Keys, Nominals, and Concrete Domains

Carsten Lutz[1] **Carlos Areces**[2] **Ian Horrocks**[3] **Ulrike Sattler**[1]

[1]Inst. for Theoretical Computer Science
Technical University Dresden
Germany
⟨lastname⟩@tcs.inf.tu-dresden.de

[2] INRIA Lorraine
Nancy
France
Carlos.Areces@loria.fr

[3]Dept. of Computer Science
University of Manchester
UK
horrocks@cs.man.ac.uk

Abstract

Many description logics (DLs) combine knowledge representation on an abstract, logical level with an interface to "concrete" domains such as numbers and strings. We propose to 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 present (un)decidability results and tight NEXPTIME complexity bounds.

1 Introduction

Description Logics (DLs) are a family of popular knowledge representation formalisms. Many expressive DLs combine powerful logical languages with an interface to concrete domains (e.g., integers, reals, strings) and built-in predicates (e.g., $<$, sub-string-of) [Lutz, 2002b]. These can be used to form descriptions such as "employee working for the government and earning more than her boss" that combine "abstract" logical components (e.g., working for the government) with components using concrete domains and predicates (e.g., a numerical comparison of earnings).

DLs with concrete domains have turned out to be useful for reasoning about conceptual (database) models [Lutz, 2002e], and as the basis for expressive ontology languages [Horrocks *et al.*, 2002]. So far, however, they have not been able to express *key constraints*, i.e., constraints expressing the fact that certain "concrete features" uniquely determine the identity of the instances of a certain class. E.g., the concrete feature "social security number (SSN)" might serve as a key for citizens of the US, and the combination of identification number and manufacturer might serve as a key for vehicles. Such constraints are important both in databases and in realistic ontology applications. In a DL context, key constraints have so far only been considered on logical, "non-concrete" domains [Borgida and Weddell, 1997; Calvanese *et al.*, 2000; Khizder *et al.*, 2001; Toman and Weddell, 2002].

It is easy to see that concrete keys can express *nominals*, i.e., concepts to be interpreted as singleton sets (closely related to the "one-of" operator): e.g., if SSN is a key for Human (SSN keyfor Human), then the concept "Human with SSN 1234" ($\mathtt{Human} \sqcap \exists \mathtt{SSN}.=_{1234}$) has at most one instance.

In this paper, we extend the well-known DLs with concrete domains $\mathcal{ALC}(\mathcal{D})$ and $\mathcal{SHOQ}(\mathcal{D})$ [Baader and Hanschke, 1991; Horrocks and Sattler, 2001] with key constraints and analyse the complexity of reasoning with the resulting logics $\mathcal{ALCOK}(\mathcal{D})$ and $\mathcal{SHOQK}(\mathcal{D})$. We show that allowing complex concepts to occur in key constraints dramatically increases the complexity of $\mathcal{ALC}(\mathcal{D})$ (which is PSPACE-complete): it becomes undecidable. Restricting key constraints to atomic concepts (such as "Human" in the above example) still yields a NEXPTIME-hard formalism, even for rather simple (PTIME) concrete domains. We show several variants of this result that depend on other characteristics of key constraints, such as the number of concrete features and the "path length". This effect is consistent with the observation that the PSPACE upper bound for $\mathcal{ALC}(\mathcal{D})$ is not robust [Lutz, 2003].

Additionally, we prove the NEXPTIME bounds to be tight by presenting tableau algorithms for $\mathcal{ALCOK}(\mathcal{D})$ and $\mathcal{SHOQK}(\mathcal{D})$ with *key admissible* concrete domains that are in NP, where key admissibility is a simple and natural property. We have chosen to devise tableau algorithms since they have the potential to be implemented in efficient reasoners and have been shown to behave well in practise [Horrocks *et al.*, 2000]. Due to space restrictions, we can only sketch proofs and refer to [Lutz *et al.*, 2002] for more details.

2 Preliminaries

First, we formally introduce the description logic $\mathcal{ALCOK}(\mathcal{D})$.

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

Let $\mathsf{N_C}$, $\mathsf{N_O}$, $\mathsf{N_R}$, $\mathsf{N_{cF}}$ be pairwise disjoint and countably infinite sets of *concept names*, *nominals*, *role names*, and *concrete features*. We assume that $\mathsf{N_R}$ has 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 $f_1, \ldots, f_n$ ($n \geq 0$) 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 (i) every concept name and every nominal is a concept, and (ii) if C and D are concepts, R is a role name, g is a concrete feature, $u_1, \ldots, u_n$ are paths, and $P \in \Phi_\mathcal{D}$ is a predicate of arity n, then the following expressions are also concepts:

$\neg C$, $C \sqcap D$, $C \sqcup D$, $\exists R.C$, $\forall R.C$, $\exists u_1, \ldots, u_n.P$, and $g\uparrow$.

A *key definiton* is an expression $(u_1, \ldots, u_k \text{ keyfor } C)$ for $u_1, \ldots, u_k$ $(k \geq 1)$ paths and C a concept. A finite set of key definitions is called a *key box*. $\diamondsuit$

As usual, we use $\top$ to denote an arbitrary propositional tautology. Throughout this paper, we will consider several fragments of the logic $\mathcal{ALCOK}(\mathcal{D})$: $\mathcal{ALCO}(\mathcal{D})$ is obtained from $\mathcal{ALCOK}(\mathcal{D})$ by admitting only empty key boxes; 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 standard inference problems: concept satisfiability and concept subsumption.

Definition 2. 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 and $d \in \Delta_\mathcal{I}$, 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:

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

An interpretation $\mathcal{I}$ 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 $a, b \in C^\mathcal{I}$, $u_i^\mathcal{I}(a) = u_i^\mathcal{I}(b)$ for $1 \leq i \leq n$ implies $a = b$. $\mathcal{I}$ 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.* $\mathcal{K}$ (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 DLs 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 $C \not\sqsubseteq_\mathcal{K} \neg\top$. Thus we can concentrate on concept satisfiability when investigating the complexity of reasoning: the above reduction implies the corresponding bounds for subsumption and the complementary complexity class (usually co-NExpTime in this paper).

When devising decision procedures for DLs which are not tied to a *particular* concrete domain, *admissibility* of the concrete domain usually serves as a well-defined interface between the decision procedure and concrete domain reasoners [Baader and Hanschke, 1991; Lutz, 2002b]:

Definition 3. Let $\mathcal{D}$ be a concrete domain. A $\mathcal{D}$*-conjunction* is a (finite) predicate conjunction of the form

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

where P_i is an n_i-ary predicate for $i < k$ and the $x_j^{(i)}$ are variables. 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 $i < k$. We say that the concrete domain $\mathcal{D}$ is *admissible* iff (i) $\Phi_\mathcal{D}$ contains a name $\top_\mathcal{D}$ for $\Delta_\mathcal{D}$; (ii) $\Phi_\mathcal{D}$ is closed under negation, and (iii) satisfiability of $\mathcal{D}$-conjunctions is decidable. We refer to the satisfiability of $\mathcal{D}$-conjunctions as $\mathcal{D}$*-satisfiability*. $\diamondsuit$

As we shall see, it sometimes makes a considerable difference w.r.t. complexity and decidability to restrict key boxes in various ways. Because of this, it is convenient to introduce the following notions:

Definition 4. 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 $(g_1, \ldots, g_n \text{ keyfor } C)$ with $g_1, \ldots, g_n \in \mathsf{N_{cF}}$; *simple* if it is both path-free and Boolean; and a *unary key box* if all key definitions in $\mathcal{K}$ are of the form $(u \text{ keyfor } C)$. 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. $\diamondsuit$

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 Complexity Bounds

In this section, we present lower complexity bounds for DLs with concrete domains, key boxes and nominals. We start by showing that 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, as shown in Section 4, the restriction to Boolean key boxes recovers decidability. Next, we prove that satisfiability of path-free $\mathcal{ALCK}(\mathcal{D})$-concepts w.r.t. simple key boxes is NExpTime-hard for many concrete domains and that this holds even if we restrict ourselves to unary key boxes. Finally, we identify a concrete domain such that $\mathcal{ALCO}(\mathcal{D})$-concept satisfiability (without key boxes) is already NExpTime-hard.

Undecidability of $\mathcal{ALCK}(\mathcal{D})$-concept satisfiability w.r.t. general key boxes is proved by reduction of the undecidable Post Correspondence Problem (PCP) [Post, 1946].

Definition 5. An *instance* P of the PCP 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 problem is to decide whether a given instance P has a solution. $\diamondsuit$

The reduction uses the admissible concrete domain W introduced in [Lutz, 2003], whose domain is the set of words over Σ and whose predicates express concatenation of words. For each PCP instance $P = (\ell_1, r_1), \ldots, (\ell_k, r_k)$, we define a concept C_P and unary key box $\mathcal{K}_P$ such that P has *no* solution iff C_P is satisfiable w.r.t. $\mathcal{K}_P$. Intuitively, C_P and $\mathcal{K}_P$ enforce an infinite, k-ary tree, where each node represents a sequences of integers, i.e. a potential solution. The role of the key box is to guarantee that the tree is of infinite depth; concrete features are used to store the left and right concatenations corresponding to the potential solutions; and concatenation predicates from the concrete domain W are used to compute them. Finally, an inequality predicate also provided by W is used to guarantee that none of the potential solutions is indeed a solution. Since it is known that W-satisfiability is in PTIME [Lutz, 2003], we obtain the following theorem.

Theorem 6. *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.*

As shown in [Lutz, 2003; Lutz *et al.*, 2002], the reduction can easily be adapted to more natural concrete domains such as numerical ones based on the integers and providing predicates for equality to zero and one, binary equality, addition, and multiplication.

We now establish lower bounds for $\mathcal{ALCK}(\mathcal{D})$ with Boolean key boxes and for $\mathcal{ALCO}(\mathcal{D})$. These results are obtained using a NEXPTIME-complete variant of the well-known, undecidable domino problem [Knuth, 1968].

Definition 7. 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. For $\mathfrak{D}$ a domino system and $a = a_0, \ldots, a_{n-1} \in T^n$ an *initial condition*, 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: (i) if $\tau(x, y) = t$ and $\tau(x+1 \mod 2^{n+1}, y) = t'$, then $(t, t') \in H$; (ii) if $\tau(x, y) = t$ and $\tau(x, y+1 \mod 2^{n+1}) = t'$, then $(t, t') \in V$; and (iii) $\tau(i, 0) = a_i$ for $i < n$. $\Diamond$

This variant of the domino problem is NEXPTIME-complete [Lutz, 2003]. The three NEXPTIME lower bounds are obtained by using suitable and admissible concrete domains D_1, D_2, and D_3 to reduce the above domino problem. More precisely, the simplest concrete domain D_1 is used in the reduction to $\mathcal{ALCK}(\mathsf{D}_1)$-concept satisfiability w.r.t. Boolean (n-ary) key boxes, the slightly more complex D_2 is used in the reduction to $\mathcal{ALCK}(\mathsf{D}_2)$-concept satisfiability w.r.t. Boolean *unary* key boxes, and the most powerful concrete domain D_3 is used in the reduction to $\mathcal{ALCO}(\mathsf{D}_3)$-concept satisfiability without key boxes.

The idea underlying all three reductions is to use concept names $X_0, \ldots X_n, Y_0, \ldots, Y_n$ to represent positions in the $2^{n+1} \times 2^{n+1}$-torus: if a is a domain element representing the position (i, j), then $a \in X_\ell^\mathcal{I}$ expresses that the ℓ-th bit in the binary coding of i is 1, and $a \in Y_\ell^\mathcal{I}$ expresses that the ℓ-th bit of j is 1. We use standard methods to enforce that there exists a domain element for every position in the torus. The main difference between the three reductions is how it is ensured that no position is represented by two *different* domain elements—we call this *uniqueness of positions*.

The first reduction uses the very simple concrete domain D_1, which is based on the set $\{0, 1\}$ and only provides unary predicates $=_0$, $=_1$, and their negations. Uniqueness of positions is ensured by translating the position (i, j) of a domain element a into concrete domain values: for $\mathsf{xpos}_\ell \in \mathsf{N_{cF}}$, we enforce that $\mathsf{xpos}_\ell^\mathcal{I}(a) = 1$ if $a \in X_\ell^\mathcal{I}$ and 0 otherwise (analogously for ypos_ℓ and Y_ℓ). Then the key definition $(\mathsf{xpos}_0, \ldots, \mathsf{xpos}_n, \mathsf{ypos}_0, \ldots, \mathsf{ypos}_n \text{ keyfor } \top)$ obviously ensures uniqueness of positions. Since the reduction concept is path-free and D_1-satisfiability is easily seen to be in PTIME, we obtain the following:

Theorem 8. *D_1-satisfiability is in* PTIME *and satisfiability of path-free $\mathcal{ALCK}(\mathsf{D}_1)$-concepts w.r.t. simple key boxes is* NEXPTIME*-hard.*

The (somewhat artificial) concrete domain D_1 can be replaced by many natural concrete domains $\mathcal{D}$ proposed in the literature [Baader and Hanschke, 1992; Haarslev and Möller, 2002; Lutz, 2002b; 2002d]: it suffices that $\mathcal{D}$ provides two unary predicates denoting disjoint singleton sets.

The second reduction uses the more complex concrete domain D_2, which "stores" whole bit vectors rather than only single bits. In D_2, we can translate the position (i, j) of an element a from concepts X_ℓ, Y_k into a *single* bit vector of length $2(n + 1)$ that is then stored as a bv-successor of a, where bv is a concrete feature. Since we replaced the $2(n+1)$ concrete features used in the first reduction (one for each bit) by the single feature bv, it now suffices to use the simple unary key box $(\mathsf{bv} \text{ keyfor } \top)$ to ensure uniqueness of positions. As in D_1, the reduction concept is path-free. In [Lutz *et al.*, 2002], it is shown that D_2-satisfiability is in PTIME.

Theorem 9. *D_2-satisfiability is in* PTIME *and satisfiability of path-free $\mathcal{ALCK}(\mathsf{D}_2)$-concepts w.r.t. simple unary key boxes is* NEXPTIME*-hard.*

Again, the artificial concrete domain D_2 can be replaced by more natural ones: we can simulate bit vectors using integers and the necessary operations on bit vectors by unary predicates $=_n$ for every interger n and a ternary addition predicate—for more details see [Lutz *et al.*, 2002].

The last lower bound is concerned with the DL $\mathcal{ALCO}(\mathcal{D})$. In the absence of key boxes, we need a different reduction strategy and the more complex concrete domain D_3, which extends D_2 with so-called domino arrays that allow us to store the tiling of the *whole* torus in a single concrete domain value. We can then ensure uniqueness of positions using a *single* nominal. Computationally, the concrete domain D_3 is still very simple, namely in PTIME. However, it no longer suffices to use only path-free concepts.

Theorem 10. *D_3-satisfiability is in* PTIME *and satisfiability of $\mathcal{ALCO}(\mathsf{D}_3)$-concepts is* NEXPTIME*-hard.*

4 Reasoning Procedures

We describe two tableau-based decision procedures for concept satisfiability in DLs with concrete domains, nominals, and keys. The first is for $\mathcal{ALCOK}(\mathcal{D})$-concepts w.r.t. Boolean key boxes. This algorithm yields a NEXPTIME upper complexity bound matching the lower bounds established in Section 3. The second procedure is for $\mathcal{SHOQK}(\mathcal{D})$ w.r.t. path-free key boxes and also yields a tight NEXPTIME upper complexity bound. $\mathcal{SHOQK}(\mathcal{D})$ is an extension of the DL $\mathcal{SHOQ}(\mathcal{D})$ introduced in [Horrocks and Sattler, 2001;

Pan and Horrocks, 2002], which provides a wealth of expressive possibilities such as transitive roles, role hierarchies, nominals, qualifying number restrictions, and general TBoxes with a path-free concrete domain constructor and path-free key boxes. Path-freeness of $\mathcal{SHOQK}(\mathcal{D})$'s concrete domain constructor is crucial for decidability. Moreover, it allows us to admit general rather than only 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: starting with an initial data structure induced by the input concept, the algorithm repeatedly applies completion rules. Eventually, the algorithm either finds an obvious contradiction or it encounters a contradiction-free situation in which no more completion rules are applicable. In the former case the input concept is unsatisfiable, while in the latter case it is satisfiable.

Existing tableau algorithms for DLs with concrete domains use admissibility as an "interface" between the tableau algorithm and a concrete domain reasoner [Lutz, 2002b; Baader and Hanschke, 1991]. In the presence of keys, this is not enough: besides knowing whether a given $\mathcal{D}$-conjunction is satisfiable, the concrete domain reasoner has to provide information on variables that 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 identifies two of the variables v_1, v_2, 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 satisfiability of the input concept. E.g., this information transfer reveals the unsatisfiability of $\exists R.A \sqcap \exists R.(\neg A \sqcap B) \sqcap \exists R.(\neg A \sqcap \neg B) \sqcap \forall R.\exists g.<_2$ w.r.t. g keyfor $\top$. To formalize this requirement, we strengthen the notion of admissibility into *key-admissibility*.

Definition 11. A concrete domain $\mathcal{D}$ is *key-admissible* iff (i) $\Phi_\mathcal{D}$ contains a name $\top_\mathcal{D}$ for $\Delta_\mathcal{D}$; (ii) $\Phi_\mathcal{D}$ is closed under negation, and (iii) 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 $v, v' \in V$ $\delta(v) = \delta(v')$ iff $v \sim v'$. Such an equivalence relation is henceforth called a *concrete equivalence*. We say that *extended $\mathcal{D}$-satisfiability is in NP* if there exists an algorithm as above running in polynomial time. ♢

It can easily be seen that any concrete domain that is admissible and provides for an equality predicate is also key-admissible [Lutz *et al.*, 2002].

In the following, we assume that all concepts (the input concept and those occuring in key boxes) are in *negation normal form (NNF)*, i.e., negation occurs only in front of concept names and nominals; if the concrete domain $\mathcal{D}$ is admissible, then every $\mathcal{ALCOK}(\mathcal{D})$-concept can be converted into an equivalent one in NNF [Lutz *et al.*, 2002]. We use $\dot\neg C$ to denote the result of converting the concept $\neg C$ into NNF, $\mathsf{sub}(C)$ to denote the set of subconcepts of C, and $\mathsf{sub}(\mathcal{K})$ to denote the set of subconcepts of all concepts occurring in key box $\mathcal{K}$. Moreover, we use $\mathsf{cl}(C, \mathcal{K})$ as abbreviation for the set

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

Complexity of $\mathcal{ALCOK}(\mathcal{D})$

We start the presentation of the $\mathcal{ALCOK}(\mathcal{D})$ tableau algorithm by introducing the underlying data structure.

Definition 12. 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 $(V_\mathsf{a}, V_\mathsf{c}, E, \mathcal{L})$ with a set of nodes $V_\mathsf{a} \cup V_\mathsf{c}$ such that $V_\mathsf{a} \subseteq \mathsf{O_a}$, $V_\mathsf{c} \subseteq \mathsf{O_c}$, and all nodes from V_c are leaves. Each node $a \in V_\mathsf{a}$ of the tree is labeled with a subset $\mathcal{L}(a)$ of $\mathsf{cl}(C, \mathcal{K})$; each edge $(a, b) \in E$ with $a, b \in V_\mathsf{a}$ is labeled with a role name $\mathcal{L}(a, b)$ occurring in C or $\mathcal{K}$; and each edge $(a, x) \in E$ with $a \in V_\mathsf{a}$ and $x \in V_\mathsf{c}$ is labeled with a concrete feature $\mathcal{L}(a, x)$ occurring in C or $\mathcal{K}$.

For $\mathbf{T} = (V_\mathsf{a}, V_\mathsf{c}, E, \mathcal{L})$ and $a \in V_\mathsf{a}$, we use $\mathsf{lev}_\mathbf{T}(a)$ to denote the depth at which a occurs in $\mathbf{T}$ (starting with the root node at 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 $\mathbf{T} = (V_\mathsf{a}, V_\mathsf{c}, E, \mathcal{L})$ is a completion tree for C and $\mathcal{K}$, $\mathcal{P}$ is a function mapping each $P \in \Phi_\mathcal{D}$ with arity n appearing 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 $\mathbf{T} = (V_\mathsf{a}, V_\mathsf{c}, E, \mathcal{L})$ be a completion tree. A node $b \in V_\mathsf{a}$ is an *R-successor* of a node $a \in V_\mathsf{a}$ if $(a, b) \in E$ and $\mathcal{L}(a, b) = R$. Similarly, a node $x \in V_\mathsf{c}$ is a *g-successor* of a if $(a, x) \in E$ and $\mathcal{L}(a, x) = g$. For paths u, the notion *u-successor* is defined in the obvious way. ♢

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 the equivalence relation $\approx_\mathsf{c} \supseteq \sim$ on concrete nodes.

Definition 13. Let $S = (\mathbf{T}, \mathcal{P}, \prec, \sim)$ be a completion system for a concept C and a key box $\mathcal{K}$ with $\mathbf{T} = (V_\mathsf{a}, V_\mathsf{c}, E, \mathcal{L})$, and let $\approx$ be an equivalence relation on V_a. For each $R \in \mathsf{N_R}$, a node $b \in V_\mathsf{a}$ is an *$R/{\approx}$-neighbor* of a node $a \in V_\mathsf{a}$ if there exists a node $c \in V_\mathsf{a}$ such that $a \approx c$ and b is an R-successor of c. For paths u, the notion *$u/{\approx}$-neighbor* is defined analogously.

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

$$\begin{aligned}
\approx_\mathsf{a}^0 = &\{(a,a) \mid a \in V_\mathsf{a}\} \\
&\cup \{(a,b) \in V_\mathsf{a}^2 \mid \exists N \in \mathsf{N_O} : N \in \mathcal{L}(a) \cap \mathcal{L}(b)\} \\
\approx_\mathsf{a}^{i+1} = &\approx_\mathsf{a}^i \cup \\
&\{(a,b) \in V_\mathsf{a}^2 \mid \exists c \in \mathsf{V_a},\ f \in \mathsf{N_{aF}} : \\
&\qquad a \text{ and } b \text{ are } f/{\approx_\mathsf{a}^i}\text{-neighbors of } c\} \cup \\
&\{(a,b) \in V_\mathsf{a}^2 \mid \exists (u_1, \ldots, u_n \text{ keyfor } D) \in \mathcal{K}, \\
&\qquad \exists x_1, \ldots, x_n : x_j \text{ is } u_j/{\approx_\mathsf{a}^i}\text{-neighbor of } a, \\
&\qquad \exists y_1, \ldots, y_n : y_j \text{ is } u_j/{\approx_\mathsf{a}^i}\text{-neighbor of } b, \\
&\qquad D \in \mathcal{L}(a) \cap \mathcal{L}(b) \wedge (\forall 1 \leq j \leq n : x_j \sim y_j)\}
\end{aligned}$$

Finally, set $\approx_{\mathsf{a}} = \bigcup_{i \geq 0} \approx^i_{\mathsf{a}}$, and define $x \approx_{\mathsf{c}} y$ if $x \sim y$ or there are $a \in V_{\mathsf{a}}$ and $g \in \mathsf{N_{cF}}$ such that x and y are $g/{\approx_{\mathsf{a}}}$-neighbors of a. $\Diamond$

Intuitively, if we have $a \approx_{\mathsf{a}} b$, then a and b describe the same domain element of the constructed model (and similarly for the $\approx_{\mathsf{c}}$ relation on concrete nodes).

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 system* $S_{C_0} = (\mathbf{T}_{C_0}, \mathcal{P}_\emptyset, \emptyset, \emptyset)$, where $\mathbf{T}_{C_0} = (\{a_0\}, \emptyset, \emptyset, \{a_0 \mapsto \{C_0\}\})$ and $\mathcal{P}_\emptyset$ 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 14. Let $S = (\mathbf{T}, \mathcal{P}, \prec, \sim)$ be a completion system with $\mathbf{T} = (V_{\mathsf{a}}, V_{\mathsf{c}}, E, \mathcal{L})$. An element of $\mathsf{O_a}$ or $\mathsf{O_c}$ is called *fresh* in $\mathbf{T}$ if it does not appear in $\mathbf{T}$. We use the following notions:

$\mathbf{S + aRb}$**:** Let $a \in V_{\mathsf{a}}$, $b \in \mathsf{O_a}$ 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 (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 $\mathsf{lev}_\mathbf{T}(b) \leq \mathsf{lev}_\mathbf{T}(c)$.

$\mathbf{S + agx}$**:** Let $a \in V_{\mathsf{a}}$, $x \in \mathsf{O_c}$ 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 (a, x) to E and setting $\mathcal{L}(a, x) = g$.

When nesting $+$, 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 $a \in V_{\mathsf{a}}$ and $x \in \mathsf{O_c}$ is fresh in $\mathbf{T}$, we denote the completion system S' that can be obtained from S by taking fresh nodes $b_1, \ldots, b_n \in \mathsf{O_a}$ and setting

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

The completion rules are given in Figure 1, where we assume that newly introduced nodes are always fresh. The $\mathsf{R}\sqcup$ and Rch rules are non-deterministic and the upper five rules are well-known from existing tableau algorithms for $\mathcal{ALC}(\mathcal{D})$-concept satisfiability (c.f. for example [Lutz, 2002d]). Only $\mathsf{R}\forall$ deserves a comment: it considers $R/{\approx_{\mathsf{a}}}$-neighbors rather than R-successors since $\approx_{\mathsf{a}}$ relates nodes denoting the same domain element.

The last two rules are necessary for dealing with key boxes. The "choose rule" Rch (c.f. [Hollunder and Baader, 1991; Horrocks *et al.*, 2000]) guesses whether an abstract node a satisfies C in case of C occurring in a key definition and a having neighbors for all paths u_i in this key definition. The Rp rule deals with equalities between abstract nodes as recorded by the $\approx_{\mathsf{a}}$ relation: if $a \approx_{\mathsf{a}} b$, then a and b describe the same element, and thus their node labels should be identical. We choose one representative for each equivalence class of $\approx_{\mathsf{a}}$ (the node that is minimal w.r.t. $\prec$) and make sure that the representative's node label contains the labels of all the nodes it represents.

Definition 15. Let $S = (\mathbf{T}, \mathcal{P}, \prec, \sim)$ be a completion system for a concept C and a key box $\mathcal{K}$ with $\mathbf{T} = (V_{\mathsf{a}}, V_{\mathsf{c}}, \prec, \sim)$. We say that the completion system S is *concrete domain satisfiable* iff the conjunction

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

is satisfiable. S contains a *clash* iff (i) there is an $a \in V_{\mathsf{a}}$ and an $A \in \mathsf{N_C}$ such that $\{A, \neg A\} \subseteq \mathcal{L}(a)$; (ii) there are $a \in V_{\mathsf{a}}$ and $x \in V_{\mathsf{c}}$ such that $g\uparrow \in \mathcal{L}(a)$ and x is $g/{\approx_{\mathsf{a}}}$-neighbor of a; or (iii) S is not concrete domain satisfiable. If S does not contain a clash, then S is called *clash-free*. S is *complete* if no completion rule is applicable to S. $\Diamond$

We now give the tableau algorithm in pseudocode notation, where check denotes the algorithm computing concrete equivalences as described in Definition 11:

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

The algorithm realizes 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. The concrete domain reasoner may then return in $\mathsf{check}(\zeta_S)$ further "equalities" $\sim$ and so forth.

A similar interplay takes place in the course of several recursion steps: equalities of concrete nodes provided by the

$\mathsf{R}\sqcap$ if $C_1 \sqcap C_2 \in \mathcal{L}(a)$ and $\{C_1, C_2\} \not\subseteq \mathcal{L}(a)$
then $\mathcal{L}(a) := \mathcal{L}(a) \cup \{C_1, C_2\}$

$\mathsf{R}\sqcup$ 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 $C \in \{C_1, C_2\}$

$\mathsf{R}\exists$ if $\exists R.C \in \mathcal{L}(a)$ and there is no $R/\approx_a$-neighbor b of a such that $C \in \mathcal{L}(b)$,
then $S := S + aRb$ and $\mathcal{L}(b) := \{C\}$

$\mathsf{R}\forall$ 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 $\mathcal{L}(b) := \mathcal{L}(b) \cup \{C\}$

$\mathsf{R}\exists\mathsf{c}$ if $\exists u_1, \ldots, u_n.P \in \mathcal{L}(a)$ and there are no u_i-successors x_i of a (for $1 \leq i \leq n$) with $(x_1, \ldots, x_n) \in \mathcal{P}(P)$
then $S := (S + au_1x_1 + \cdots + au_nx_n)$ and $\mathcal{P}(P) := \mathcal{P}(P) \cup \{(x_1, \ldots, x_n)\}$

Rch if $(u_1, \ldots, u_n \text{ keyfor } C) \in \mathcal{K}$, $\{C, \dot\neg C\} \cap \mathcal{L}(a) = \emptyset$, and there are $u_i/{\approx_{\mathsf{a}}}$-neighbors x_i of a (for $1 \leq i \leq n$)
then $\mathcal{L}(a) := \mathcal{L}(a) \cup \{D\}$ for some $D \in \{C, \dot\neg C\}$

Rp if $\mathcal{L}(b) \not\subseteq \mathcal{L}(a)$ and $a \in V_{\mathsf{a}}$ is minimal w.r.t. $\prec$ such that $a \approx_{\mathsf{a}} b$
then $\mathcal{L}(a) := \mathcal{L}(a) \cup \mathcal{L}(b)$

Figure 1: Completion rules for $\mathcal{ALCOK}(\mathcal{D})$.

concrete domain reasoner may make new rules applicable (for example Rp and then R∃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. Note that, in the absence of keys boxes, there is much less interaction: it suffices to apply the concrete domain satisfiability check only once after the completion rules have been exhaustively applied [Baader and Hanschke, 1991].

In [Lutz *et al.*, 2002], we prove that the algorithm runs in non-deterministic exponential time: there are exponential bounds on the number of abstract and concrete nodes in the completion system, on the number of while loop iterations in each recursion step, and on the size of ζ_S. This yields the following upper bound, which is tight by Theorem 9.

Theorem 16. *For $\mathcal{D}$ a key-admissible concrete domain such that extended $\mathcal{D}$-satisfiability is in* NP*, $\mathcal{ALCOK}(\mathcal{D})$-concept satisfiability w.r.t. Boolean key boxes is in* NEXPTIME*.*

Complexity of $\mathcal{SHOQK}(\mathcal{D})$

We have designed a tableau algorithm for $\mathcal{SHOQK}(\mathcal{D})$ as a combination of the one for $\mathcal{SHOQ}(\mathcal{D})$ in [Horrocks and Sattler, 2001] and the one for $\mathcal{ALCOK}(\mathcal{D})$ presented above. It is restricted to path-free concepts and path-free key boxes, but can handle complex concepts in key boxes. The most important difference from the $\mathcal{ALCOK}(\mathcal{D})$ algorithm is as follows: in the presence of non-Boolean key boxes, the Rch rule may add concepts of positive "role depth" to arbitrary nodes in the completion tree. Thus the role depth does not automatically decrease with the depth of nodes in the tree (as in the case of $\mathcal{ALCOK}(\mathcal{D})$) and a naive tableau algorithm would construct infinite trees. However, even for $\mathcal{SHOQ}(\mathcal{D})$ without key boxes, one has to enforce termination artificially by using a cycle detection mechanism called *blocking*—whereas the $\mathcal{ALCOK}(\mathcal{D})$ algorithm terminates "naturally". It can be shown that blocking can be used in the presence of key boxes without corrupting soundness or completeness. A detailed description of this algorithm and a correctness proof is given in [Lutz *et al.*, 2002]. As a by-product of the $\mathcal{SHOQK}(\mathcal{D})$ tableau algorithm, we obtain a *small model property*: every satisfiable $\mathcal{SHOQK}(\mathcal{D})$-concept has a model of size exponential in the concept length. Thus we obtain the following upper bound, which is tight by Theorem 9.

Theorem 17. *For $\mathcal{D}$ a key-admissible concrete domain such that $\mathcal{D}$-satisfiability is in* NP*, $\mathcal{SHOQK}(\mathcal{D})$-concept satisfiability w.r.t. path-free key boxes is in* NEXPTIME*.*

5 Summary

We have identified key boxes as an interesting extension of description logics with concrete domains, introduced a number of natural description logics, and provided a comprehensive analysis of the decidability and complexity of reasoning. Moreover, we have proposed tableau algorithms for two such (NEXPTIME-complete) logics.

The main result of our investigations is that key constraints are rather powerful, since they dramatically increase the complexity of reasoning: PSPACE $\mathcal{ALC}(\mathcal{D})$ becomes undecidable with unrestricted key boxes, and NEXPTIME-complete with Boolean key boxes—provided that the concrete domain $\mathcal{D}$ is not too complex, i.e., extended $\mathcal{D}$-satisfiability is in NP.

References

[Baader and Hanschke, 1991] F. Baader and P. Hanschke. A scheme for integrating concrete domains into concept languages. In *Proc. of IJCAI-91*, Morgan-Kaufmann, 1991.

[Baader and Hanschke, 1992] F. Baader and P. Hanschke. Extensions of concept languages for a mechanical engineering application. In *Proc. of GWAI-92*, vol. 671 of LNCS, Springer, 1992.

[Borgida and Weddell, 1997] A. Borgida and G.E. Weddell. Adding uniqueness constraints to description logics (preliminary report). In *Proc. of DOOD97*, volume 1341 of *LNCS*, pages 85–102. Springer, 1997.

[Calvanese *et al.*, 2000] D. Calvanese, G. De Giacomo, and M. Lenzerini. Keys for Free in Description Logics. In *Proc. of DL2000*, number 33 in http://CEUR-WS.org/, 2000.

[Haarslev and Möller, 2002] V. Haarslev and R. Möller. Practical reasoning in racer with a concrete domain for linear inequations. In *Proc. of DL2002*, number 53 in http://CEUR-WS.org/, 2002.

[Hollunder and Baader, 1991] B. Hollunder and F. Baader. Qualifying number restrictions in concept languages. In *Proc. of KR'91*, Morgan-Kaufmann, 1991.

[Horrocks and Sattler, 2001] I. Horrocks and U. Sattler. Ontology reasoning in the $\mathcal{SHOQ}(D)$ description logic. In *Proc. of IJCAI-01*, Morgan-Kaufmann, 2001.

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

[Horrocks *et al.*, 2002] I. Horrocks, P.F. Patel-Schneider, and F. van Harmelen. Reviewing the design of DAML+OIL: An ontology language for the semantic web. In *Proc. of AAAI 2002*, 2002.

[Khizder *et al.*, 2001] V. Khizder, D. Toman, and G. Weddell. On Decidability and Complexity of Description Logics with Uniqueness Constraints. In *Proc. of ICDT 2001*, 2001.

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

[Lutz *et al.*, 2002] C. Lutz, C. Areces, I. Horrocks, and U. Sattler. Keys, nominals, and concrete domains. LTCS-Report 02-04, Technical University Dresden, 2002. See http://lat.inf.tu-dresden.de/research/reports.html.

[Lutz, 2002b] C. Lutz. Description logics with concrete domains—a survey. In *Proc. of AiML2002*, 2002.

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

[Lutz, 2002e] C. Lutz. Reasoning about entity relationship diagrams with complex attribute dependencies. In *Proc. of DL2002*, number 53 in http://CEUR-WS.org/, 2002.

[Lutz, 2003] C. Lutz. NExpTime-complete description logics with concrete domains. *ACM Transactions on Computational Logic*, 2003. To appear.

[Pan and Horrocks, 2002] J. Z. Pan and I. Horrocks. Reasoning in the $\mathcal{SHOQ}(\mathcal{D}_n)$ description logic. In *Proc. of DL2002*, number 53 in http://CEUR-WS.org/, 2002.

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

[Toman and Weddell, 2002] D. Toman and G. Weddell. On reasoning about structural equality in XML: A description logic approach. In *Proc. of ICDT2003*, number 2572 in LNCS, pages 96–110, Springer, 2002.