) \in \mathcal{P}(P)$ then *add a g_i-successor of* s for each $1 \leq i \leq n$, for y_i the g_i-successor of s, add $(y_1, \ldots, y_n)$ to $\mathcal{P}(P)$, and *update* $\sim_c$
R$\forall$	if $\forall R.C \in \mathcal{L}(s)$, s is not blocked, and there is an R-successor t of s with $C \notin \mathcal{L}(t)$, then $\mathcal{L}(t) := \mathcal{L}(t) \cup \{C\}$
R$\forall_+$	if $\forall S.C \in \mathcal{L}(s)$, s is not blocked, there is some R with $\mathsf{Trans}(R)$ and $R \sqsubseteq^{*} S$, and an R-successor t of s with $\forall R.C \notin \mathcal{L}(t)$, then $\mathcal{L}(t) := \mathcal{L}(t) \cup \{\forall R.C\}$
Rch	if s is an S-successor of s' and $(\leqslant n\ S\ C) \in \mathcal{L}(s')$ or s has g_i-successors x_i for all $1 \leq i \leq n$ and $(g_1, \ldots g_n$ keyfor $C) \in \mathcal{K}$ and s is not blocked and $\{C, \dot\neg C\} \cap \mathcal{L}(s) = \emptyset$, then $\mathcal{L}(s) := \mathcal{L}(s) \cup \{E\}$ for some $E \in \{C, \dot\neg C\}$
R$\approx_\mathsf{a}$	if $s \approx_\mathsf{a} t$, $\mathcal{L}(t) \not\subseteq \mathcal{L}(s)$, $s \prec t$, and s is not blocked, then set $\mathcal{L}(s) := \mathcal{L}(s) \cup \mathcal{L}(t)$

Figure 11: The completion rules for $\mathcal{SHOQK}(\mathcal{D})$.

(C1) for some concept name $A \in \mathsf{N_C}$ and some node $s \in V_a$, $\{A, \neg A\} \subseteq \mathcal{L}(s)$;

(C2) the $\mathcal{D}$-conjunction

$$\bigwedge_{\substack{P \text{ used in} D, \mathcal{K} \\ (x_1, \ldots, x_n) \in \mathcal{P}(P)}} P(x_1, \ldots, x_n) \wedge \bigwedge_{x \sim_c y} x = y$$

is not satisfiable;

(C3) $s \neq s$ for some $s \in V_a$;

(C4) for some $s \in V_a$ and $g \in \mathsf{N_{cF}}$, we have $g\uparrow \in \mathcal{L}(s)$ and s has a g-successor.

A completion system not containing a clash is called *clash-free*. The completion system is *complete* if it contains a clash or if none of the completion rules is applicable. $\Diamond$

Due to the simplicity of the algorithm, we refrain from describing it in pseudo-code notation: the algorithm starts with the initial completion system and then repeatedly applies the completion rules. If a clash is detected, it returns unsatisfiable. If a complete and clash-free completion system is found, then the algorithm returns satisfiable. Note that, since some of the completion rules are non-deterministic, the algorithm is also non-deterministic.

We are now ready for proving termination, soundness, and completeness of the tableau algorithm, starting with termination. In the following, we use $|D, \mathcal{R}, \mathcal{K}|$ to denote $|\,\mathsf{cl}^+(D, \mathcal{R}, \mathcal{K})|$. Recall that this number is polynomial in the size of D, $\mathcal{R}$, $\mathcal{K}$.

Lemma 48 (Termination). *When started with a $\mathcal{SHOQK}(\mathcal{D})$ concept D in NNF, a role box $\mathcal{R}$, and a path-free key box $\mathcal{K}$ in NNF, the tableau algorithm terminates.*

Proof. Assume that there are D, $\mathcal{R}$, and $\mathcal{K}$ such that the tableau algorithm does not terminate. This means that there is an infinite sequence $S_0, S_1, \ldots$ of completion systems such that (a) S_0 is the initial completion system S_D and (b) S_{i+1} is the result of applying a completion rule to S_i. This is only possible if the $\mathsf{R}\exists$ or the $\mathsf{R}\geqslant$ rules are applied infinitely often: it is easily seen that the rules $\mathsf{R}\sqcap$, $\mathsf{R}\sqcup$, $\mathsf{R}\leqslant$, $\mathsf{R}\exists\mathsf{c}$, $\mathsf{R}\forall$, $\mathsf{R}\forall_+$, Rch, and $\mathsf{R}{\approx}_\mathsf{a}$ can only be applied finitely often to completion systems whose set of abstract nodes V_a does not increase since they either add concepts into node labels (whose size is bound), they add concrete nodes (whose number is bound linearly by the number of abstract nodes), or they remove abstract nodes from the tree. Hence there is a sub-sequence $S_{i_1}, S_{i_2}, \ldots$ such that S_{i_j} is the result of applying the $\mathsf{R}\exists$ or the $\mathsf{R}\geqslant$ rule to S_{i_j-1}. Let s_{i_ℓ} be the abstract node to which the $\mathsf{R}\exists$ or the $\mathsf{R}\geqslant$ rule was applied in $S_{i_\ell-1}$. Now, since $s_\ell \prec s_k$ implies that s_ℓ was *not* generated after s_k and the number of successor nodes of a node is bound, we find a further sub-sequence $S_{j_1}, S_{j_2}, \ldots$ satisfying

$$s_{j_k} \prec s_{j_{k+1}}.$$

Let $\mathcal{L}_j$ be the labeling function in S_j. Since each abstract node is labeled with a subset $\mathcal{L}_j$ of $\mathsf{cl}^+(D, \mathcal{R}, \mathcal{K})$, there are nodes $s_{j_k} \prec s_{j_\ell}$ with $k \lneq \ell$ and $\mathcal{L}_{j_k}(s_{j_k}) = \mathcal{L}_{j_\ell}(s_{j_\ell})$. Now node labels can only increase and, if a node t is removed, its label is conjoined to the label of a node $\hat{t}$ with $\hat{t} \prec t$. Thus there is a node t in the completion system S_{j_ℓ} with $t \prec s_{j_\ell}$ and $\mathcal{L}_{j_\ell}(s_{j_\ell}) \subseteq \mathcal{L}_{j_\ell}(t)$. By definition, s_{j_ℓ} is thus blocked in S_{j_ℓ}, contradicting the assumption that the $\mathsf{R}\exists$ or the $\mathsf{R}\geqslant$ rule is applied to s_{j_ℓ} in S_{j_ℓ}. ❑

Lemma 49 (Soundness). *If the expansion rules can be applied to a $\mathcal{SHOQK}(\mathcal{D})$ concept D in NNF, a role box $\mathcal{R}$, and a path-free key box $\mathcal{K}$ such that they yield a complete and clash-free completion forest, then D has a tableau w.r.t. $\mathcal{R}$ and $\mathcal{K}$.*

Proof. Let $\mathbf{T} = ((V_a, V_c, E, \mathcal{L}), \mathcal{P}, \sim_c, \prec)$ be a complete and clash-free completion system. Clash-freeness implies the existence of a solution δ for the concrete predicates in $\mathbf{T}$ satisfying $\delta(x) = \delta(y)$ iff $x \sim_{\mathsf{c}} y$ according to Definition 29. From $\mathbf{T}$ and δ, we define a finite tableau $T = (\mathbf{S}_a, \mathbf{S}_c, \hat{E}, \hat{\mathcal{L}}, \hat{\mathcal{P}})$ as follows:

$$\begin{aligned}
\mathbf{S}_a &:= \{s \in V_a \mid s \text{ occurs in } \mathbf{T} \text{ and is not blocked}\}\\
\mathbf{S}_c &:= \{\delta(x) \mid (s,x) \in E(g) \text{ for some } s \in \mathbf{S}_a \text{ and some } g\}\\
\hat{\mathcal{L}}(s) &:= \mathcal{L}(s) \cap \mathsf{cl}(D, \mathcal{R}, \mathcal{K})\\
\hat{E}(s,t) &:= \{S \mid t \text{ is an } S\text{-successor of } s \text{ or } t \text{ blocks an } S\text{-successor } t' \text{ of } s\}\\
e(s,g) &:= \begin{cases} \delta(x) & \text{if } x \text{ is a } g\text{-successor of } s\\ \text{undefined} & \text{if } x \text{ has no } g\text{-successor}\end{cases}\\
\hat{\mathcal{P}} &:= \text{the restriction of } \mathcal{P} \text{ to } \mathbf{S}_c.
\end{aligned}$$

It remains to show that T satisfies **(T1)**–**(T14)**, which is basically a consequence of $\mathbf{T}$ being clash-free and complete.

- **(T1)** is satisfied since $\mathbf{T}$ does not contain a clash **(C1)**.
- **(T2)** is satisfied since the $\mathsf{R}\sqcap$ rule cannot be applied, and thus $C_1 \sqcap C_2 \in \hat{\mathcal{L}}(s)$ implies $C_1, C_2 \in \hat{\mathcal{L}}(s)$.
- **(T3)** is satisfied since the $\mathsf{R}\sqcup$ rule cannot be applied, and thus $C_1 \sqcup C_2 \in \hat{\mathcal{L}}(s)$ implies $\{C_1, C_2\} \cap \hat{\mathcal{L}}(s) \neq \emptyset$.
- For **(T4)**, consider $s, t \in \mathbf{S}_a$ with $R \in \hat{E}(s,t)$ and $R \mathrel{\underline{\overset{*}{\sqsubseteq}}} S$. Then $R \in \hat{E}(s,t)$ implies that t is or blocks an R-successor of s. By definition of "successor", t is or blocks an S-successor of s, and thus $S \in \hat{E}(s,t)$.
- For **(T5)**, let $\forall R.C \in \hat{\mathcal{L}}(s)$ and $R \in \hat{E}(s,t)$. If t is an R-successor of s, then s not being blocked implies $C \in \mathcal{L}(t)$ since the $\mathsf{R}\forall$ rule cannot be applied. If t blocks an R-successor t' of s, then s not being blocked and the fact that the $\mathsf{R}\forall$ rule cannot be applied yields $C \in \mathcal{L}(t')$, and the blocking condition implies $C \in \mathcal{L}(t)$.

  In both cases, we thus have $C \in \hat{\mathcal{L}}(t)$.
- **(T6)** and **(T7)** are satisfied for the same reasons as **(T5)** with $\mathsf{R}\forall$ replaced with $\mathsf{R}\exists$ and $\mathsf{R}\forall_+$.
- For **(T8)**, consider s with $(\geqslant n\ R\ C) \in \hat{\mathcal{L}}(s)$. Hence $(\geqslant n\ R\ C) \in \mathcal{L}(s)$ and completeness of $\mathbf{T}$ implies the existence of R-successors $t_1, \ldots, t_n$ of s with $C \in \mathcal{L}(t_i)$ and $t_i \neq t_j$ for all $i \neq j$. The latter implies, for each $i \neq j$, the existence of $A^{nRC}_{k_i} \in \mathcal{L}(t_i)$ and $A^{nRC}_{k_j} \in \mathcal{L}(t_j)$ with $k_i \neq k_j$. For **(T8)** to be satisfied, it remains to verify that
  - no t_i can block a t_j: if this was the case, the blocking condition would imply that $\{A^{nRC}_{k_i}, A^{nRC}_{k_j}\} \subseteq \mathcal{L}(t_i)$.
  - no t can block both t_i and t_j with $i \neq j$: similarly, this implies that $\{A^{nRC}_{k_i}, A^{nRC}_{k_j}\} \subseteq \mathcal{L}(t)$.

In each case, we would have a clash **(C3)**, in contradiction to **T** being clash-free.

- For **(T9)**, consider s with $(\leqslant n\ R\ C) \in \hat{\mathcal{L}}(s)$. Hence $(\leqslant n\ R\ C) \in \mathcal{L}(s)$ and, since the $\mathsf{R}\leqslant$ rule cannot be applied, there are at most n R-successors t_i of s. Since each t_i is either not blocked or blocked by exactly one other node, there are at most n $u_i \in \mathbf{S}_a$ with $R \in \hat{E}(s, u_i)$ and $C \in \hat{\mathcal{L}}(u_i)$.

- For **(T10)**, let $(\leqslant n\ R\ C) \in \hat{\mathcal{L}}(s)$ and $R \in \hat{E}(s, t)$. Hence $(\leqslant n\ R\ C) \in \mathcal{L}(s)$ and t either is or blocks an R-successor of s. In the first case, non-applicability of the Rch rule implies that $\{C, \dot\neg C\} \cap \mathcal{L}(t) \neq \emptyset$. In the second case, $\{C, \dot\neg C\} \cap \mathcal{L}(t') \neq \emptyset$ for t' the R-successor of s blocked by t, and thus the blocking condition yields $\{C, \dot\neg C\} \cap \mathcal{L}(t) \neq \emptyset$. In both cases, this implies $\{C, \dot\neg C\} \cap \hat{\mathcal{L}}(t) \neq \emptyset$.

  Next, consider $(g_1, \dots, g_n \text{ keyfor } C) \in \mathcal{K}$ and s such that $e(s, g_i)$ is defined for each i. Hence s has a g_i-successor for each i, and thus s not being blocked and the non-applicability of the Rch imply that $\{C, \dot\neg C\} \cap \hat{\mathcal{L}}(t) \neq \emptyset$.

- For **(T11)**, consider $N \in \hat{\mathcal{L}}(s) \cap \hat{\mathcal{L}}(t)$. By definition, $N \in \mathcal{L}(s) \cap \mathcal{L}(t)$ and thus $s \approx_{\mathsf{a}} t$. Moreover, totality of $\prec$ implies that we can assume without loss of generality that $s \prec t$ or $s = t$. Thus non-applicability of the $\mathsf{R}{\approx_{\mathsf{a}}}$ rule implies that $\mathcal{L}(t) \subseteq \mathcal{L}(s)$, and thus t not being blocked implies $s = t$.

- **(T12)** is satisfied since the rule $\mathsf{R}\exists\mathsf{c}$ cannot be applied.

- For **(T13)**, clash-freeness implies the satisfiability of

$$\bigwedge_{P \text{ used in } D,\mathcal{K}} \; \bigwedge_{(x_1,\dots,x_n)\in\mathcal{P}(P)} P(x_1, \dots, x_n). \qquad (*)$$

  By choice of δ, $\delta(x) = \delta(y)$ iff $x \sim_{\mathsf{c}} y$, and thus **(T13)** is satisfied.

- For **(T14)**, let $(g_1, \dots, g_n \text{ keyfor } C) \in \mathcal{K}$, $C \in \hat{\mathcal{L}}(s) \cap \hat{\mathcal{L}}(t)$, and $e(s, g_i) = e(t, g_i)$, for all $1 \leq i \leq n$. Thus $C \in \mathcal{L}(s) \cap \mathcal{L}(t)$ and, by choice of e and δ, we have $x_i \sim_{\mathsf{c}} y_i$ for $g_i \in E(s, x_i) \cap E(t, y_i)$. Hence $s \approx_{\mathsf{a}} t$. Without loss of generality, we assume that $s \prec t$ or $s = t$. Thus non-applicability of the $\mathsf{R}{\approx_{\mathsf{a}}}$ rule implies that $\mathcal{L}(t) \subseteq \mathcal{L}(s)$, and thus t not being blocked implies $s = t$.

- **(T15)** is satisfied by definition of T and since **T** does not contain a clash **(C4)**.

❏

Lemma 50 (Completeness). *If a $\mathcal{SHOQK}(\mathcal{D})$-concept D in NNF has a tableau w.r.t. $\mathcal{R}$ and $\mathcal{K}$, then the expansion rules can be applied to D, $\mathcal{R}$, and $\mathcal{K}$ such that they yield a complete and clash-free completion forest.*

Proof. Given a tableau $T = (\mathbf{S}_a, \mathbf{S}_c, \hat{\mathcal{L}}, \hat{E}, e, \hat{\mathcal{P}})$ for D w.r.t. $\mathcal{R}$ and $\mathcal{K}$, we can steer the non-deterministic rules $\mathsf{R}\sqcup$, Rch, and $\mathsf{R}{\approx_\mathsf{a}}$ in such a way that each rule application preserves clash-freeness. This together with termination from Lemma 48 finishes the proof.

Inductively with the generation of new nodes, we define a mapping π from nodes of the completion tree to individuals in the tableau and concrete values in such a way that

- $\mathcal{L}(s) \cap \mathsf{cl}(D, \mathcal{R}, \mathcal{K}) \subseteq \hat{\mathcal{L}}(\pi(s))$ for each $s \in \mathbf{S}_a$,
- if t is an R-successor of s, then $R \in \hat{E}(\pi(s), \pi(t))$,
- if x is a g-successor of s, then $e(\pi(x), g) = \pi(x)$, and
- if $s \neq t$, then $\pi(s) \neq \pi(t)$.

A mapping satisfying these three conditions is called *correct* in the following. Due to **(T1)** and the first property, we do not encounter a clash **(C1)**. The first and third property together with **(T12)** and **(T13)** ensure that a clash **(C2)** does not occur. A clash **(C3)** cannot occur due to the last property. The first and the third property together with **(T15)** ensure that a clash **(C4)** does not occur.

The total mapping π is inductively defined as follows: let δ be a solution for the equation in **(T13)**. Choose a node $\hat{s}_0$ with $D \in \hat{\mathcal{L}}(\hat{s}_0)$, and set $\pi(s_0) := \hat{s}_0$ for s_0 the root node of the completion tree. Obviously, π is correct. We will now show that each completion rule can be applied in such a way that π either is still correct or that π can be extended to a correct mapping.

- An application of the rule $\mathsf{R}\sqcap$ preserves correctness of π due to **(T2)**.
- Due to **(T3)**, the rule $\mathsf{R}\sqcup$ can be applied such that correctness is preserved.
- If the rule $\mathsf{R}\exists$ adds a new node t for $\exists R.C \in \mathcal{L}(s)$, then correctness implies $\exists R.C \in \hat{\mathcal{L}}(\pi(s))$, and thus **(T6)** implies the existence of some $\hat{t} \in \mathbf{S}_a$ with $R \in E(\pi(s), \hat{t})$ and $C \in \hat{\mathcal{L}}(\hat{t})$. Thus extending π with $\pi(t) := \hat{t}$ obviously yields a correct mapping.
- If the rule $\mathsf{R}\geqslant$ adds n nodes t_i for $(\geqslant n\ R\ C) \in \mathcal{L}(s)$, then correctness implies $(\geqslant n\ R\ C) \in \hat{\mathcal{L}}(\pi(s))$, and thus **(T8)** implies the existence of $\hat{t}_1, \ldots, \hat{t}_n \in \hat{\mathbf{S}}_a$ with $\hat{t}_i \neq \hat{t}_j$ for $i \neq j$, $R \in E(\pi(s), \hat{t}_i)$, and $C \in \hat{\mathcal{L}}(\hat{t}_i)$. Thus extending π with $\pi(t_i) := \hat{t}_i$ obviously yields a correct mapping.
- Assume that the $\mathsf{R}\leqslant$ rule is applicable to a node s with $(\leqslant n\ R\ C) \in \mathcal{L}(s)$ and more than n R-successors t_i with $C \in \mathcal{L}(t_i)$. Then correctness implies that $(\leqslant n\ R\ C) \in \hat{\mathcal{L}}(\pi(s))$, $R \in \hat{E}(\pi(s), \pi(t_i))$, and $C \in \hat{\mathcal{L}}(t_i)$. Thus, by **(T9)**, there are $i \neq j$ with $\pi(t_i) = \pi(t_j)$. Again, correctness implies that *not* $t_i \neq t_j$ and, without loss of generality, we can assume that $t_i \prec t_j$. Hence applying the rule and thereby merging $\mathcal{L}(t_j)$ into $\mathcal{L}(t_i)$ preserves correctness.
- For the rule $\mathsf{R}\exists\mathsf{c}$, π can be extended in a similar way: if a new g_i-successor x_i of s is added, then extending π with $\pi(x_i) := \hat{e}(\pi(s), g_i)$ yields a correct π.

- For the $\mathsf{R}\forall$ rule, π does not need to be extended, and **(T5)**, **(T4)**, and the definition of R-successors imply that correctness is preserved.
- The $\mathsf{R}\forall_+$ rule is similar, with the only difference that **(T7)** takes the place of **(T5)**.
- Due to **(T10)**, the rule Rch can be applied without violating correctness.
- For $\mathsf{R}{\approx_\mathsf{a}}$, we consider two reasons for $\mathsf{R}{\approx_\mathsf{a}}$ to be applicable:
  - $N \in \mathcal{L}(s) \cap \mathcal{L}(t)$. Then correctness of π and **(T11)** imply that $\pi(s) = \pi(t)$.
  - $(g_1, \ldots, g_n \text{ keyfor } C) \in \mathcal{K}$, $C \in \mathcal{L}(s) \cap \mathcal{L}(t)$, and $g_i \in E(s, x_i) \cap E(t, y_i)$ and $x_i \sim_\mathsf{c} y_i$ for $1 \leq i \leq n$. Then correctness implies that $\hat{e}(\pi(s), g_i) = \hat{e}(\pi(t), g_i)$, and thus **(T14)** together with the first property of correctness imply that $\pi(s) = \pi(t)$.

  In both cases, applying $\mathsf{R}{\approx_\mathsf{a}}$ to s and t preserves correctness.

❑

As an immediate consequence of Lemmas 44, 48, 49, and 50, the tableau algorithm always terminates and answers "D is satisfiable w.r.t. $\mathcal{R}$ and $\mathcal{K}$" if and only if the input concept D is satisfiable w.r.t. the input role box $\mathcal{R}$ and the input key box $\mathcal{K}$. Since concept satisfiability w.r.t. TBoxes can be reduced to concept satisfiability without TBoxes, we obtain the following result:

Theorem 51. *The tableau algorithm decides satisfiability of $\mathcal{SHOQK}(\mathcal{D})$ concepts w.r.t. TBoxes, role boxes, and path-free key boxes.*

Since concept subsumption can be reduced to concept (un)satisfiability, the algorithm can also be used to decide subsumption of $\mathcal{SHOQK}(\mathcal{D})$-concepts w.r.t. TBoxes, role boxes, and path-free key boxes.

It is not hard to see that the proof of Lemma 50 together with Lemmas 44 and 48 yield a *bounded model property* for $\mathcal{SHOQK}(\mathcal{D})$: if a $\mathcal{SHOQK}(\mathcal{D})$-concept D is satisfiable w.r.t. a role box $\mathcal{R}$ and a path-free key box $\mathcal{K}$, Lemma 50 implies that the tableau algorithm constructs a complete and clash-free completion forest for D, $\mathcal{R}$, and $\mathcal{K}$. By the definition of blocking, the number of abstract nodes in a completion forest that are not blocked is bounded by 2^m, where $m = \# \mathsf{cl}^+(D, \mathcal{R}, \mathcal{K})$ is polynomial in the size of C, $\mathcal{R}$, and $\mathcal{K}$: if $s \neq t \in V_a$ are abstract nodes in a completion forest and $\mathcal{L}(s) = \mathcal{L}(t)$, then either s blocks t, t blocks s, or they are both blocked by another node u. Moreover, it is easily seen that the number of concrete successors per abstract node is bounded by the number of concrete features in C, $\mathcal{R}$, and $\mathcal{K}$. Now, in the proof of Lemma 50, the abstract nodes in a tableau constructed from a complete and clash-free completion forest coincide with the nodes that are not blocked in the completion forest. Finally, in the proof of Lemma 44, the interpretation domain of a model constructed from a tableau coincides with the abstract nodes in the tableau. Summing up, a $\mathcal{SHOQK}(\mathcal{D})$-concept that is satisfiable w.r.t. $\mathcal{R}$ and $\mathcal{K}$ has a model of size $|\Delta_\mathcal{I}| \leq 2^m$ for $m = \# \mathsf{cl}^+(D, \mathcal{R}, \mathcal{K})$.

Thus "guessing" an interpretation with at most 2^m nodes and then checking whether it is a model of D, $\mathcal{R}$, and $\mathcal{K}$ is an alternative algorithm for deciding satisfiability of

$\mathcal{SHOQ}(\mathcal{D})$-concepts w.r.t. role boxes and key boxes. Since this algorithm can clearly be implemented in NExpTime, Theorem 19 implies the following tight complexity bound.

Theorem 52. *Satisfiability of $\mathcal{SHOQK}(\mathcal{D})$ concepts w.r.t. TBoxes, role boxes, and path-free key boxes is* NExpTime*-complete.*

By the standard reduction of concept subsumption to concept (un)satisfiability and vice versa, we obtain co-NExpTime-completeness for $\mathcal{SHOQK}(\mathcal{D})$-concept subsumption w.r.t. TBoxes, role boxes, and path-free key boxes.

5 Conclusion

In this paper, we have identified key constraints as an interesting extension of description logics with concrete domains. Starting from this observation, we introduced a number of natural description logics and provided a comprehensive analysis of the decidability and complexity of reasoning. The main observation of our investigations is that key boxes may have dramatic consequences on the complexity of reasoning: for example, the PSpace-complete DL $\mathcal{ALC}(\mathcal{D})$ becomes NExpTime-complete if extended with Boolean key boxes and undecidable if extended with general key boxes. We present various properties of concrete domains and key boxes which imply decidability (NExpTime-completeness) of $\mathcal{ALC}(\mathcal{D})$ and $\mathcal{SHOQ}(\mathcal{D})$ w.r.t. key boxes satisfying this property.

We selected $\mathcal{ALC}(\mathcal{D})$ and $\mathcal{SHOQ}(\mathcal{D})$ as the basis for our analysis since, in our opinion, these are the most fundamental description logics with concrete domains. Going one step further, it would be interesting to combine key boxes with other extensions of concrete domains, such as the ones presented in [38] and [39]. To name only one possibility, the extension of both $\mathcal{ALCOK}(\mathcal{D})$ and $\mathcal{SHOQ}(\mathcal{D})$ with inverse roles seems to be a natural idea. Note that inverse roles interact with several of the available means of expressivity: while $\mathcal{ALC}$ with inverse roles is PSpace complete [32], $\mathcal{ALCO}$ with inverse roles is ExpTime-complete [1] and $\mathcal{ALC}(\mathcal{D})$ with inverse roles even NExpTime-complete [39].

Other options for future research are more closely related to the material presented in this paper. For example, is $\mathcal{SHOQK}(\mathcal{D})$-concept satisfiability still decidable if we drop the requirement of key boxes to be path-free? Moreover, we do not know the time complexity of the presented tableau algorithm for $\mathcal{SHOQK}(\mathcal{D})$-concept satisfiability. If it runs in (non-deterministic) exponential time, it would directly yield Theorem 52 rather than via a bounded model property.

References

[1] C. Areces, P. Blackburn, and M. Marx. A road-map on complexity for hybrid logics. In J. Flum and M. Rodríguez-Artalejo, editors, *Computer Science Logic*, number 1683 in Lecture Notes in Computer Science, pages 307–321. Springer-Verlag, 1999.

[2] F. Baader. Logic-based knowledge representation. In M. Wooldridge and M. Veloso, editors, *Artificial Intelligence Today, Recent Trends and Developments*, number 1600 in Lecture Notes in Computer Science, pages 13–41. Springer-Verlag, 1999.

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

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

[5] 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. Springer-Verlag, 1992.

[6] F. Baader, I. Horrocks, and U. Sattler. Description logics for the semantic web. *KI – Künstliche Intelligenz*, 3, 2002. To appear.

[7] F. Baader, C. Lutz, H. Sturm, and F. Wolter. Fusions of description logics and abstract description systems. *Journal of Artificial Intelligence Research (JAIR)*, 16:1–58, 2002.

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

[9] 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)*, pages 336–340. John Wiley & Sons, 1998.

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

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

[12] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, 284(5):34–43, 2001.

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

[14] A. Borgida and P. F. Patel-Schneider. A semantics and complete algorithm for subsumption in the classic description logic. *Journal of Artificial Intelligence Research*, pages 277–308, 1994.

[15] A. Borgida and G. E. Weddell. Adding uniqueness constraints to description logics (preliminary report). In F. Bry, R. Ramakrishnan, and K. Ramamohanarao,

editors, *Proceedings of the 5th International Conference on Deductive and Object-Oriented Databases (DOOD97)*, volume 1341 of *LNCS*, pages 85–102. Springer, 1997.

[16] D. Calvanese, G. De Giacomo, and M. Lenzerini. On the decidability of query containment under constraints. In *Proceedings of the 17th ACM SIGACT SIGMOD SIGART Symposium on Principles of Database Systems (PODS'98)*, pages 149–158, 1998.

[17] D. Calvanese, G. De Giacomo, and M. Lenzerini. Keys for free in description logics. In F. Baader and U. Sattler, editors, *Proceedings of the 2000 International Workshop in Description Logics (DL2000)*, number 33 in CEUR-WS (http://ceur-ws.org/), pages 79–88, 2000.

[18] D. Calvanese, M. Lenzerini, and D. Nardi. Description logics for conceptual data modeling. In J. Chomicki and G. Saake, editors, *Logics for Databases and Information Systems*, pages 229–263. Kluwer Academic Publisher, 1998.

[19] M. Dean, D. Connolly, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. Web ontology language (OWL) reference version 1.0. W3C Working Draft, 2002.

[20] D. Fensel, F. van Harmelen, I. Horrocks, D. McGuinness, and P. F. Patel-Schneider. OIL: An ontology infrastructure for the semantic web. *IEEE Intelligent Systems*, 16(2):38–45, 2001.

[21] R. L. Graham, D. E. Knuth, and O. Patashnik. *Concrete Mathematics.* Addison Wesley Publ. Co., Reading, Massachussetts, 1990.

[22] V. Haarslev, C. Lutz, and R. Möller. Foundations of spatioterminological reasoning with description logics. In A. Cohn, L. Schubert, and S.C.Shapiro, editors, *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98)*, pages 112–124. Morgan Kaufman, 1998.

[23] V. Haarslev and R. Möller. RACER system description. In R. Goré, A. Leitsch, and T. Nipkow, editors, *Proceedings of the First International Joint Conference on Automated Reasoning (IJCAR'01)*, number 2083 in Lecture Notes in Artifical Intelligence, pages 701–705. Springer-Verlag, 2001.

[24] V. Haarslev, R. Möller, and M. Wessel. The description logic $\mathcal{ALCNH}_{R^+}$ extended with concrete domains: A practically motivated approach. In R. Goré, A. Leitsch, and T. Nipkow, editors, *Proceedings of the First International Joint Conference on Automated Reasoning IJCAR'01*, number 2083 in Lecture Notes in Artifical Intelligence, pages 29–44. Springer-Verlag, 2001.

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

[26] B. Hollunder and F. Baader. Qualifying number restrictions in concept languages. In *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*, pages 335–346, Boston, MA, USA, 1991.

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

[28] I. Horrocks. Using an expressive description logic: Fact or fiction? In *Proceedings of the Sixth International Conference on the Principles of Knowledge Representation and Reasoning (KR98)*, pages 636–647, 1998.

[29] I. Horrocks. Reasoning with expressive description logics: Theory and practice. In A. Voronkov, editor, *Proceedings of the 18th International Conference on Automated Deduction (CADE 2002)*, number 2392 in Lecture Notes in Artificial Intelligence, pages 1–15. Springer, 2002.

[30] I. Horrocks, P. F. Patel-Schneider, and F. van Harmelen. Reviewing the design of DAML+OIL: An ontology language for the semantic web. In *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI 2002)*, 2002.

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

[32] 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, 1999.

[33] I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for very expressive description logics. *Logic Journal of the IGPL*, 8(3):239–264, 2000.

[34] G. Kamp and H. Wache. CTL - a description logic with expressive concrete domains. Technical Report LKI-M-96/01, Laboratory for Artificial Intelligence (LKI), Universitity of Hamburg, Germany, 1996.

[35] V. L. Khizder, D. Toman, and G. E. Weddell. On decidability and complexity of description logics with uniqueness constraints. In J. V. den Bussche and V. Vianu, editors, *Proceedings of the 8th International Conference on Database Theory (ICDT2001)*, volume 1973 of *LNCS*, pages 54–67. Springer, 2001.

[36] D. Knuth. *The Art of Computer Programming*, volume 1. Addison-Wesley, 1968.

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

[38] C. Lutz. Description logics with concrete domains—a survey. In *Advances in Modal Logics (AiML) 2002*, 2002.

[39] C. Lutz. NExpTime-complete description logics with concrete domains. *ACM Transactions on Computational Logic*, 2002. to appear.

[40] C. Lutz. PSPACE reasoning with the description logic $\mathcal{ALCF}(\mathcal{D})$. *Logic Journal of the IGPL*, 10(5):535–568, 2002.

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

[42] J. Z. Pan and I. Horrocks. Reasoning in the $\mathcal{SHOQ}(\mathcal{D}_n)$ description logic. In I. Horrocks and S. Tessaris, editors, *Proceedings of the International Workshop in Description Logics 2002 (DL2002)*, number 53 in CEUR-WS (http://ceur-ws.org/), pages 53–62, 2002.

[43] E. M. Post. A variant of a recursively unsolvable problem. *Bulletin of the American Mathematical Society*, 52:264–268, 1946.

[44] K. D. Schild. A correspondence theory for terminological logics: Preliminary report. In J. Mylopoulos and R. Reiter, editors, *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 466–471. Morgan Kaufmann, 1991.