Keys, Nominals, and Concrete Domains

Carsten Lutz\textsuperscript{1} Carlos Areces\textsuperscript{2} Ian Horrocks\textsuperscript{3} Ulrike Sattler\textsuperscript{1}

\textsuperscript{1}Inst. for Theoretical Computer Science
Technical University Dresden
Germany
(lastname)@tcs.inf.tu-dresden.de

\textsuperscript{2}INRIA Lorraine
Nancy France
Carlos.Areces@loria.fr

\textsuperscript{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., for 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 key for Human), then the concept “Human with SSN 1234” (Human \( \sqcap \exists \text{SSN } = 1234 \)) has at most one instance.

In this paper, we extend the well-known DLs with concrete domains \( \mathcal{ALC}(D) \) and \( \mathcal{SHOQ}(D) \) [Baader and Hanschke, 1991; Horrocks and Sattler, 2001] with key constraints and analyse the complexity of reasoning with the resulting logics \( \mathcal{ALCOK}(D) \) and \( \mathcal{SHOQK}(D) \). We show that allowing complex concepts to occur in key constraints dramatically increases the complexity of \( \mathcal{ALC}(D) \) (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 \( \mathcal{ALC}(D) \) is not robust [Lutz, 2003].

Additionally, we prove the NEXPTIME bounds to be tight by presenting tableau algorithms for \( \mathcal{ALCOK}(D) \) and \( \mathcal{SHOQK}(D) \) 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 \( \mathcal{ALCOK}(D) \).

Definition 1. A concrete domain \( D \) is a pair \((\Delta_D, \Phi_D)\), where \( \Delta_D \) is a set and \( \Phi_D \) a set of predicate names. Each predicate name \( P \in \Phi_D \) is associated with an arity \( n \) and an \( n \)-ary predicate \( P^D \subseteq \Delta_D^n \).

Let \( N_C, N_O, N_R, N_F \) be pairwise disjoint and countably infinite sets of concept names, nominals, role names, and concrete features. We assume that \( N_R \) has a countably infinite subset \( N_F \) of abstract features. A path \( u \) is a composition \( f_1 \ldots f_n g \) of \( n \) abstract features \( f_1, \ldots, f_n \) (\( n \geq 0 \)) and a concrete feature \( g \). Let \( D \) be a concrete domain. The set of \( \mathcal{ALCOK}(D) \)-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 \( P \in \Phi_D \) is a predicate of arity \( n \),

...
then the following expressions are also concepts:
\[ \neg C, \ C \cap D, \ C \cup D, \ \exists R. C, \ \forall R. C, \ \exists u_1, \ldots, u_n, P, \text{ and } g^+. \]

A key definition is an expression \((u_1, \ldots, u_k)\) key for \(C\) 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.

As usual, we use \(\mathcal{T}\) to denote an arbitrary propositional tautology. Throughout this paper, we will consider several fragments of the logic \(ALCO(\mathcal{D})\): \(ALCO(\mathcal{D})\) is obtained from \(ALCO(\mathcal{D})\) by admitting only empty key boxes; by disallowing the use of nominals, we obtain the fragment \(ALC(\mathcal{D})\) of \(ALCO(\mathcal{D})\) and \(ALCK(\mathcal{D})\) of \(ALCO(\mathcal{D})\).

The description logic \(ALCO(\mathcal{D})\) 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 \((\Delta_T, \mathcal{I})\), where \(\Delta_T\) is a non-empty set, called the domain, and \(\mathcal{I}\) is the interpretation function. The interpretation function maps each concept name \(C\) to a subset \(C^\mathcal{I}\) of \(\Delta_T\), each nominal \(N\) to a singleton subset \(N^\mathcal{I}\) of \(\Delta_T\), each role name \(R\) to a subset \(R^\mathcal{I}\) of \(\Delta_T \times \Delta_T\), each abstract feature \(f\) to a partial function \(f^\mathcal{I}\) from \(\Delta_T\) to \(\Delta_T\), and each concrete feature \(g\) to a partial function \(g^\mathcal{I}\) from \(\Delta_T\) to \(\Delta_T\).

If \(u = f_1 \ldots f_n g\) is a path and \(d \in \Delta_T\), then \(u^\mathcal{I}(d)\) is defined as \(g^\mathcal{I}(f_1^\mathcal{I}(d), \ldots, f_n^\mathcal{I}(d))\). The interpretation function is extended to arbitrary concepts as follows:

\[
\begin{align*}
(\neg C)^\mathcal{I} & := \Delta_T \setminus C^\mathcal{I} \\
(C \cap D)^\mathcal{I} & := C^\mathcal{I} \cap D^\mathcal{I} \\
(C \cup D)^\mathcal{I} & := C^\mathcal{I} \cup D^\mathcal{I} \\
(\exists R.C)^\mathcal{I} & := \{d \in \Delta_T \mid \exists e \in \Delta_T : (d, e) \in R^\mathcal{I} \land e \in C^\mathcal{I}\} \\
(\forall R.C)^\mathcal{I} & := \{d \in \Delta_T \mid \forall e \in \Delta_T : (d, e) \in R^\mathcal{I} \rightarrow e \in C^\mathcal{I}\} \\
(g^+) & := \{d \in \Delta_T : g^\mathcal{I}(d) \text{ undefined}\} \\
(\exists u_1, \ldots, u_n, P)^\mathcal{I} & := \{d \in \Delta_T \mid \exists x_1, \ldots, x_n \in \Delta_P : u_1^\mathcal{I}(d) = x_1, \ldots, u_n^\mathcal{I}(d) = x_n\} \subseteq P^\mathcal{I}
\end{align*}
\]

An interpretation \(\mathcal{I}\) is a model of a concept \(C\) iff \(C^\mathcal{I} \neq \emptyset\). Moreover, \(\mathcal{I}\) satisfies a key definition \((u_1, \ldots, u_n, \text{key} for C)\) 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 key box \(D\) w.r.t. \(\mathcal{K}\) (written \(C \sqsubseteq_K D\)) iff \(C^\mathcal{I} \subseteq D^\mathcal{I}\) for all models \(\mathcal{I}\) of \(\mathcal{K}\).

It is well-known that, in DLs providing for negation, subsumption can be reduced to (un)satisfiability and vice versa: \(C \sqsubseteq_K D\) iff \(C \cap \neg D\) is unsatisfiable w.r.t. \(\mathcal{K}\) and \(C\) is satisfiable w.r.t. \(\mathcal{K}\) iff \(C \not\sqsubseteq_K \neg \mathcal{T}\). 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 \(D\)-conjunction is a (finite) predicate conjunction of the form

\[
c = \bigwedge_{i \leq k} P_i(x_0^{(i)}, \ldots, x_n^{(i)}),
\]

where \(P_i\) is an \(n_i\)-ary predicate for \(i < k\) and the \(x_j^{(i)}\) are variables. A \(D\)-conjunction \(c\) is satisfiable iff there exists a function \(\delta\) mapping the variables in \(c\) to elements of \(\Delta_T\) such that \((\delta(x_0^{(0)}), \ldots, \delta(x_{n_1}^{(0)})) \in P_1^\mathcal{D}\) for each \(i < k\). We say that the concrete domain \(\mathcal{D}\) is admissible iff (i) \(\Phi_P\) contains a name \(\top_P\) for \(\mathcal{D}\); (ii) \(\Phi_P\) is closed under negation, and (iii) satisfiability of \(D\)-conjunctions is decidable. We refer to the satisfiability of \(D\)-conjunctions as \(D\)-satisfiability.

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 \((g_1, \ldots, g_n)\) key for \(C\) with \(g_1, \ldots, g_n \in N_{FL}\); 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 \((u)\) key for \(C\). 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 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 \(ALCK(\mathcal{D})\)-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 \(ALCK(\mathcal{D})\)-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 \(ALCO(\mathcal{D})\)-concept satisfiability (without key boxes) is already NEXP-TIME-hard.

Undecidability of \(ALCK(D)\)-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 \((x_1, r_1), \ldots, (x_k, r_k)\) of pairs of words over some alphabet \(\Sigma\). A sequence of integers \(i_1, \ldots, i_m\), with \(m \geq 1\), is called a solution for \(P\) iff \(i_1, \ldots, i_m = r_1, \ldots, r_m\). The problem is to decide whether a given instance \(P\) has a solution.

The reduction uses the admissible concrete domain \(W\) introduced in [Lutz, 2003], whose domain is the set of words over \(\Sigma\) and whose predicates express concatenation of words. For each PCP instance \(P = ((x_1, r_1), \ldots, (x_k, r_k))\), we define a
concept $C_P$ and unary key box $K_P$ such that $P$ has no solution iff $C_P$ is satisfiable w.r.t. $K_P$. Intuitively, $C_P$ and $K_P$ enforce an infinite, $k$-ary tree, where each node represents a sequence 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 $D$ such that $D$-satisfiability is in PTIME and satisfiability of $\text{ALC}^\text{K}(\mathcal{D})$-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 $\text{ALC}^\text{K}(\mathcal{D})$ with Boolean key boxes and for $\text{ALC}^\text{CO}(\mathcal{D})$. These results are obtained using a NEXPTIME-complete variant of the well-known, undecidable domino problem [Knuth, 1968].

**Definition 7.** A domino system $\mathcal{D}$ is a triple $(T, H, V)$, where $T \subseteq \mathbb{N}$ is a finite set of tile types and $H, V \subseteq T \times T$ represent the horizontal and vertical matching conditions. For $\mathcal{D}$ a domino system and $a = a_0, \ldots, a_{n-1} \in T^n$ an initial condition, a mapping $\tau : \{0, \ldots, 2^{n+1}\} \times \{0, \ldots, 2^{n+1}\} \rightarrow T$ is a solution for $\mathcal{D}$ and $a$ if, 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 $i < n$.

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 $\text{ALC}^\text{K}(\mathcal{D}_1)$-concept satisfiability w.r.t. Boolean ($n$-ary) key boxes, the slightly more complex $D_2$ is used in the reduction to $\text{ALC}^\text{K}(\mathcal{D}_2)$-concept satisfiability w.r.t. Boolean unary key boxes, and the most powerful concrete domain $D_3$ is used in the reduction to $\text{ALC}^\text{CO}(\mathcal{D}_3)$-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 $a \in X_i^j$ expresses that the $i$-th bit in the binary coding of $i$ is 1, and $a \in Y_i^j$ expresses that the $j$-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 $x_{ps_1} \in \mathcal{N}$, we enforce that $x_{ps_1}(a) = 1$ if $a \in X_i^j$ and 0 otherwise (analogously for $y_{ps_2}$ and $y_{ps_3}$). Then the key definition $(x_{ps_1}, \ldots, x_{ps_n}, y_{ps_1}, \ldots, y_{ps_n}, \text{key for } T)$ 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 $\text{ALC}^\text{K}(\mathcal{D}_1)$-concepts w.r.t. simple key boxes is NEXPTIME-hard.

The (somewhat artificial) concrete domain $D_2$ can be replaced by many natural concrete domains $D$ proposed in the literature [Baader and Hanschke, 1992; Haarslev and Möller, 2002; Lutz, 2002b; 2002d]: it suffices that $D$ 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 $X_i, Y_j$ 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 $\{\text{bv key for } T\}$ 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 $\text{ALC}^\text{K}(\mathcal{D}_2)$-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 $=_{n,1}$ for every integer $n$ and a ternary addition predicate—for more details see [Lutz et al., 2002].

The last lower bound is concerned with the DL $\text{ALC}^\text{CO}(\mathcal{D})$. 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 $\text{ALC}^\text{CO}(\mathcal{D}_3)$-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 $\text{ALC}^\text{K}(\mathcal{D})$-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 $\text{SHOIQ}^\text{K}(\mathcal{D})$ w.r.t. path-free key boxes and also yields a tight NEXPTIME upper complexity bound. $\text{SHOIQ}^\text{K}(\mathcal{D})$ is an extension of the DL $\text{SHOIQ}(\mathcal{D})$ 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 $SHQOK(D)$’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 $D$-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 $N = \langle \mathbb{N}, \{ \aleph_n \mid n \in \mathbb{N} \} \rangle$ and the $N$-conjunction $c = \leq_2(v_1) \land \leq_2(v_2) \land \leq_2(v_3)$. Obviously, every solution $\delta$ 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 $\exists R. A \land \exists R. (\neg A \land B) \land \exists R. (\neg A \land B) \land \exists R. \exists g. \leq_2 w.r.t. g$ key for $T$. To formalize this requirement, we strengthen the notion of admissibility into key-admissibility.

**Definition 11.** A concrete domain $D$ is key-admissible iff (i) $\Phi_D$ contains a name $\tau_D$ for $\Delta_D$; (ii) $\Phi_D$ is closed under negation, and (iii) there exists an algorithm that takes as input a $\Delta_D$-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 $\delta$ 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 relation. We say that extended $D$-satisfiability is in NP if there exists an algorithm as above running in polynomial time.\hfill\Box

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 occurring 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 $D$ is admissible, then every $ALCOK(D)$-concept can be converted into an equivalent one in NNF [Lutz et al., 2002]. We use $\neg C$ to denote the result of converting the concept $C$ into NNF, $\text{sub}(C)$ to denote the set of subconcepts of $C$, and $\text{sub}(K)$ to denote the set of subconcepts of all concepts occurring in key box $K$. Moreover, we use $\text{cl}(C, K)$ as abbreviation for the set $\text{sub}(C) \cup \text{sub}(K) \cup \{ \neg D \mid D \in \text{sub}(K) \}$.

**Complexity of $ALCOK(D)$**

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

**Definition 12.** Let $O_1$ and $O_2$ be disjoint and countably infinite sets of abstract and concrete nodes. A completion tree for an $ALCOK(D)$-concept $C$ and a key box $K$ is a finite, labeled tree $(V_C, V_C, E, L)$ with a set of nodes $V_C \cup V_C'$ such that $V_C \subseteq O_1$, $V_C' \subseteq O_2$, and all nodes from $V_C'$ are leaves. Each node $a \in V_C$ of the tree is labeled with a subset $\mathcal{L}(a)$ of $\text{cl}(C, K)$; each edge $(a, b) \in E$ with $a, b \in V_C$ is labeled with a role name $\mathcal{L}(a, b)$ occurring in $C$ or $K$; and each edge $(a, x) \in E$ with $a \in V_C$ and $x \in V_C'$ is labeled with a concrete feature $\mathcal{L}(a, x)$ occurring in $C$ or $K$.

For $T = (V_C, V_C, E, L)$ and $a \in V_C$, we use $\text{lev}_T(a)$ to denote the depth at which $a$ occurs in $T$ (starting with the root node at depth 0). A completion system for an $ALCOK(D)$-concept $C$ and a key box $K$ is a tuple $(T, P, \prec, \sim)$, where $T = (V_C, V_C, E, L)$ is a completion tree for $C$ and $K$. $P$ is a function mapping each $P \in \Phi_D$ with arity $n$ appearing in $C$ to a subset of $V_C^n$, $\prec$ is a linear ordering of $V_C$ such that $\text{lev}_T(a) \leq \text{lev}_T(b)$ implies $a \prec b$, and $\sim$ is an equivalence relation on $V_C$. Let $T = (V_C, V_C, E, L)$ be a completion tree. A node $b \in V_C$ in $T$ is an $R$-successor of a node $a \in V_C$ if $(a, b) \in E$ and $\mathcal{L}(a, b) = R$. Similarly, a node $x \in V_C$ 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.\hfill\Box

Intuitively, the relation $\sim$ records equalities between concrete nodes that have been found during the model construction process. The relation $\prec$ induces an equivalence relation $\approx_s$ on abstract nodes which, in turn, yields the equivalence relation $\approx_c \supseteq \sim$ on concrete nodes.

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

We define a sequence of equivalence relations $\approx_s^0 \subseteq \approx_s^1 \subseteq \cdots$ on $V_C$ as follows:

$$\approx_s^0 = \{(a, a) \mid a \in V_C\}$$

$$\approx_s^{n+1} = \approx_s^n \cup \{(a, b) \in V_C^2 \mid \exists e \in V_C, f \in N_{\Phi_D} : (a, b, f) \in E \land (a, f) \approx_s\text{-neighbors of } c\} \cup \{(a, b) \in V_C^2 \mid \exists u_1, \ldots, u_n \text{ key for } D \in K,\exists x_1, \ldots, x_n : x_j = u_j/\approx_s\text{-neighbor of } a, \exists y_1, \ldots, y_n : y_j = u_j/\approx_s\text{-neighbor of } b, D \in \mathcal{L}(a) \cap \mathcal{L}(b) \land (\forall 1 \leq j \leq n : x_j \sim y_j)\}$$
Finally, set \( \approx_3 = \bigcup_{i \geq 0} \approx_i \), and define \( x \approx_c y \) if \( x \sim y \) or there are \( a \in V_3 \) and \( g \in N_{ef} \) such that \( x \) and \( y \) are \( g/\approx_3 \)-neighbors of \( a \).

Intuitively, if we have \( a \approx_3 b \), then \( a \) and \( b \) describe the same domain element of the constructed model (and similarly for the \( \approx_c \) relation on concrete nodes).

Let \( \mathcal{D} \) be a key-admissible concrete domain. To decide the satisfiability of an \( \mathcal{ALCOK}(\mathcal{D}) \)-concept \( C_0 \), a Boolean key box \( \mathcal{K} \) (both in NNF), the tableau algorithm is started with the initial completion system \( S_{C_0} = (T_{C_0}, P_0, \emptyset, \emptyset) \), where \( T_{C_0} = \{ \{a_0\}, \emptyset, \emptyset, \{a_0 \mapsto \{C_0\}\} \} \) and \( P_0 \) 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 \( S = (T, P, \prec, \sim) \) be a completion system with \( T = (V_2, V_3, E, L) \). An element of \( O_a \) or \( O_c \) is called *fresh* in \( T \) if it does not appear in \( T \). We use the following notions:

- **S + aRb:** Let \( a \in V_3, b \in O_3 \) fresh in \( T \), and \( R \in N_R \). We write \( S + aRb \) to denote the completion system \( S' \) that can be obtained from \( S \) by adding \((a, b)\) to \( E \) and setting \( L(a, b) = R \) and \( L(b) = \emptyset \). Moreover, \( b \) is inserted into \( \prec \) such that \( b \prec c \) implies \( \text{lev}_T(b) \leq \text{lev}_T(c) \).

- **S + agx:** Let \( a \in V_3, x \in O_3 \) fresh in \( T \), and \( g \in N_{ef} \). We write \( S + agx \) to denote the completion system \( S' \) that can be obtained from \( S \) by adding \((a, x)\) to \( E \) and setting \( L(a, x) = g \).

When nesting +, we omit brackets writing, e.g., \( S + aR_1 b + bR_2 c \) for \( \sigma(S + aR_1 b + bR_2 c) \). Let \( u = f_1 \ldots f_n g \) be a path. With \( S + aux \), where \( a \in V_3 \) and \( x \in O_3 \) is fresh in \( T \), we denote the completion system \( S' \) that can be obtained from \( S \) by taking fresh nodes \( b_1, \ldots, b_n \in O_3 \) and setting \( S' := S + a f_1 b_1 + \cdots + a f_n b_n + b_n g x \).

The completion rules are given in Figure 1, where we assume that newly introduced nodes are always fresh. The \( R_l \) and \( R_h \) 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 \( R_y \) serves a comment: it considers \( R/\approx_3 \)-neighbors rather than \( R \)-successors since \( \approx_3 \) relates nodes denoting the same domain element.

The last two rules are necessary for dealing with key boxes. The “choose rule” \( R_h \) (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 \( R_p \) rule deals with equalities between abstract nodes as recorded by the \( \approx_3 \) relation: if \( a \approx_3 b \), 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 \( \approx_3 \) (the node that is minimal w.r.t. \( \sim \)) and make sure that the representative’s node label contains the labels of all the nodes it represents.

**Definition 15.** Let \( S = (T, \prec, \sim) \) be a completion system for a concept \( C \) and a key box \( \mathcal{K} \) with \( T = (V_2, V_3, \prec, \sim) \). We say that the completion system \( S \) is *concrete domain satisfiable* iff the conjunction

\[
\zeta_S = \bigwedge_{(x, y) \in \mathcal{C}(P)} P(x_1, \ldots, x_n) \wedge (x, y) = (x, y) \wedge x \approx_3 y
\]

is satisfiable. \( S \) contains a *clash* iff (i) there is an \( a \in V_3 \) and an \( A \in N_C \) such that \( \{A, \neg A\} \subseteq L(a) \); (ii) there are \( a \in V_3 \) and \( x \in V_2 \) such that \( y \uparrow C(a) \) and \( y \neq g/\approx_3 \)-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 \( S \).

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

**define procedure sat(S) do**

- if \( S \) contains a clash then return unsatisfiable
- \( \sim := \text{check}(\zeta_S) \)
- compute \( \sim \) and then \( \approx_c \)
- while \( \sim \neq \approx_c \) do
- if \( S \) contains a clash then return unsatisfiable
- if \( S \) is complete then return satisfiable
- apply a completion rule to \( S \) yielding \( S' \)
- \text{return 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 \( \approx_3 \) and \( \approx_c \) even more equalities between concrete nodes. The concrete domain reasoner may then return in 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
concrete domain reasoner may make new rules applicable (for example R_p and then R_{c}c) which changes P and thus also C_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 nondeterministic 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 C_S. This yields the following upper bound, which is tight by Theorem 9.

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

**Complexity of $\text{SHOQK}(D)$**

We have designed a tableau algorithm for $\text{SHOQK}(D)$ as a combination of the one for $\text{SHOQ}(D)$ in [Horrocks and Sattler, 2001] and the one for $\text{ALCOK}(D)$ 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 $\text{ALCOK}(D)$ 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 $\text{ALCOK}(D)$) and a naive tableau algorithm would construct infinite trees. However, even for $\text{SHOQK}(D)$ without key boxes, one has to enforce termination artificially by using a cycle detection mechanism called blocking—whereas the $\text{ALCOK}(D)$ 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 $\text{SHOQK}(D)$ tableau algorithm, we obtain a small model property: every satisfiable $\text{SHOQK}(D)$-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 D a key-admissible concrete domain such that D-satisfiability is in NP, $\text{SHOQK}(D)$-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 $\text{ALC}(D)$ becomes undecidable with unrestricted key boxes, and NEXPTIME-complete with Boolean key boxes—provided that the concrete domain D is not too complex, i.e., extended D-satisfiability is in NP.

**References**


