Technical University Dresden
Department of Computer Science

Master's Thesis on

Description Logics with Concrete Domains and Functional Dependencies

by

Maja Miličić

born on 9th May 1978 in Belgrade

Advisor: Dr. Carsten Lutz

Supervising Professor: Prof. Dr. Franz Baader

Dresden, February 2004

Abstract

Description Logics (DLs) are a family of logic-based knowledge representation formalisms designed to represent and reason about conceptual knowledge. Due to a nice compromise between expressivity and the complexity of reasoning, DLs have found applications in many areas such as, e.g., modelling database schemas and the semantic web. However, description logics represent knowledge in an abstract way and lack the power to describe more concrete (quantitative) qualities like size, duration, or amounts. The standard solution is to equip DLs with concrete domains, e.g., natural numbers with predicates $=$, $<$, $+$ or strings with a string concatenation predicate. Moreover, recently it has been suggested that the expressive power of DLs with concrete domains can be further enhanced by providing them with database-like key constraints. Key constraints can be a source of additional inconsistencies in database schemas, and DLs applied in reasoning about database schemas are thus wanted to be able to capture such constraints. Up to now, only the integration of uniqueness key constraints into DLs with concrete domains has been investigated. In this thesis, we continue this investigation by considering another type of keys, called functional dependencies. Functional dependencies allow us to state that a property is uniquely determined by a set of properties, such as: "all books with the same ISBN number have the same title". We will focus on $\mathcal{ALC}$ (the basic propositionally closed DL) and $\mathcal{ALC}(\mathcal{D})$ ($\mathcal{ALC}$ equipped with an arbitrary concrete domain $\mathcal{D}$). We show that functional dependencies, just like uniqueness constraints, dramatically increase the complexity of reasoning in the PSpace-complete $\mathcal{ALC}(\mathcal{D})$: reasoning in $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$ ($\mathcal{ALC}(\mathcal{D})$ extended with functional dependencies) is undecidable in the general case, while NExpTime-complete in a slightly restricted version.

Contents

Acknowledgements

With the precious help of my advisor Carsten Lutz, I made my first steps into research in the field of description logics. I want to thank him for his great support, infinite patience, and for everything I learned from him about the scientific work.

Moreover, I would like to thank German Academic Exchange Office (DAAD) for the grant I received during the whole period of my studies in the International Masters Programme in Computational Logic at the TU Dresden. I am also very grateful to the professors of the CL Programme, Prof. Franz Baader and Prof. Steffen Hölldobler, for their constant support during my studies.

Finally, I must not forget to mention my friends from the CL Programme, my flatmates, and, most importantly, my parents and brothers, who all, in their way, give me love, support and joy.

Chapter 1

Introduction

In the 1960's and 1970's it was recognized that knowledge representation (KR) and reasoning is the main part of any intelligent system. Early KR systems such as semantic networks [Quillian 1968] and frames [Minsky 1975] used simple graphs and structured objects to represent knowledge and many algorithms were developed to manipulate these data structures. However, these systems had a major drawback, namely a lack of formal semantics. The subsequent research on overcoming this drawback led to the creation of *description logics*.

Description Logics (DLs) are a family of logic-based knowledge representation formalisms designed to represent and reason about conceptual knowledge in a structured and semantically well-understood way (see [Baader et al. 2003a]). In other words, most DLs are well-behaved fragments of first order logic. In general, they sacrifice some of the expressivity of first order logic in order to regain decidability of reasoning.

The basic notions in DLs are *concepts* (unary predicates) and *roles* (binary relations). A specific DL is mainly characterized by a set of *constructors* it provides to build more complex concepts and roles out of atomic ones. The basic propositionally closed description logic is called $\mathcal{ALC}$ (Attributive Language with Complements) [Schmidt-Schauß & Smolka 1991]. Intuitively, the following $\mathcal{ALC}$-concept:

$$\mathsf{Female} \sqcap \forall \mathsf{has\_child}.\neg\mathsf{Female} \sqcap \exists \mathsf{has\_child}.(\mathsf{Doctor} \sqcup \mathsf{Lawyer})$$

describes "A mother who has only sons and at least one of them is a doctor or a lawyer". Here, Female, Doctor, and Lawyer are atomic concepts, while has_child is a role. In order to illustrate that $\mathcal{ALC}$ corresponds to a first order logic fragment with only two variables, we give a translation of the above concept into first order logic:

$$\begin{aligned} \mathsf{Female}(x) \quad & \wedge \quad \forall y.(\mathsf{has\_child}(x,y) \rightarrow \neg\mathsf{Female}(y)) \\ & \wedge \quad \exists y.(\mathsf{has\_child}(x,y) \wedge (\mathsf{Doctor}(y) \vee \mathsf{Lawyer}(y))) \end{aligned}$$

From the logical point of view, description logics are closely related to modal logics, e.g. $\mathcal{ALC}$ itself is a notational variant of the multi modal logic $\mathbf{K}_\omega$. To see the relation between $\mathcal{ALC}$ and $\mathbf{K}_\omega$, it suffices to view atomic concepts as propositional variables, roles as modal parameters, the $\forall$ constructor as a box operator, and $\exists$ as a diamond. More complex description logics provide for various additional expressive means, such as: nominals (individuals) [Areces et al. 1999]; number restrictions [Hollunder & Baader 1991]; inverse and transitive roles and role hierarchies [Horrocks & Sattler 1999]; Boolean operations on roles [Lutz & Sattler 2001]. Most of these expressive means have their counterpart in modal logics. Moreover, DLs can be equipped with a terminological component, called a *TBox*.

The most important reasoning problems considered in DLs are *satisfiability* and *subsumption* of concepts. A concept is said to be satisfiable if it is consistent. With concept subsumption, a determination of subconcept–superconcept relationships is meant. Unsurprisingly, the higher the expressivity of a DL is, the more complex these reasoning problems are. $\mathcal{ALC}$-concept satisfiability is a PSPACE-complete problem, and various decidable extensions of $\mathcal{ALC}$ are PSPACE-, EXPTIME-, and even NEXPTIME-complete.

In propositionally closed description logics, concept subsumption can be reduced to concept satisfiability. Thus, it suffices to develop a procedure which decides satisfiability of concepts. One of the most popular such decision procedures is a *tableau algorithm* [Baader & Sattler 2001]. In general, tableau algorithms decide whether a concept is satisfiable by trying to construct a model for it. Although they do not always provide optimal upper complexity bounds, tableau algorithms are amenable to many optimizations and can often be efficiently implemented. The DL systems FaCT [Horrocks 1998] and Racer [Haarslev & Möller 2001] are successful implementations of the tableau algorithm for the expressive description logic $\mathcal{SHIQ}$ [Horrocks et al. 1999]; they use special techniques to avoid non-determinism and ensure termination.

Due to successfully implemented DL reasoners, as well as a nice compromise between the expressiveness and the complexity of reasoning, in the last decade DLs found applications in various new areas, such as:

- Reasoning about database conceptual models expressed in entity-relationship diagrams or object-oriented schemas (like UML) [Calvanese et al. 1998]. DL translations of database schemas and DL reasoning techniques are used to detect inconsistencies and to retrieve implicit information.

- Providing ontologies for the sematic web [Baader et al. 2003b]. DLs are the foundation of several web ontology languages, including OIL, DAML+OIL [Horrocks et al. 2002], and OWL.

The new applications pointed out that "plain" description logics have the following shortcoming: they lack a means to represent information of a more "concrete" nature, such as sizes, durations, amounts, or spatial extensions. Capturing such concrete qualities turned out to be a task of great importance in the mentioned applications. For example, in order to define a notion of a valid credit card in a web ontology, we need a means of storing the expiry year of the credit card and checking whether it is greater than the current year. In reasoning about database schemas, concrete "datatypes" are needed to capture integrity constraints.

The standard way of how to integrate numbers and other datatypes into description logics is proposed in [Baader & Hanschke 1991]. The idea is to extend DLs with so-called *concrete domains*. A concrete domain $\mathcal{D}$ consists of a set and predicates with a fixed extension over this set. Some concrete domains are, for example:

- the set of natural numbers $\mathbb{N}$ with the following predicates: unary $\geq_0$, binary $=$, and ternary $+$;
- a set of words over some alphabet Σ with a unary predicate $\mathsf{empty\_word}$, binary $\mathsf{prefix\_of}$, and a ternary concatenation predicate.

The integration of concrete domains into description logics is achieved by adding

1. *abstract features*, which are functional roles (such as $\mathsf{has\_mother}$);
2. *concrete features*, which are assigning values from the concrete domain to logical objects (such as size or $\mathsf{population}$);
3. a new constructor which describes constraints on a concrete domain. It has the form $\exists u_1, \ldots, u_n.P$, where u_i are sequences $f_1 \cdots f_k g$ of k abstract features f_1,...,f_k followed by a single concrete feature g (called *paths*), and P is an n-ary predicate from the concrete domain.

The logic obtained in this way, by integrating $\mathcal{ALC}$ with a concrete domain $\mathcal{D}$, is denoted with $\mathcal{ALC}(\mathcal{D})$. A valid credit card can be defined with the $\mathcal{ALC}(\mathcal{D})$-concept:

$$\mathsf{CreditCard} \sqcap \exists \mathsf{expiry\_year}. \geq_{2004}$$

In this example, $\mathsf{CreditCard}$ is a concept and $\mathsf{expiry\_year}$ is a concrete feature. We use a concrete domain $\mathcal{D}$ based on natural numbers, and $\geq_{2004}$ is a unary predicate with the obvious extension.

Let us consider now a database example. Assume that we want to model a database schema about employees in some company where every employee has at most one direct supervisor. With the $\mathcal{ALC}(\mathcal{D})$-concept:

$$\mathsf{Employee} \sqcap \exists (\mathsf{overtime\_hours}), (\mathsf{boss}\ \mathsf{overtime\_hours}). >$$

we describe an employee who works more overtime hours than his/her boss. Here, Employee is a concept, boss is an abstract feature, while overtime_hours is a concrete one. We use again a numerical concrete domain, and $>$ is the binary "greater than" predicate.

In [Lutz 2002b] it is proved that reasoning with $\mathcal{ALC}(\mathcal{D})$ PSpace-complete if reasoning with the concrete domain $\mathcal{D}$ is in PSpace. Moreover, in [Lutz 2002a] and [Lutz 2003a] it is shown that $\mathcal{ALC}(\mathcal{D})$ with different extensions is NExpTime-complete if reasoning with $\mathcal{D}$ is in NP.

Description logics with concrete domains turned out to be very useful in the discussed applications. The web ontology languages mentioned above are all based on DLs equipped with concrete datatypes. Moreover, in reasoning about database schemas as entity relationship and UML diagrams, concrete domains are used to capture integrity constraints on numerical data [Lutz 2002c].

Recently it has been proposed [Lutz et al. 2003] to further extend the expressive power of DLs with concrete domains by providing them with *key constraints*, as known from databases. Key constraints can be a source of additional inconsistencies in database schemas and, thus, DLs applied in reasoning about database schemas should be able to capture such constraints. The most important key constraints are uniqueness constraints and functional dependencies. *Uniqueness constraints* allow one to describe that a set of properties uniquely determines the identity of an object, such as

Americans are uniquely identified by their social security number

Functional dependencies, in contrast, allow us to state that a property is functionally determined by a set of properties, such as in the following examples:

All books with the same ISBN number have the same title

The extra pay of Microsoft employees is determined by their position and the number of overtime hours they have worked

Although mainly related to databases, key constraints can clearly be very useful in other DL applications.

The research on adding key constraints to DLs with concrete domains performed in [Lutz et al. 2003] revealed that uniqueness constraints have a severe impact on the decidability and computational complexity of reasoning: adding uniqueness constraints to the PSpace-complete $\mathcal{ALC}(\mathcal{D})$ leads to undecidability in the general case, while reasoning becomes NExpTime-complete if the extension is done in a more careful way.

In this thesis, we continue this research, by considering the second type of key constraints – functional dependencies, and their integration into DLs with concrete domains. We should note that functional dependencies in context of DLs have already been investigated in [Borgida & Weddell 1997, Calvanese et al. 2001, Khizder et al. 2001], but only DLs *without* concrete domains were considered. More precisely, we consider an extension of $\mathcal{ALC}(\mathcal{D})$ with *key boxes*, which are finite sets of functional dependencies of the form

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

where $u_1,\ldots,u_n$ and u are paths, and C is a concept. Intuitively, the above functional dependency states that, for the instances of C which share the same values for $u_1,\ldots,u_n$, the values for u must also be the same.

The goal of this thesis is to perform an analysis of the impact on decidability and computational complexity of adding functional dependencies to DLs with concrete domains. Intuitively, functional dependencies are significantly weaker than the uniqueness constraints considered in [Lutz et al. 2003]. While uniqueness constraints can "simulate" nominals and, thus, influence the structure of the logic domain, functional dependencies only state constraints on concrete data and should not affect the logical part of a DL.

The main outcome of our analysis is, however, that the seemingly weaker functional dependencies have an equally dramatic effect on the decidability and complexity of reasoning as uniqueness constraints. It turns out that in $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$– the logic obtained by adding functional dependencies to $\mathcal{ALC}(\mathcal{D})$– reasoning becomes undecidable, and that a slight restriction on the structure of functional dependencies regains decidability but leads to NExpTime-completeness.

The rest of the thesis is organized as follows:

In Chapter 2, we formally introduce the syntax and semantics of $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$-concepts and key boxes. We also define admissible concrete domains. We introduce a class of restricted key boxes, called *safe*, which helps us to preserve the decidability of reasoning.

In Chapter 3, we give the lower complexity bounds for $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$. In Section 3.1, we prove by reduction of the Post Correspondence Problem that $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$-concept satisfiability is undecidable in the general case. Moreover, in Section 3.2 we prove by reduction of a NExpTime-complete version of the tiling problem that $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$-concept satisfiability w.r.t. to safe key boxes is NExpTime-hard.

In Chapter 4, we show that $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$-concept satisfiability w.r.t. safe key boxes is decidable. The decidability is shown in Section 4.1 via a tableau algorithm – in order to ensure its termination, we combine here for the first time blocking with concrete domains. In Section 4.2, we prove the termination, soundness and completeness of the algorithm. Moreover, these proofs yield a NExpTime upper complexity bound for $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$-concept satisfiability w.r.t. safe key boxes.

Finally, concluding remarks are given in Chapter 5.

Chapter 2

The Description Logic $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$

In this section, we formally introduce the syntax and semantics of the description logic $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$. We start with the definition of a concrete domain.

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 is associated with an arity n and an n-ary predicate $P^{\mathcal{D}} \subseteq \Delta_{\mathcal{D}}^n$. $\Diamond$

Based on concrete domains, we define $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$-concepts, as well as weak and strong functional dependencies together with key boxes.

Definition 2 ($\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$ Syntax). Let N_C, N_R and N_{cF} be pairwise disjoint and countably infinite sets of *concept names, role names*, and *concrete features*. Furthermore, we assume that N_R contains a countably infinite subset 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 $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$-concepts is the smallest set such that

- every concept name is a concept
- 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:

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

A *weak functional dependency* is an expression

$$(u_1, \ldots, u_k \text{ wkeyfor } C, u),$$

and a *strong functional dependency* is an expression

$$(u_1, \ldots, u_k \text{ skeyfor } C, u)$$

where $u_1,\ldots,u_k$ ($k \geq 1$) and u are paths, and C is a concept. A finite set of functional dependencies is called *key box*. If a key box contains only weak functional dependencies, it is called *weak*. $\Diamond$

We use $\top$ as abbreviation for an arbitrary propositional tautology, $\bot$ as abbreviation for $\neg\top$, and $C \rightarrow D$ as abbreviation for $\neg C \sqcup D$. Intuitively, the two kinds of functional dependencies differ in the following way: if two instances a and b of C have the same values for the paths $u_1,\ldots,u_n$, then the weak functional dependencies enforce that they share the same values for u only if both a and b have a defined u-value. The strong ones, additionally, enforce that the u-value of b is defined if this is the case for a and vice versa; these values must then also coincide. Sometimes we will use $(u_1, \ldots, u_k \text{ depfor } C, u)$ if the type of the functional dependency is not important.

Before giving the formal semantics of $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$, let us consider the following examples of functional dependencies:

Example 1:

1. The first example from Chapter 1: "books with the same ISBN number have the same titles" can be modelled as the strong functional dependency:

$$\big(\mathsf{isbn} \text{ skeyfor } \mathsf{Book}, \mathsf{title}\big)$$

where Book is a concept, and isbn, title are concrete features.

2. Let us now consider our second example from Chapter 1: "the extra pay of Microsoft employees is determined by their position and the number of overtime hours they have worked". This can be expressed with the *n-ary* functional dependency:

$$\big(\mathsf{overtime\_hours}, \mathsf{position} \text{ wkeyfor } \exists\mathsf{works\_for.Microsoft}, \mathsf{extra\_pay}\big).$$

The concept $\exists$works_for.Microsoft refers to Microsoft employees, and overtime_hours, position, and extra_pay are concrete features. Here is natural to use the weak form of functional dependency, since we want to allow for the case that an arbitrary employee gives up on his extra pay (e.g., in favor of more vacation days).

3. Suppose that we want to express that the tax married Germans pay is uniquely defined by their income, their spouse's income and the number of children they have. We can use the *sequence* of features (spouse income) to refer to the income of the spouse in the strong functional dependency:

$$\big(\mathsf{income}, (\mathsf{spouse\ income}), \mathsf{num\_child} \text{ skeyfor } \mathsf{Married} \sqcap \mathsf{German}, \mathsf{tax}\big)$$

Here, the concept Married $\sqcap$ German describes married Germans, spouse is a concrete feature, while income, num_child, and tax are concrete ones.

4. Finally, let us consider the example of an integrated patient management system. Assume that the illness diagnoses have their unique identification codes, as do stations of each hospital. We can state that patients of one hospital with the same diagnosis are put into the same station, with the following functional dependency:

$$\big((\mathsf{diagnosis\ id}), (\mathsf{hospital\ name}) \text{ skeyfor } \mathsf{Patient}, (\mathsf{station\ id})\big)$$

Clearly, Patient is a concept; diagnosis, hospital, and station are abstract features, while id and name are concrete ones. Note that we used the sequence of features (station id) on the *right-hand side* of the functional dependency.

Let us now introduce the semantics of $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$ together with the most common reasoning problems.

Definition 3 ($\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$ 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 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:

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

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 weak functional dependency $(u_1, ..., u_k \text{ wkeyfor } C, u)$ if, for all $a, b \in C^\mathcal{I}$, the following holds: if, for $1 \leq i \leq n$, $u_i^\mathcal{I}(a) = u_i^\mathcal{I}(b)$ and $u^\mathcal{I}(a)$ and $u^\mathcal{I}(b)$ are defined, then $u^\mathcal{I}(a) = u^\mathcal{I}(b)$.

$\mathcal{I}$ *satisfies* a strong functional dependency $(u_1, ..., u_k \text{ skeyfor } C, u)$ if, for all $a, b \in C^{\mathcal{I}}$, the following holds: if, for $1 \leq i \leq n$, $u_i^{\mathcal{I}}(a) = u_i^{\mathcal{I}}(b)$ and $u^{\mathcal{I}}(a)$ is defined, then $u^{\mathcal{I}}(b)$ is defined and $u^{\mathcal{I}}(a) = u^{\mathcal{I}}(b)$.

$\mathcal{I}$ is a *model* of a key box $\mathcal{K}$ iff $\mathcal{I}$ satisfies all functional dependencies 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 $\mathcal{K}$. $\diamondsuit$

In $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$, as well as in any description logic that provides for negation and conjunction, concept subsumption can be reduced to concept 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 $C \not\sqsubseteq_{\mathcal{K}} \bot$. Therefore, it suffices to concentrate on concept satisfiability when investigating the complexity of $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$.

When developing decision procedures for $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$, we do not want to restrict us to a particular concrete domain $\mathcal{D}$. Therefore, we need a good interface between the logic and concrete domain reasoner. This is usually achieved by requiring that the satisfiability of finite predicate conjunctions in $\mathcal{D}$ is decidable [Baader & Hanschke 1991, Lutz 2002a]. Such concrete domains are called admissible:

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.

A concrete domain $\mathcal{D}$ is *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, 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.* $\diamondsuit$

It will be shown in subsequent chapters that the decidability and complexity of $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$ reasoning problems depends on whether we use key boxes as introduced so far (i.e., general ones) or a restricted version. Thus, we introduce a safe class of key boxes that will help us to preserve decidability:

Definition 5 (safe, unary) A key box $\mathcal{K}$ is called

- *safe* if none of concepts appearing in functional dependencies in $\mathcal{K}$ has a subconcept of the form $\exists u_1, \ldots, u_n.P$;
- a *unary key box* if all functional dependencies in $\mathcal{K}$ are of the form $(u \text{ depfor } C, u')$.

$\Diamond$

Note that requiring a key box to be safe is not a very severe restriction. Safe key boxes are still very expressive; if we revisit the examples from this chapter, we see that they are all modelled using safe key boxes.

Finally, if we want to emphasize that the key box we use is not unary, we refer to it as *n-ary key box*.

Chapter 3

Lower bounds

In this section, we present the lower complexity bounds for the description logic $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$. We first show that $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$-concept satisfiability w.r.t. general key boxes is undecidable. As it will be shown in Chapter 4, the decidability can be regained if we restrict ourselves to safe key boxes as introduced in Definition 5. In this case, we show that reasoning is NExpTime-hard. Both results are obtained via variations of the corresponding reductions for $\mathcal{ALCK}(\mathcal{D})$ ($\mathcal{ALC}(\mathcal{D})$ with uniqueness constraints) presented in [Lutz et al. 2002].

3.1 Undecidability of $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$ with General Key Boxes

We prove that satisfiability of $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$-concepts w.r.t. key boxes is undecidable in the general case. The proof is done by reduction of the well-known undecidable Post Correspondence Problem (PCP) [Post 1946].

Definition 6 (PCP). An *instance* P of the *Post Correspondence Problem* is given by a finite, non-empty list $(l_1, r_1), \ldots, (l_k, r_k)$ 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

$$l_{i_1} \cdots l_{i_m} = r_{i_1} \cdots r_{i_m}$$

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

In order to encode the PCP, it is natural to take a concrete domain based on words and word concatenation. We will use the concrete domain W introduced in [Lutz 2003b].

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

$$
\begin{array}{rcl}
\mathsf{Step} & := & \bigsqcap_{1\le i\le k} \exists f_i.(\exists g.\, =_\epsilon \sqcap \exists s.\, =_\epsilon \sqcap \exists l,r.\, \neq) \\
& & \sqcap \bigsqcap_{1\le i\le k} (\exists l, f_i l.\mathsf{conc}_{l_i} \sqcap\ \exists r, f_i r.\mathsf{conc}_{r_i}) \\
C_P & := & \exists l.\, =_\epsilon \sqcap \exists r.\, =_\epsilon \\
& & \sqcap\ \exists R.(\exists g.\, =_\epsilon \sqcap \exists s.\, \neq_\epsilon \sqcap \neg\mathsf{Step}) \\
& & \sqcap\ \mathsf{Step} \\
\mathcal{K}_P & := & \{(g \text{ wkeyfor } \neg\mathsf{Step}, s)\}
\end{array}
$$

Figure 3.1: The $\mathcal{ALC}(\mathsf{W})^{\mathcal{FD}}$ reduction concept C_P and key box $\mathcal{K}_P$

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

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

$\Diamond$

It is obvious that the concrete domain W satisfies the Conditions 1 and 2 of admissibility. Moreover, in [Lutz 2003b] it is shown that W-satisfiability is in PTime, and, thus, W is admissible. This is important, since we want to show that the undecidability is not due to non-admissibility or a high complexity of the concrete domain, but due to functional dependencies .

For an instance of the Post Correspondence Problem $P = (l_1, r_1), \ldots, (l_k, r_k)$, we define now an $\mathcal{ALC}(\mathsf{W})^{\mathcal{FD}}$-concept C_P and a key box $\mathcal{K}_P$ such that models of C_P and $\mathcal{K}_P$ encode all potential solutions for P. The reduction concept C_P and key box $\mathcal{K}_P$ are given in Figure 3.1. Here, R is a role name, $f_1, \ldots, f_k$ denote abstract features, while l, r, g, and s denote concrete features. The definition of the concept Step is not a TBox definition, but serves only as abbreviation.

The models of C_P and $\mathcal{K}_P$ have the form of an infinite k-ary tree whose edges are labelled with $f_1, \ldots, f_k$ and whose root has an additional R-successor, as shown in Figure 3.2. Intuitively, each node represents a sequence of indices $i_1, \ldots, i_n$, its l-successor represents the left concatenation $l_{i_1} \cdots l_{i_n}$, and its r-successor the corresponding right concatenation $r_{i_1} \cdots r_{i_n}$. By the definition of

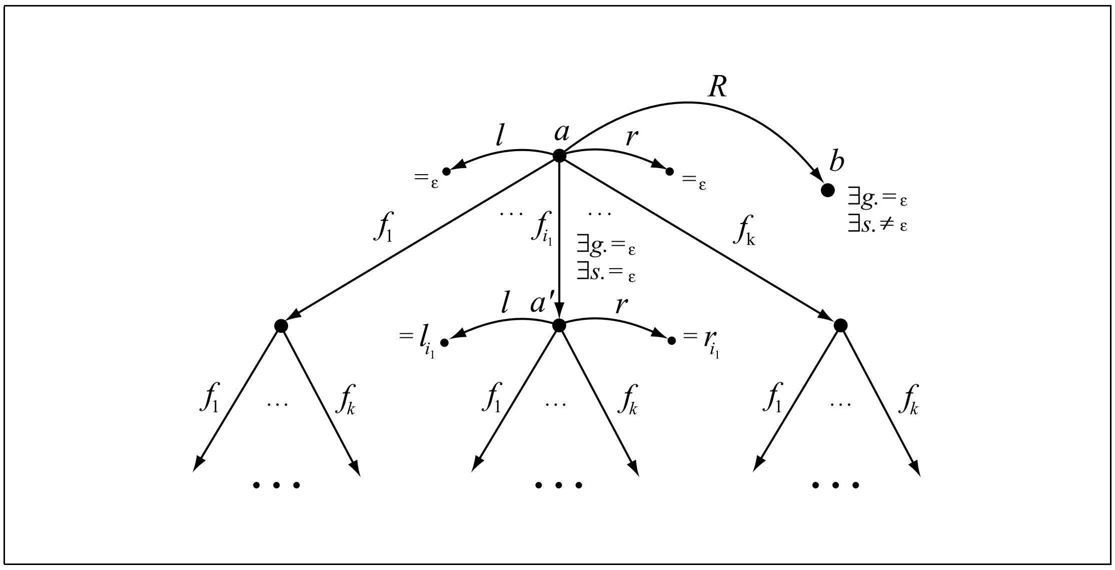

Figure 3.2: A model $\mathcal{I}$ of C_P and $\mathcal{K}_P$

$\mathcal{K}_P$ it is ensured that every potential solution is considered, and by C_P that no potential solution is indeed a solution. Thus, C_P and $\mathcal{K}_P$ have a common model iff P has no solution.

Note that the key box $\mathcal{K}_P$ used in the reduction is a weak one, which means that restricting us to weak key boxes would not preserve decidability.

Lemma 1 *Let $P = (l_1, r_1), \ldots, (l_k, r_k)$ be a PCP. Then P has a solution iff the concept C_P is unsatisfiable w.r.t. the key box $\mathcal{K}_P$.*

Proof. For both directions, we will show the contrapositives.

First we show that, if C_P is satisfiable w.r.t. $\mathcal{K}_P$, then P has no solution. Let the interpretation $\mathcal{I}$ be a model of C_P and $\mathcal{K}_P$. Then there exists an $a \in \Delta_\mathcal{I}$ such that $a \in C_P^\mathcal{I}$. Due to the first line of C_P we have that $l^\mathcal{I}(a) = r^\mathcal{I}(a) = \epsilon$. Due to the second line of C_P, a has an R-successor b such that $b \in (\exists g. =_\epsilon)^\mathcal{I}$, $b \in (\exists s. \neq_\epsilon)^\mathcal{I}$ and $b \in (\neg\mathsf{Step})^\mathcal{I}$.

We prove the following claim in order to show that $\mathcal{I}$ enumerates all potential solutions of P:

Claim: For every $n \geq 1$ and every sequence of indices $S = i_1, \ldots, i_n$, $1 \leq i_j \leq k$, there is a $c \in \Delta_\mathcal{I}$ such that:

(P1) $c = f_{i_n}^\mathcal{I}(\cdots(f_{i_1}^\mathcal{I}(a))\cdots)$,

(P2) $l^\mathcal{I}(c) = l_{i_1} \cdots l_{i_n}$, $r^\mathcal{I}(c) = r_{i_1} \cdots r_{i_n}$,

(P3) $l^\mathcal{I}(c) \neq r^\mathcal{I}(c)$, and

(P4) $c \in \mathsf{Step}^{\mathcal{I}}$.

Proof. We prove the claim by induction on n:

1. $n = 1$. Let $S = i_1$. Due to the third line of C_P and the first line of Step, a has an f_{i_1}-successor a' ($a' = f_{i_1}^{\mathcal{I}}(a)$) such that $a' \in (\exists g.{=_\epsilon})^{\mathcal{I}}$ and $a' \in (\exists s.{=_\epsilon})^{\mathcal{I}}$. Moreover, from the definition of Step and since $l^{\mathcal{I}}(a) = r^{\mathcal{I}}(a) = \epsilon$, it follows that $l^{\mathcal{I}}(a') = l_{i_1}$, $r^{\mathcal{I}}(a') = r_{i_1}$, and $l^{\mathcal{I}}(a') \neq r^{\mathcal{I}}(a')$. Thus, the conditions (P1), (P2), and (P3) are satisfied. In order to show that (P3) holds, we assume the opposite, i.e., that $a' \in (\neg\mathsf{Step})^{\mathcal{I}}$. Since $b \in (\neg\mathsf{Step})^{\mathcal{I}}$ and $g^{\mathcal{I}}(a') = g^{\mathcal{I}}(b)$, the functional dependency $(g \text{ wkeyfor } \neg\mathsf{Step}, s) \in \mathcal{K}_P$ implies $s^{\mathcal{I}}(a') = s^{\mathcal{I}}(b)$, which does not hold. Therefore, we have that $a' \in \mathsf{Step}^{\mathcal{I}}$ (P4) and the base case is proved.

2. $n = m + 1$, $m \geq 1$. Let $S = i_1, \ldots, i_m, i_{m+1}$. By inductional hypothesis, for $n = m$ and $S' = i_1, \ldots, i_m$ there is a $c \in \Delta_{\mathcal{I}}$ such that (P1)-(P4) hold for c and S'. Since $c \in \mathsf{Step}^{\mathcal{I}}$, the node c has an $f_{i_{m+1}}$-successor c' such that $c' \in (\exists g.{=_\epsilon})^{\mathcal{I}}$, $c' \in (\exists s.\ {=_\epsilon})^{\mathcal{I}}$, and $l^{\mathcal{I}}(c') \neq r^{\mathcal{I}}(c')$. Since $c = f_{i_m}^{\mathcal{I}}(\cdots(f_{i_1}^{\mathcal{I}}(a))\cdots)$, we obtain that $c' = f_{i_{m+1}}^{\mathcal{I}}(f_{i_m}^{\mathcal{I}}(\cdots(f_{i_1}^{\mathcal{I}}(a))\cdots))$. Moreover, due to the second line of Step and since $l^{\mathcal{I}}(c) = l_{i_1}\cdots l_{i_m}$, we get that $l^{\mathcal{I}}(c') = l_{i_1}\cdots l_{i_{m+1}}$. Analogously we obtain $r^{\mathcal{I}}(c') = r_{i_1}\cdots r_{i_{m+1}}$. Thus, c fulfills the conditions (P1), (P2) and (P3). Reasoning similarly as in the previous case, we get that (P4) is satisfied as well, namely $c' \in \mathsf{Step}^{\mathcal{I}}$. Thus, the induction step is proved.

The immediate consequence of this claim is that no potential solution of P is indeed a solution.

For the other direction, we show that if P has no solution, then C_P is satisfiable w.r.t. $\mathcal{K}_P$. We construct an interpretation $\mathcal{I}$ that is a model of C_P and $\mathcal{K}_P$. If $w = i_1 \cdots i_n$ is a sequence of indices, with $\mathsf{lc}(w)$ we denote the concatenation of the words $l_1 \cdots l_n$ and with $\mathsf{rc}(w)$ we denote the concatenation of the words $r_1 \cdots r_n$. We define

$$\begin{array}{rcl}
\Delta_{\mathcal{I}} & := & \{i_1 \cdots i_n \mid n \geq 0 \text{ and } 1 \leq i_j \leq k \text{ for } 1 \leq j \leq n\} \cup \{d\} \\
R^{\mathcal{I}} & := & \{(\epsilon, d)\} \\
f_i^{\mathcal{I}}(w) & := & wi \text{ for } w \in \Delta_{\mathcal{I}} \setminus \{d\},\ \ f_i^{\mathcal{I}}(d) \text{ undefined}, 1 \leq i \leq k \\
l^{\mathcal{I}}(w) & := & \mathsf{lc}(w) \text{ for } w \in \Delta_{\mathcal{I}} \setminus \{d\},\ \ l^{\mathcal{I}}(d) \text{ undefined} \\
r^{\mathcal{I}}(w) & := & \mathsf{rc}(w) \text{ for } w \in \Delta_{\mathcal{I}} \setminus \{d\},\ \ r^{\mathcal{I}}(d) \text{ undefined} \\
g^{\mathcal{I}}(w) & := & \epsilon \text{ for } w \in \Delta_{\mathcal{I}} \\
s^{\mathcal{I}}(w) & := & \left\{ \begin{array}{ll} \epsilon, & w \in \Delta_{\mathcal{I}} \setminus \{d\} \\ a \in \Sigma^+, & w = d \end{array} \right.
\end{array}$$

Since P has no solution, we have $\mathsf{lc}(w) \neq \mathsf{rc}(w)$ for all $w \in \Delta_\mathcal{I} \setminus \{d\}$ and, due to the definition of $f_i^\mathcal{I}$ for $1 \leq i \leq k$, $l^\mathcal{I}$, $r^\mathcal{I}$, $g^\mathcal{I}$, and $s^\mathcal{I}$ we have that $\mathsf{Step}^\mathcal{I} = \Delta_\mathcal{I} \setminus \{d\}$. Now it is easy to see that $\mathcal{I}$ is a model of C_P ($\epsilon \in C_P^\mathcal{I}$) and $\mathcal{K}_P$. $\Box$

The following theorem is an immediate consequence of the previous lemma.

Theorem 1 *The satisfiability of $\mathcal{ALC}(\mathsf{W})^{\mathcal{FD}}$-concepts w.r.t. weak key boxes is undecidable.*

In the following corollary, we emphasize the fact that the undecidability result is obtained by using a simple concrete domain, and that it holds even for unary weak key boxes which contain only paths of length one (*path-freeness*):

Corollary 1 *There exists a concrete domain $\mathcal{D}$ such that $\mathcal{D}$-satisfiability is in* PTIME *and $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$-satisfiability w.r.t. unary path-free weak key boxes is undecidable.*

A remark is in order. The concrete domain W may seem unnatural and the obtained undecidability result, therefore, not very relevant. However, in [Lutz 2002a] it is shown that words over alphabet Σ can be interpreted as numbers written at base $\#\Sigma + 1$ (without "0 digit"). Thus, the corresponding natural numbers at base 10 can be used to represent non-empty words, and 0 to represent the empty word. Moreover, the concatenation of two words v and w can be expressed as $vw = v \cdot (\#\Sigma + 1)^{|w|} + w$, where $|w|$ denotes the length of the word w. Since exponentiation can be expressed as multiple multiplications, similarly as in [Lutz et al. 2002] we obtain the following theorem:

Theorem 2 *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 predicates $=_0$ with $(=_0)^\mathcal{D} = \{0\}$ and $\neq_0$ with $(\neq_0)^\mathcal{D} = \Delta_\mathcal{D} \setminus \{0\}$,*
2. *binary equality and inequality predicates,*
3. *a ternary predicate $+$ with $(+)^\mathcal{D} \cap \{(k_1, k_2, x) \mid k_1, k_2 \in \mathbb{N}$ and $x \in \Delta_\mathcal{D}\} = \{(k_1, k_2, k_1 + k_2) \mid k_1, k_2 \in \mathbb{N}\}$, and*
4. *a ternary 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 satisfiability of $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$-concepts w.r.t. (general) weak key boxes is undecidable.

3.2 NExpTime-Hardness of $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$ with Safe Key Boxes

In this section, we will consider $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$ with safe key boxes. Recall that safe key boxes do not allow subconcepts of the form $\exists u_1, \ldots, u_n.P$ of concepts occurring in functional dependencies. We will prove a NExpTime-lower bound for $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$-satisfiability w.r.t. safe key boxes by reducing a NExpTime-complete variant of the well-known undecidable domino problem.

A domino problem is given by a finite set of tile types, all of which are of the same size, have a square shape and colored edges. An unlimited number of tiles of each type is available. In the NExpTime-variant of the domino problem 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 parallel edges are "glued" together. The adjacent tiles must be pairwise compatible, meaning that their touching edges must be of the same color.

Definition 8 (Domino System). A *domino system* $\mathcal{D}$ is a triple (T, H, V), where $T = \{0, \ldots, K\}$, $K \in \mathbb{N}$ is a finite set of *tile types* and $H, V \subseteq T \times T$ are the horizontal and vertical matching conditions. Let $\mathcal{D}$ be a domino system and $a = a_0, \ldots, a_{n-1}$ an *initial condition*, i.e. an n-tuple of tiles. A mapping $\tau : \{0, \ldots, 2^{n+1} - 1\} \times \{0, \ldots, 2^{n+1} - 1\} \to T$ is a *solution* for $\mathcal{D}$ and a iff, for all $x, y < 2^{n+1}$, the following holds:

- if $\tau(x, y) = t$ and $\tau(x +_{2^{n+1}} 1, y) = t'$, then $(t, t') \in H$
- if $\tau(x, y) = t$ and $\tau(x, y +_{2^{n+1}} 1) = t'$, then $(t, t') \in V$
- $\tau(i, 0) = a_i$ for $i < n$

where $+_i$ denotes addition modulo i. $\diamondsuit$

It is shown in [Börger et al. 1997, Lutz 2002a] that the variant of the domino problem we introduced is NExpTime-complete. Now we introduce a concrete domain D which we need for encoding the domino problem.

Definition 9 (Concrete Domain D). The concrete domain D is defined by setting $\Delta_\mathsf{D} := \{0, 1\}$ and Φ_D to the smallest set containing the following predicates:

- unary predicates $\top_\mathsf{D}$ and $\bot_\mathsf{D}$ with $\top_\mathsf{D}^\mathsf{D} = \Delta_\mathsf{D}$ and $\bot_\mathsf{D}^\mathsf{D} = \emptyset$;
- unary predicates $=_0$ and $=_1$ with $=_i^\mathsf{D} = \{i\}$.

$\diamondsuit$

It is not difficult to see that D is an admissible concrete domain, and that D-satisfiability is in PTime. The concrete values we want to encode in this concrete domain are x- and y-coordinates of the $2^{n+1} \times 2^{n+1}$ torus and the labels of every grid position, i.e., the tile type that covers it. Since $\Delta_{\mathsf{D}} = \{0, 1\}$, we have to use the binary representation of these values. In the case of x- and y-coordinate, it suffices to use concrete features xpos_0,. . . , xpos_n and ypos_0,. . . , ypos_n to represent them binary. As for grid labels, we can encode them as l_0,. . . , l_m, where $m = [\log_2 K]$.

For a given domino system $\mathcal{D}$ and initial condition a, we now define an $\mathcal{ALC}(\mathsf{D})^{\mathcal{FD}}$-concept $C_{\mathcal{D},a}$ and a safe key box $\mathcal{K}_{\mathcal{D},a}$ such that $C_{\mathcal{D},a}$ is satisfiable w.r.t. $\mathcal{K}_{\mathcal{D},a}$ iff $\mathcal{D}$ has a solution with initial condition $a = a_0, \ldots, a_{n-1}$. The reduction is given in Figure 3.3.

We use here $\forall R^i.C$ as an abbreviation for the n-fold nesting $\forall R. \cdots \forall R.C$. The names X_i and Y_i denote concept names; R, R_x and R_y denote role names, while xpos_i, ypos_i and l_i denote concrete features. The definitions of the concepts TreeX, TreeY, etc. serve only as abbreviation. We use $\mathsf{bit}_i(n)$ to denote the i-th bit of the binary representation of $n \in \mathbb{N}$. Before proving formally the correctness of the reduction, let us explain some parts of the concept $C_{\mathcal{D},a}$. The TreeX concept enforces the existence of a binary tree T_x of depth $n+1$. The edges of this tree are labelled with role R, and the concept names X_0, . . . , X_n are used for a binary numbering of the leaves of T_x. For $d \in \Delta_{\mathcal{I}}$ we define $\mathsf{xpos}(d)$ in the following way:

$$\mathsf{xpos}(d) = \sum_{i=0}^{n} \alpha_i(d) \cdot 2^i \text{ where } \alpha_i(d) = \begin{cases} 1, & \text{if } d \in X_i^{\mathcal{I}} \\ 0, & \text{otherwise.} \end{cases}$$

It is ensured by the TreeX and DistX_k concepts that, for each $i < 2^{n+1}$, there is a leaf d_i of T_x with $\mathsf{xpos}(d_i) = i$. Furthermore, the TreeY enforces that each leaf d_i of T_x is a root of another tree of depth $n+1$; let us call this tree T_y^i. Due to the TreeY, DistX_n, and DistY_k concepts, we have that the leaves of $\bigcup_{i=0}^{n+1} T_y^i$ are binarily numbered by concepts X_0, . . . , X_n, Y_0, . . . , Y_n. Moreover, for each leaf d of the tree T_y^i it holds that $\mathsf{xpos}(d) = i$. Let us now define $\mathsf{ypos}(d)$ for $d \in \Delta_{\mathcal{I}}$ analogously to xpos:

$$\mathsf{ypos}(d) = \sum_{i=0}^{n} \beta_i(d) \cdot 2^i \text{ where } \beta_i(d) = \begin{cases} 1, & \text{if } d \in Y_i^{\mathcal{I}} \\ 0, & \text{otherwise.} \end{cases}$$

We have that, for each $i, j < 2^{n+1}$ there is a leaf $d_{i,j}$ of T_y^i such that $\mathsf{ypos}(d_{i,j}) = j$ (and $\mathsf{xpos}(d_{i,j}) = i$). Intuitively, each $d_{i,j}$ represents the (i, j) position of the $2^{n+1} \times 2^{n+1}$-torus.

The concrete features xpos_0, . . . , xpos_n are used as a binary representation of the x-coordinate (xpos) and concrete features ypos_0, . . . , ypos_n as a binary representation of the y-coordinate (ypos) for every $d_{i,j}$ and its horizontal neighbours

$$
\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 \rightarrow \forall R.X_i) \sqcap (\neg X_i \rightarrow \forall R.\neg X_i))\\
\mathsf{DistY}_k &:= \bigsqcap_{i=0..k}((Y_i \rightarrow \forall R.Y_i) \sqcap (\neg Y_i \rightarrow \forall R.\neg Y_i))\\
\mathsf{TransXPos} &:= \bigsqcap_{i=0..n}((X_i \rightarrow \exists \mathsf{xpos}_i.\ =_1) \sqcap (\neg X_i \rightarrow \mathsf{xpos}_i.\ =_0))\\
\mathsf{TransYPos} &:= \bigsqcap_{i=0..n}((Y_i \rightarrow \exists \mathsf{ypos}_i.\ =_1) \sqcap (\neg Y_i \rightarrow \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}((Y_i \rightarrow \forall R_x.Y_i) \sqcap (\neg Y_i \rightarrow \forall R_x.\neg Y_i))\\
&\quad \bigsqcap_{k=0..n}(\bigsqcap_{j=0..k} X_j) \rightarrow ((X_k \rightarrow \forall R_x.\neg X_k) \sqcap (\neg X_k \rightarrow \forall R_x.X_k))\\
&\quad \bigsqcap_{k=0..n}(\bigsqcup_{j=0..k} \neg X_j) \rightarrow ((X_k \rightarrow \forall R_x.X_k) \sqcap (X_k \rightarrow \forall R_x.\neg X_k))\\
\mathsf{YSuccOk} &:= \bigsqcap_{i=0..n}((X_i \rightarrow \forall R_y.X_i) \sqcap (\neg X_i \rightarrow \forall R_y.\neg X_i))\\
&\quad \bigsqcap_{k=0..n}(\bigsqcap_{j=0..k} Y_j) \rightarrow ((Y_k \rightarrow \forall R_y.\neg Y_k) \sqcap (\neg X_k \rightarrow \forall R_y.Y_k))\\
&\quad \bigsqcap_{k=0..n}(\bigsqcup_{j=0..k} \neg Y_j) \rightarrow ((Y_k \rightarrow \forall R_y.Y_k) \sqcap (X_k \rightarrow \forall R_y.\neg Y_k))\\
\mathsf{Tile}_i &:= \bigsqcap_{j=0..m} \exists l_j.\ =_{\mathsf{bit}_j(i)}\\
\mathsf{Label} &:= \bigsqcup_{i \in T} \mathsf{Tile}_i\\
\mathsf{CheckMatch} &:= \bigsqcup_{(i,j)\in H}(\mathsf{Tile}_i \sqcap \forall R_x.\mathsf{Tile}_j) \sqcap \bigsqcup_{(i,j)\in V}(\mathsf{Tile}_i \sqcap \forall R_y.\mathsf{Tile}_j)\\
\mathsf{Init} &:= \bigsqcap_{i=0..n-1}\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) \rightarrow \mathsf{Tile}_{a_i}\Big)\\
C_{\mathcal{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{Init} \sqcap \mathsf{Label} \sqcap \mathsf{CheckMatch})\\
\mathcal{K}_{\mathcal{D},a} &:= \{(\mathsf{xpos}_0, \ldots, \mathsf{xpos}_n, \mathsf{ypos}_0, \ldots, \mathsf{ypos}_n \text{ wkeyfor } \top, l_0),\\
&\quad \vdots\\
&\quad \mathsf{xpos}_0, \ldots, \mathsf{xpos}_n, \mathsf{ypos}_0, \ldots, \mathsf{ypos}_n \text{ wkeyfor } \top, l_m)\}
\end{aligned}
$$

Figure 3.3: The $\mathcal{ALC}(\mathsf{D})^{\mathcal{FD}}$ reduction concept $C_{\mathcal{D},a}$ and key box $\mathcal{K}_{\mathcal{D},a}$

(R_x-successors) and vertical neighbours (R_y-successors). This is realized by the concepts TransX, TransY, and Succs.

The definition of the $\mathsf{XSuccOk}$ and $\mathsf{YSuccOk}$ concepts is the DL formulation of the propositional formula:

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

which encodes incrementation modulo 2^{n+1}, i.e., if $x_0,\ldots,x_n$ is the binary representation of the number t, and $x'_0,\ldots,x'_n$ of the number t', we have that $t' = t +_{2^{n+1}} 1$. These concepts, together with the Succs concept ensure that every $d_{i,j}$ has a horizontal and a vertical neighbour with correct coordinates.

The concepts Tile_i for $i \leq K$ represent tile types. The definition of the Label concept ensures that every grid position is covered with a tile type. For $d \in \mathsf{Label}^{\mathcal{I}}$ we define

$$l(d) = \sum_{i=0}^{m} l_i^{\mathcal{I}}(d) \cdot 2^i.$$

The definition of the Tile_i concepts implies that $d \in \mathsf{Tile}_i^{\mathcal{I}}$ iff $l(d) = i$. This means that l takes the role of the "tiling function", i.e., a potential solution for $\mathcal{D}$ and a. Thus, it is ensured that every grid position is covered with no more than one domino type. The functional dependencies $(\mathsf{xpos}_0, \ldots, \mathsf{xpos}_n, \mathsf{ypos}_0, \ldots, \mathsf{ypos}_n$ wkeyfor $\top, l_i)$ for $i \leq m$ ensure that the x- and y-coordinates uniquely define the tile that covers position (x, y) of the grid.

The initial and matching conditions of the domino system are tackled by the Label, Init, and $\mathsf{CheckMatch}$ concepts.

Lemma 2 *Let $\mathcal{D}$ be a domino system and a an initial condition. Then $\mathcal{D}$ and a have a solution iff the concept $C_{\mathcal{D},a}$ is satisfiable w.r.t. the key box $\mathcal{K}_{\mathcal{D},a}$.*

Proof. Let $\mathcal{D} = (T, H, V)$ be a domino system and $a = a_0, \ldots, a_{n-1}$ an initial condition. First assume that $C_{\mathcal{D},a}$ is satisfiable w.r.t. $\mathcal{K}_{\mathcal{D},a}$ and an interpretation $\mathcal{I}$ is their model. Using induction on n and the definitions of the TreeX, DistX_i, TreeY, and DistY_i concepts it is not difficult to show that there exists a $d_{i,j} \in \Delta_{\mathcal{I}}$, for all $i, j < 2^{n+1}$, such that

$$\mathsf{xpos}(d_{i,j}) = i \text{ and } \mathsf{ypos}(d_{i,j}) = j,$$

i.e., every $d_{i,j}$ corresponds to the (i, j) position of a $2^{n+1} \times 2^{n+1}$-torus. The domain elements $d_{i,j}$ are leaves of a binary tree of depth $2(n+1)$ whose root is an instance of $C_{\mathcal{D},a}$, and whose existence is enforced by the definition of the concepts named above.

The second line of $C_{\mathcal{D},a}$ ensures that $d_{i,j} \in (\mathsf{TransXPos} \sqcap \mathsf{TransYPos})^{\mathcal{I}}$. Due to the concept $\mathsf{TransXPos}$, for every $d \in (\mathsf{TransXPos})^{\mathcal{I}}$ it holds that $\mathsf{xpos}_0^{\mathcal{I}}(d)$,..., $\mathsf{xpos}_n^{\mathcal{I}}(d)$ is the binary representation of $\mathsf{xpos}(d)$; similarly, due to $\mathsf{TransYPos}$, for every $d \in (\mathsf{TransYPos})^{\mathcal{I}}$ it holds that $\mathsf{ypos}_0^{\mathcal{I}}(d)$,..., $\mathsf{ypos}_n^{\mathcal{I}}(d)$ is the binary representation of $\mathsf{ypos}(d)$.

Moreover, the definition of $C_{\mathcal{D},a}$ implies that $d_{i,j} \in (\mathsf{Succs} \sqcap \mathsf{XSuccOk} \sqcap \mathsf{YSuccOk})^{\mathcal{I}}$. Using the definitions of the concepts Succs, $\mathsf{XSuccOk}$, and $\mathsf{YSuccOk}$, we obtain that $\emptyset \neq R_x^{\mathcal{I}}(d_{i,j}), R_y^{\mathcal{I}}(d_{i,j}) \subseteq (\mathsf{TransXPos} \sqcap \mathsf{TransYPos})^{\mathcal{I}}$, and:

$$\begin{array}{l} \text{if } (d_{i,j}, e) \in R_x^{\mathcal{I}} \text{ then } \mathsf{xpos}(e) = i +_{2^{n+1}} 1 \text{ and } \mathsf{ypos}(e) = j \\ \text{if } (d_{i,j}, e) \in R_y^{\mathcal{I}} \text{ then } \mathsf{xpos}(e) = i \text{ and } \mathsf{ypos}(e) = j +_{2^{n+1}} 1 \end{array} \tag{3.1}$$

Due to the third line of $C_{\mathcal{D},a}$ we obtain that $d_{i,j} \in (\mathsf{Label} \sqcap \mathsf{CheckMatch} \sqcap \mathsf{Init})^{\mathcal{I}}$. By definitions of the Label and Tile_k concepts, it is ensured that every grid position is labelled with precisely one tile, and by Init that the initial condition is satisfied, i.e.:

$$l(d_{i,0}) = a_i \text{ for } i < n$$

Due to the $\mathsf{CheckMatch}$ concept, we have:

$$\begin{array}{lll} \text{if } (d_{i,j}, e) \in R_x^{\mathcal{I}} & \text{then} & (l(d_{i,j}), l(e)) \in H \\ \text{if } (d_{i,j}, e) \in R_y^{\mathcal{I}} & \text{then} & (l(d_{i,j}), l(e)) \in V \end{array} \tag{3.2}$$

Finally, the functional dependencies

$$(\mathsf{xpos}_0, \ldots, \mathsf{xpos}_n, \mathsf{ypos}_0, \ldots, \mathsf{ypos}_n \text{ wkeyfor } \top, l_i)$$

enforce that if $\mathsf{xpos}(e) = \mathsf{xpos}(e')$ and $\mathsf{ypos}(e) = \mathsf{ypos}(e')$ for some $e, e' \in (\mathsf{TransXPos} \sqcap \mathsf{TransYPos} \sqcap \mathsf{Label})^{\mathcal{I}}$, then $l_i^{\mathcal{I}}(e) = l_i^{\mathcal{I}}(e')$ for $i \leq m$, i.e., $l(e) = l(e')$. Combining this with (3.1) and (3.2), we obtain the following:

$$(l(d_{i,j}), l(d_{i+_{2^{n+1}}1,j})) \in H \text{ and } (l(d_{i,j}), l(d_{i,j+_{2^{n+1}}1})) \in V$$

Thus, the matching condition is satisfied and if we define a mapping

$$\tau : \{0, \ldots, 2^{n+1} - 1\} \times \{0, \ldots, 2^{n+1} - 1\} \to T$$

with $\tau(i, j) = l(d_{i,j})$, then τ is a solution for $\mathcal{D}$ and a.

Now let us consider the "only if" direction. Assume that $\tau : \{0, \ldots, 2^{n+1} - 1\} \times \{0, \ldots, 2^{n+1} - 1\} \to T$ is a solution for $\mathcal{D}$ and a. We use τ to construct an interpretation $\mathcal{I}$. First we define the interpretation domain:

$$\begin{array}{rcl}\Delta_\mathcal{I} & := & \{d^{(k)}_{i,0} \mid k \leq n+1,\ i < 2^k\} \\ & & \cup\{d^{(n+1+k)}_{i,j} \mid k \leq n+1,\ i < 2^{n+1},\ j < 2^k\}\end{array}$$

For $p \leq n$, we define $X^\mathcal{I}_p$ and $Y^\mathcal{I}_p$ as follows:

$$\begin{array}{rcl}X^\mathcal{I}_p & := & \{d^{(k)}_{i,j} \in \Delta_\mathcal{I} \mid p < k \leq 2n+2,\ \mathsf{bit}_p(i) = 1\} \\ Y^\mathcal{I}_p & := & \{d^{(k)}_{i,j} \in \Delta_\mathcal{I} \mid n+p < k \leq 2n+2,\ \mathsf{bit}_p(j) = 1\}\end{array}$$

We continue with the interpretation of the role names R, R_x, and R_y:

$$\begin{array}{rcl}R^\mathcal{I} & := & \{(d^{(k)}_{i,0}, d^{(k+1)}_{2i,0}), (d^{(k)}_{i,0}, d^{(k+1)}_{2i+1,0}) \mid k \leq n,\ d^{(k)}_{i,0} \in \Delta_\mathcal{I}\} \\ & & \cup\{(d^{(k)}_{i,j}, d^{(k+1)}_{i,2j}), (d^{(k)}_{i,j}, d^{(k+1)}_{i,2j+1}) \mid n < k < 2n+2,\ d^{(k)}_{i,j} \in \Delta_\mathcal{I}\} \\ R^\mathcal{I}_x & := & \{(d^{(2n+2)}_{i,j}, d^{(2n+2)}_{i+_{2^{n+1}}1,j}) \mid d^{(2n+2)}_{i,j} \in \Delta_\mathcal{I}\} \\ R^\mathcal{I}_y & := & \{(d^{(2n+2)}_{i,j}, d^{(2n+2)}_{i,j+_{2^{n+1}}1}) \mid d^{(2n+2)}_{i,j} \in \Delta_\mathcal{I}\}\end{array}$$

Finally, we give the interpretation of the concrete features xpos_p and ypos_p for every $p \leq n$ and of the concrete features l_k for $k \leq m$:

$$\begin{array}{rcl}\mathsf{xpos}^\mathcal{I}_p(d^{(2n+2)}_{i,j}) & := & \mathsf{bit}_p(i) \\ \mathsf{ypos}^\mathcal{I}_p(d^{(2n+2)}_{i,j}) & := & \mathsf{bit}_p(j) \\ l^\mathcal{I}_k(d^{(2n+2)}_{i,j}) & := & \mathsf{bit}_k(\tau(i,j))\end{array}$$

Here, $\mathsf{xpos}^\mathcal{I}_p$, $\mathsf{ypos}^\mathcal{I}_p$, and $l^\mathcal{I}_k$ are undefined for $d^{(k)}_{i,j} \in \Delta_\mathcal{I}$ with $k < 2n+2$.

It is not hard to verify that $d^{(0)}_{0,0} \in C^\mathcal{I}_{\mathcal{D},a}$. Moreover, there exist no domain elements d_1 and d_2, $d_1 \neq d_2$, such that the values $\mathsf{xpos}^\mathcal{I}_m(d_i)$, $\mathsf{ypos}^\mathcal{I}_m(d_i)$ are defined for $m \leq n$, $i = 1, 2$ and $\mathsf{xpos}^\mathcal{I}_m(d_1) = \mathsf{xpos}^\mathcal{I}_m(d_2)$, $\mathsf{ypos}^\mathcal{I}_m(d_1) = \mathsf{ypos}^\mathcal{I}_m(d_2)$ for $m \leq n$. The immediate consequence is that the interpretation $\mathcal{I}$ satisfies the functional dependencies $(\mathsf{xpos}_0, \dots, \mathsf{xpos}_n, \mathsf{ypos}_0, \dots, \mathsf{ypos}_n$ wkeyfor $\top, l_i)$ for $i \leq m$. Thus, $\mathcal{I}$ is a model of $C_{\mathcal{D},a}$ and $\mathcal{K}_{\mathcal{D},a}$. $\square$

Since the size of $C_{\mathcal{D},a}$ and $\mathcal{K}_{\mathcal{D},a}$ is polynomial in n and m, we obtain the following theorem:

Theorem 3 *The satisfiability of $\mathcal{ALC}(\mathsf{D})^{\mathcal{FD}}$-concepts w.r.t. to safe key boxes is* NExpTime-*hard.*

Now we combine the obtained result with the complexity of D-satisfiability:

Corollary 2 *There exists a concrete domain $\mathcal{D}$ such that $\mathcal{D}$-satisfiability is in* PTime *and $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$-satisfiability w.r.t. safe key boxes is* NExpTime-*hard.*

As an immediate consequence of the previous reduction we obtain the theorem:

Theorem 4 *Let $\mathcal{D}$ be a concrete domain such that $\{0,1\} \subseteq \Delta_\mathcal{D}$ and $\Phi_\mathcal{D}$ contains predicates $=_0$ and $=_1$ with $(=_0)^\mathcal{D} = \{0\}$ and $(=_1)^\mathcal{D} = \{1\}$. Then satisfiability of $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$-concepts w.r.t. safe key boxes is* NExpTime*-hard.*

Let us notice that the key box used in the reduction is a weak one, which means that dropping strong functional dependencies does not help to overcome NExpTime-hardness.

In the presented reduction, we used a very simple concrete domain and therefore we needed an n-ary key box for the reduction. If we use a more complex concrete domain D' such that $\mathbb{N} \subseteq \Delta_{\mathsf{D}'}$ and $\Phi_{\mathsf{D}'}$ contains unary predicates $=_k$ for $k \in \mathbb{N}$ with the obvious extensions and the ternary predicates $+$ and $*$, we can do the reduction by using only a unary key box. More precisely, we can introduce concrete features pos and lab such that:

$$\mathsf{pos}^\mathcal{I}(d) = \mathsf{xpos}(d) + 2^{n+1} \cdot \mathsf{ypos}(d) \text{ and } \mathsf{lab}^\mathcal{I}(d) = l(d)$$

This can be achieved by modifying the reduction as shown in [Lutz et al. 2002], and by defining $\mathsf{Tile}_i := \exists \mathsf{lab}.\ =_i$. Then it suffices to have the unary key box with a single functional dependency:

$$\{(\mathsf{pos} \text{ wkeyfor } \top, \mathsf{lab})\}$$

for the reduction. As a consequence, we obtain the following theorem:

Theorem 5 *Let $\mathcal{D}$ be a concrete domain such that $\mathbb{N} \subseteq \Delta_\mathcal{D}$ and $\Phi_\mathcal{D}$ contains the following predicates:*

1. *a unary predicate $=_k$ with $(=_k)^\mathcal{D} = \{k\}$ for each $k \in \mathbb{N}$,*
2. *a ternary predicate $+$ with $(+)^\mathcal{D} \cap \{(k_1,k_2,x) \mid k_1,k_2 \in \mathbb{N}$ and $x \in \Delta_\mathcal{D}\} = \{(k_1,k_2,k_1+k_2) \mid k_1,k_2 \in \mathbb{N}\}$, and*
3. *a ternary 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 satisfiability of $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$-concepts w.r.t. safe weak unary key boxes is NExpTime*-hard.*

This result shows that, in general, restricting us to unary key boxes would not lower the complexity of reasoning in $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$ with safe key boxes.

Chapter 4

A Reasoning Procedure

In this chapter, we will present a reasoning procedure for $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$. In Section 3.1 it was shown that the satisfiability of $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$-concepts w.r.t. general key boxes is undecidable. We will show that decidability can be regained if we restrict us to safe key boxes. The chosen reasoning procedure for $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$ with safe key boxes is the tableau algorithm presented in Section 4.1. In Section 4.2, we prove the correctness of this algorithm (termination, soundness, completeness) and show that the algorithm yields a tight NExpTime upper complexity bound for $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$ if $\mathcal{D}$ is a concrete domain such that extended D-satisfiability is in NP.

4.1 A Tableau Algorithm for $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$ with Safe Key Boxes

In this section, we present a tableau algorithm for $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$ with weak key boxes. As already said, tableau algorithms for description logics decide whether a concept is satisfiable by trying to construct a model for it. They start with an initial structure induced by the input concept and apply so-called completion rules until an obvious contradiction is detected or no rule is applicable anymore. In the latter case, the concept is satisfiable, and in the former it is not. Tableau algorithms for DLs with concrete domains work on trees which have two types of nodes: abstract ones that represent individuals of the logic domain, and concrete ones that are mapped to values of the concrete domain. If we do not want to commit to a particular concrete domain, we need a clear interface between the tableau algorithm and a concrete domain reasoner. This is often achieved by requiring that the concrete domain $\mathcal{D}$ is admissible (c.f. Definition 4), i.e., that the satisfiability of $\mathcal{D}$-conjunctions is decidable.

However, due to the functional dependencies, we need, additionally, information on which variables (concrete nodes) have to be mapped to the same concrete domain value in order to satisfy a certain $\mathcal{D}$-conjunction. Thus, for $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$, we need the stronger condition from [Lutz et al. 2003] called *key-admissibility*. Since the tableau algorithm we are going to develop for $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$ is non-deterministic, key-admissibility is defined in a non-deterministic manner:

Definition 10 (key-admissible). An admissible concrete domain $\mathcal{D}$ is *key-admissible* iff it additionally satisfies the following property:

- 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 $v, v' \in V$

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

We say that *extended $\mathcal{D}$-satisfiability* is in NP if there exists an algorithm as above running in polynomial time. $\diamondsuit$

As discussed in [Lutz et al. 2003], key-admissibility is not a significantly stronger property than admissibility, since every admissible concrete domain that provides for an equality predicate is also key-admissible. In [Lutz et al. 2002], it is shown that an algorithm for extended $\mathcal{D}$-satisfiability can be easily constructed from one for $\mathcal{D}$-satisfiability: every admissible concrete domain $\mathcal{D}$ that provides for an equality predicate, provides also for inequality. Then we can simply guess an equivalence relation $\sim$ on the set of variables used in the presented predicate conjunction c; decide the satisfiability of the conjunction $c \wedge \bigwedge_{v \sim v'} {=}(v, v') \wedge \bigwedge_{v \not\sim v'} {\neq}(v, v')$, and return clash if it is unsatisfiable and $\sim$ otherwise. Throughout this chapter, we assume that the concrete domain $\mathcal{D}$ provides for an equality. Let us now give an example of a key-admissible numerical concrete domain Q [Lutz 2003a]:

Example 2: The concrete domain Q is defined by setting Δ_{Q} to the set of rational numbers $\mathbb{Q}$ and Φ_{Q} to the smallest set containing the following predicates:

- unary predicates P_q for each $P \in \{<, \leq, =, \neq, \geq, >\}$ and $q \in \mathbb{Q}$ with $(P_q)^{\mathsf{Q}} = \{q' \in \mathbb{Q} \mid q' P q\}$;
- binary predicates $<, \leq, =, \neq, \geq, >$ with obvious extensions;
- ternary predicates $+$ and $\mp$ with $(+)^{\mathsf{Q}} = \{(q, q', q'') \in \mathbb{Q}^3 \mid q + q' = q''\}$ and $(\mp)^{\mathsf{Q}} = \mathbb{Q}^3 \setminus (+)^{\mathsf{Q}}$;
- unary predicates $\top_{\mathsf{Q}}$ and $\bot_{\mathsf{Q}}$ with $(\top_{\mathsf{Q}})^{\mathsf{Q}} = \mathbb{Q}$ and $(\bot_{\mathsf{Q}})^{\mathsf{Q}} = \emptyset$.

Moreover, in [Lutz 2003a] it is shown that Q-satisfiability is in PTime, and, thus, we conclude that extended Q-satisfiability is in NP.

Before presenting the tableau algorithm, we need some more prerequisites. A concept is in *negation normal form (NNF)* if negation occurs only in front of concept names. If $\mathcal{D}$ is a key-admissible concrete domain, then every $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$-concept can be converted into an equivalent one in NNF by exhaustively applying the following rewrite rules:

$$\neg(C \sqcap D) \rightsquigarrow \neg C \sqcup \neg D \qquad \neg(C \sqcup D) \rightsquigarrow \neg C \sqcap \neg D$$
$$\neg(\exists R.C) \rightsquigarrow \forall R.\neg C \qquad \neg(\forall R.C) \rightsquigarrow \forall R.\neg C$$
$$\neg\neg C \rightsquigarrow C$$
$$\neg(\exists u_1, ..., u_n.P) \rightsquigarrow \exists u_1, ..., u_n.\overline{P} \sqcup u_1\uparrow \cdots \sqcup u_n\uparrow$$
$$\neg(g\uparrow) \rightsquigarrow \exists g.\top_{\mathcal{D}}$$

Note that, for a path $u = f_1 \cdots f_n g$, we use $u\uparrow$ as abbreviation for the concept $\forall f_1. \cdots \forall f_n.g\uparrow$.

We use $\dot{\neg} C$ to denote the result of converting the concept $\neg C$ into NNF. A key box $\mathcal{K}$ is in NNF if all concepts occurring in functional dependencies in $\mathcal{K}$ are in NNF. From now on, we assume that all input concepts and key boxes are in NNF. Let C be an $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$-concept and $\mathcal{K}$ a key box. We use $\mathsf{sub}(C)$ to denote the set of subconcepts of C and $\mathsf{con}(\mathcal{K})$ to denote the set of concepts appearing on the right-hand side of functional dependencies in $\mathcal{K}$. For a set of concepts Γ, $\mathsf{sub}(\Gamma)$ denotes $\bigcup_{C \in \Gamma} \mathsf{sub}(C)$. We use $\mathsf{cl}(\mathcal{K})$ as abbreviation for the set

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

and $\mathsf{cl}(C, \mathcal{K})$ as abbreviation for the set

$$\mathsf{sub}(C) \cup \mathsf{cl}(\mathcal{K}).$$

Now we introduce the underlying data structure of the tableau algorithm:

Definition 11 (Completion system) Let O_a and O_c be disjoint and countably infinite sets of *abstract* and *concrete nodes*. A *completion tree* for an $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$-concept C and a key box $\mathcal{K}$ is a finite, labelled tree $T = (V_a, V_c, E, \mathcal{L})$ with nodes $V_a \cup V_c$, such that $V_a \subseteq O_a$, $V_c \subseteq O_c$, and all nodes from V_c are leaves. The tree is labelled as follows:

1. each node $a \in V_a$ is labelled 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 labelled 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 labelled with a concrete feature $\mathcal{L}(a, x)$ occurring in C or $\mathcal{K}$

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 an $x \in V_c$ is an g-successor of a if $(a, x) \in E$ and $\mathcal{L}(a, x) = g$. The notion u-successor for a path u is defined in the obvious way.

For $a \in V_a \cup V_c$, we use $\mathsf{lev}_T(a)$ to denote the depth on which a occurs in T (the root node is on depth 0). A *completion system* for an $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$-concept C and a key box $\mathcal{K}$ is a tuple $(T, \mathcal{P}, \sim)$ where

- $T = (V_a, V_c, E, \mathcal{L})$ is a completion tree for C and $\mathcal{K}$,
- $\mathcal{P}$ is a function mapping each $\mathcal{P} \in \Phi_{\mathcal{D}}$ of arity n in C to a subset of V_c^n,
- $\sim$ is an equivalence relation on V_c.

$\diamondsuit$

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

$$S_{C_0} = (T_{C_0}, \mathcal{P}_0, \emptyset)$$

with the initial completion tree

$$T_{C_0} = (\{a_0\}, \emptyset, \emptyset, \{a_0 \mapsto \{C_0\}\})$$

and where $\mathcal{P}_0$ maps each P occurring in C_0 to $\emptyset$.

The algorithm applies completion rules to the completion system until an obvious inconsistency (clash) is detected or no completion rule is applicable any more. We will define completion rules for $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$ after some prerequisites. Let us first introduce an operation that is used by completion rules to add new nodes to completion trees. The operation respects the functionality of abstract and concrete features.

Definition 12 ($\oplus$ Operation) An abstract or concrete node is called *fresh* w.r.t. a completion tree T if it does not appear in T. Let $S = (T, \mathcal{P}, \sim)$ be a completion system with $T = (V_a, V_c, E, \mathcal{L})$. We use the following operations:

- $S \oplus aRb$ ($a \in V_a$, $b \in O_a$ fresh in T, $R \in N_R$) yields a completion system obtained from S in the following way:
  - if $R \notin N_{aF}$ or $R \in N_{aF}$ and a has no R-successors, then add b to V_a, (a, b) to E and set $\mathcal{L}(a, b) = R$, $\mathcal{L}(b) = \emptyset$.
  - if $R \in N_{aF}$ and there is a $c \in V_a$ such that $(a, c) \in E$ and $\mathcal{L}(a, c) = R$ then rename c in T with b.

- $S \oplus agx$ ($a \in V_a$, $x \in O_c$ fresh in T, $g \in N_{cF}$) yields a completion system obtained from S in the following way:
  - if a has no g-successors, then add x to V_c, (a, x) to E and set $\mathcal{L}(a, x) = g$;
  - if a has a g-successor y, then rename y in T, $\mathcal{P}$, and $\sim$ with x.

Let $u = f_1 \cdots f_n g$ be a path. With $S \oplus aux$, where $a \in V_a$ and $x \in O_c$ is fresh in T, we denote the completion system obtained from S by taking distinct nodes $b_1, ..., b_n \in O_a$ which are fresh in T and setting

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

$\Diamond$

Now we define what is meant by an obvious inconsistency.

Definition 13 (Clash) Let $S = (T, \mathcal{P}, \sim)$ be a completion system for a concept C and a key box $\mathcal{K}$ with $T = (V_a, V_c, E, \mathcal{L})$. We say that the completion system S is *concrete domain satisfiable* iff the conjunction

$$\zeta_S = \bigwedge_{P \text{ used in } C} \ \bigwedge_{(x_1,...,x_n)\in\mathcal{P}(P)} P(x_1, ..., x_n) \wedge \bigwedge_{x\sim y} = (x, y)$$

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

1. there is an $a \in V_a$ and an $A \in N_C$ 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 a g-successor of a,
3. S is not concrete domain satisfiable.

If S does not contain a clash, S is called *clash-free.* $\Diamond$

In order to ensure the termination of the algorithm, we need a cycle detection mechanism, called *blocking.* Informally, we detect nodes in the completion tree "similar" to the previously created ones and "block" them. This means that the completion rules are applied only to unblocked nodes.

Definition 14 ($\approx$ relation, Blocking) Let $S = (T, \mathcal{P}, \sim)$ be a completion system for a concept C_0 and a key box $\mathcal{K}$ with $T = (V_a, V_c, E, \mathcal{L})$. Let u be a path. We say that nodes $a, b \in V_a$ have *similar u-successors* (written $a \approx_u b$) if the following holds:

- if a has a u-successor x, then b has a u-successor y and $x \sim y$;

- if b has a u-successor x, then a has a u-successor x and $x \sim y$.

With $\mathsf{suff}(C_0, \mathcal{K})$ we denote the set of all suffixes of paths that appear in a $\exists u_1, ..., u_n.P \in \mathsf{sub}(C_0)$ or in a functional dependency (either on the left- and right-hand side) in the key box $\mathcal{K}$.

We call abstract nodes a and b *similar* (written $a \approx b$) if

1. $\mathcal{L}(a) = \mathcal{L}(b)$, and
2. $a \approx_u b$ for all $u \in \mathsf{suff}(C_0, \mathcal{K})$.

An abstract node $a \in V_a$ is *directly blocked* by its ancestor $b \in V_a$ if $a \approx b$. An abstract node is *blocked* if it or one of its ancestors is directly blocked. $\diamondsuit$

The first part of the blocking condition, namely the requirement that a node and the one which directly blocks it have same labels is known from tableau algorithms for other DLs. The second part is introduced due to functional dependencies, and here is new that while blocking we consider not only abstract nodes and their labels, but also concrete ones and the concrete equivalence relation. An intuitive explanation of the blocking mechanism is given after we have introduced the completion rules. Before actually giving them, we introduce some notions. With check we refer to the function that computes a concrete equivalence for a given $\mathcal{D}$-conjunction. If ρ is a binary relation on $V_c \times V_c$, with ρ^* we denote the reflexive, symmetric and transitive closure of ρ.

Completion rules:

$R\sqcap$ if $C_1 \sqcap C_2 \in \mathcal{L}(a)$, a is not blocked, 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 $C_1 \sqcup C_2 \in \mathcal{L}(a)$, a is not blocked, 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 $\exists R.C \in \mathcal{L}(a)$, a is not blocked, and there is no R-successor of a such that $C \in \mathcal{L}(b)$, then set $S := S \oplus aRb$ for a fresh $b \in O_a$ and $\mathcal{L}(b) := \{C\}$

$R\forall$ if $\forall R.C \in \mathcal{L}(a)$, a is not blocked, and b is an R-successor of a such that $C \notin \mathcal{L}(b)$, then set $\mathcal{L}(b) := \mathcal{L}(b) \cup \{C\}$

$R\exists_c$ if $\exists u_1, ..., u_n.P \in \mathcal{L}(a)$, a is not blocked, and there exist no $x_1, ..., x_n \in V_c$ such that x_i is a u_i-successor of a for $1 \leq i \leq n$ and $(x_1, ..., x_n) \in \mathcal{P}(P)$ then set $S := (S \oplus au_1x_1 \oplus \cdots \oplus au_nx_n)$ with $x_1, ..., x_n \in O_c$ fresh and $\mathcal{P}(P) := \mathcal{P}(P) \cup \{(x_1, ..., x_n)\}$

Rch if $(u_1, ..., u_n$ wkeyfor $C, u) \in \mathcal{K}$ or $(u_1, ..., u_n$ skeyfor $C, u) \in \mathcal{K}$ and there exist $x_1, ..., x_n \in V_c$ such that x_i is a u_i-successor of a (not blocked) for $1 \leq i \leq n$ and $\{C, \dot{\neg} C\} \cap \mathcal{L}(a) = \emptyset$, then set $\mathcal{L}(a) := \mathcal{L}(a) \cup \{D\}$ for some $D \in \{C, \dot{\neg} C\}$

Rwkey if $C \in \mathcal{L}(a) \cap \mathcal{L}(b)$, $(u_1, ..., u_n$ wkeyfor $C, u) \in \mathcal{K}$, a and b are not blocked, a has u_i-successor x_i, b has u_i-successor y_i, and $x_i \sim y_i$ for $1 \leq i \leq n$, there is a u-successor x of a and a u-successor y of b, and $(x, y) \notin \sim$, then set $\sim := (\sim \cup \{(x, y)\})^*$

Rskey if $C \in \mathcal{L}(a) \cap \mathcal{L}(b)$, $(u_1, ..., u_n$ skeyfor $C, u) \in \mathcal{K}$, a and b are not blocked, a has u_i-successor x_i, b has u_i-successor y_i, and $x_i \sim y_i$ for $1 \leq i \leq n$, there is a u-successor x of a, and there is no u-successor z of b such that $(x, z) \in \sim$ then set $S := S \oplus buy$ with $y \in O_c$ fresh and $\sim := (\sim \cup \{(x, y)\})^*$

R$\sim$ if $\sim' = \mathsf{check}(\zeta_S) \not\subseteq \sim$ then set $\sim := \sim'$

Note that the *Rwkey*, *Rskey* and *R*$\sim$ rules which update the $\sim$ relation ensure that the update result is indeed a concrete equivalence relation.

If no completion rule is applicable to a completion system S, S is called *complete.*

Let us give some remarks on the completion rules. Among the rules there are two non-deterministic ones, namely $R\sqcup$ and *Rch*. The rules $R\sqcap$, $R\sqcup$, $R\exists$, $R\forall$, and $R\exists_c$ are known from the existing algorithm for $\mathcal{ALC}(\mathcal{D})$-concept satisfiability (see [Lutz 2002a]). The rules *Rch*, *Rwkey*, and *Rskey* deal with the key box.

The *Rch* is a so-called "choice" rule that does the following: If there is a functional dependency $(u_1, \ldots, u_n$ depfor $C, u) \in \mathcal{K}$ and there is an abstract node a with all appropriate u_i-successors, the rule adds non-deterministically C or $\dot{\neg} C$ to the $\mathcal{L}(a)$. This is necessary since both possibilities may have ramifications. Observe that, without blocking, this rules can cause infinite runs of the algorithm, e.g. due to a $(u$ wkeyfor $\exists R.\top, u) \in \mathcal{K}$ and "bad guessing" by *Rch*. The first part of the blocking condition deals with this problem.

The *Rwkey* rule deals with weak functional dependencies. If there is a $(u_1, \ldots, u_n$ wkeyfor $C, u) \in \mathcal{K}$ and there are two abstract nodes a and b with u_i- and u-successors such that the u_i-successors of a and b are associated with the same element of the concrete domain (i.e., related by $\sim$), the *Rwkey* rule makes sure that u-successors of a and b are also related by $\sim$.

Analogously, the *Rskey* rule deals with strong functional dependencies. The difference to *Rwkey* is that it is applied even if b does not have a u-successor. In this case, the necessary u-successor of b is created. This rule also endangers the termination of the algorithm. To see this, consider satisfiability of

$$C_0 = \exists g.{=}0 \sqcap \exists (fg).{=}0 \quad \text{w.r.t. } \mathcal{K} = \{(g \text{ skeyfor } \top, fg)\}.$$

Without blocking, applications of *Rskey* will generate an infinite f-chain such that each element has a g-successor that is zero as shown in Figure 4.1. The second part of the blocking condition is introduced to deal with this effect.

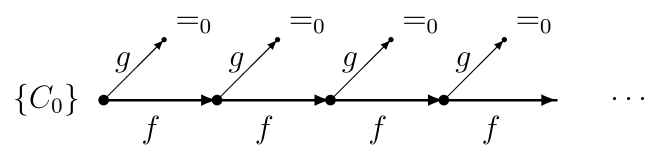

Figure 4.1: An infinite f-chain

Finally, the $R{\sim}$ rule computes an update of the concrete equivalence $\sim$ by calling the check function with argument ζ_S. The rule $R\exists_c$ adds new tuples into $\mathcal{P}(P)$ and the rules *Rwkey* and *Rskey* add new tuples into $\sim$, modifying in this way the $\mathcal{D}$-conjunction ζ_S and making it necessary to update $\sim$. The $R{\sim}$ rule does this during the run of the algorithm—in contrast to the original $\mathcal{ALC}(\mathcal{D})$ algorithm, where a single call to the concrete domain reasoner at the end of the computation is sufficient. The interleaving approach of our algorithm is essential since the equality of concrete nodes detected by the concrete domain reasoner can trigger further applications of the *Rwkey* and *Rskey* rules.

We present now the tableau algorithm in pseudo-code notation. It is started with sat(S_{C_0}).

Algorithm:

```
procedure sat($S$)
    if $S$ contains a clash then return unsatisfiable
    if $S$ is complete then return satisfiable
    $S'$ := application of a completion rule to $S$
    return sat($S'$)
```

Note that checking whether a completion system contains a clash prior to completion rule application ensures that the function check(ζ_S) in the rule $R{\sim}$ does not return clash.

4.2 Correctness of the Algorithm

Let us now prove termination, soundness and completeness of the tableau algorithm. We first introduce a few notions.

Definition 15 (Path length, Role depth)
Let $u = f_1 \cdots f_n g$ be a path. The *length* of u is defined as $|u| = n + 1$.
The *role depth* of concepts is defined inductively as follows:

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

Let C be a concept, and $\mathcal{K}$ a key box. We use $|C|$ to denote the length of C and $|\mathcal{K}|$ to denote

$$\sum_{(u_1,\ldots,u_n \text{ depfor } C,u) \in \mathcal{K}} (|u_1| + \cdots + |u_n| + |C| + |u|).$$

where depfor $\in$ {wkeyfor, skeyfor}. With $\mathsf{mpl}(C_0, \mathcal{K})$ we denote

$$\max\{|u| \mid u \in \mathsf{suff}(C_0, \mathcal{K})\}$$

Lemma 3 *Let $T = (V_a, V_c, E, \mathcal{L})$ be a completion tree constructed during the run of the tableau algorithm started on the input concept C_0 and a safe key box $\mathcal{K}$. Let $a \in V_a$ and $C \in \mathcal{L}(a)$. Then either*

$$C \in \mathsf{cl}(\mathcal{K}) \text{ or } \mathsf{lev}_T(a) \leq \mathsf{rd}(C_0) - \mathsf{rd}(C).$$

Proof. We prove the lemma by induction on the number n of rule applications after which C is added into $\mathcal{L}(a)$.

1. $n = 0$. Obvious, since $C \equiv C_0$ and a is the root node, i.e., $\mathsf{lev}_T(a) = 0$.

2. $n \geq 1$. Assume $C \notin \mathsf{cl}(\mathcal{K})$. Since $C \in \mathsf{cl}(C_0, \mathcal{K})$ and $\mathsf{cl}(C_0, \mathcal{K}) = \mathsf{sub}(C_0) \cup \mathsf{cl}(\mathcal{K})$, it must be that $C \in \mathsf{sub}(C_0)$. We will distinguish two cases.

- C is added to $\mathcal{L}(a)$ via an application of $R\sqcup$ or $R\sqcap$ rule. Then there exists an $E \in \mathcal{L}(a)$, with $E = C \sqcup D$ or $E = C \sqcap D$ for some concept $D \in \mathsf{cl}(C_0, \mathcal{K})$. We have that $C \notin \mathsf{cl}(\mathcal{K})$ implies $E \notin \mathsf{cl}(\mathcal{K})$. Since E is added to $\mathcal{L}(a)$ by an earlier rule application, by inductional hypothesis we get $\mathsf{lev}_T(a) \leq \mathsf{rd}(C_0) - \mathsf{rd}(E)$. Using $\mathsf{rd}(E) = \mathsf{rd}(C)$, we obtain $\mathsf{lev}_T(a) \leq \mathsf{rd}(C_0) - \mathsf{rd}(C)$.

- C is added to $\mathcal{L}(a)$ via an application of $R\exists$ or $R\forall$ rule. Then there is a $b \in V_a$ such that a is an R-successor of b and there is a $D \in \mathcal{L}(b)$ with $D \in \{\exists R.C, \forall R.C\}$. Obviously, $D \notin \mathsf{cl}(\mathcal{K})$. Since D is added to $\mathcal{L}(b)$ by an earlier rule application, by inductional hypothesis we get $\mathsf{lev}_T(b) \leq \mathsf{rd}(C_0) - \mathsf{rd}(D)$. Using $\mathsf{rd}(D) = \mathsf{rd}(C) + 1$ and $\mathsf{lev}_T(b) = \mathsf{lev}_T(a) - 1$, we obtain $\mathsf{lev}_T(a) \leq \mathsf{rd}(C_0) - \mathsf{rd}(C)$. $\square$

Lemma 4 *Let $S = (T, \mathcal{P}, \sim)$ with $T = (V_a, V_c, E, \mathcal{L})$ be a completion system constructed during the run of the tableau algorithm started on the input concept C_0 and a safe key box $\mathcal{K}$. Then the following holds:*

(a) *The out-degree of T is bounded by $|C_0| + |\mathcal{K}|$;*

(b) *The concrete equivalence relation $\sim$ satisfies:*

$$|V_c/_\sim| \leq \#\{c \in V_c \mid \mathsf{lev}_T(c) \leq \mathsf{rd}(C_0)\} \leq (|C_0| + |\mathcal{K}|)^{|C_0|};$$

(c) *$\approx$ is an equivalence relation on V_a and*

$$|V_a/_\approx| \leq (n+1)^{|C_0|+|\mathcal{K}|} \cdot 2^{|\mathsf{cl}(C_0,\mathcal{K})|}, \text{ where } n = |V_c/_\sim|;$$

(d) *The depth of T is bounded by $|V_a/_\approx| + \mathsf{mpl}(C_0, \mathcal{K})$.*

Proof.

(a) New nodes are created exclusively due to application of the rules $R\exists$, $R\exists_c$, and *Rskey*. The rule $R\exists$ generates at most one successor for each $\exists R.C \in \mathsf{sub}(C_0)$, and the rule $R\exists_c$ generates at most one successor for every abstract or concrete feature appearing in some $\exists u_1, ..., u_n.P \in \mathsf{sub}(C_0)$. Hence, the number of successors of an $a \in V_a$ created by applying these two rules to a is bounded by $|C_0|$. Moreover, the rule *Rskey* generates at most one successor for every abstract or concrete feature appearing in $\mathcal{K}$. Thus, the number of successors created due to the application of *Rskey* is bounded by $|\mathcal{K}|$. Therefore, the number of successors of an arbitrary abstract node a is bounded by $|C_0| + |\mathcal{K}|$.

(b) In order to prove (b), we first show that the following holds:

$$(\forall c \in V_c)(\exists c' \in V_c)(\mathsf{lev}_T(c') \leq \mathsf{rd}(C_0) \ \wedge \ c \sim c') \qquad (*)$$

Note that concrete nodes are created only due to the application of the rules $R\exists_c$ and *Rskey*. Let us define $\mathsf{rang}(c)$ for $c \in V_c$ as:

$$\mathsf{rang}(c) := \begin{cases} 0, & \text{if } c \text{ is created via an application of } R\exists_c \\ i, & \text{if } c \text{ is created via the i-th application of } \textit{Rskey} \end{cases}$$

Now we prove $(*)$ by strong induction on $\mathsf{rang}(c)$:

1. $\mathsf{rang}(c) = 0$. Then c is created due to the application of the $R\exists_c$ rule to a node $a \in V_a$ and a concept $C \in \mathcal{L}(a)$, $C = \exists u_1, ..., u_n.P$. Since $\mathcal{K}$ is safe, $C \notin \mathsf{cl}(\mathcal{K})$ and according to Lemma 3, this implies $\mathsf{lev}_T(a) \leq \mathsf{rd}(C_0) - \mathsf{rd}(C)$. Due to the definition of $\mathsf{rd}(\exists u_1, ..., u_n.P)$, we have $\mathsf{lev}_T(c) - \mathsf{lev}_T(a) \leq \mathsf{rd}(C)$ and

combining these two inequalities, we obtain: $\mathsf{lev}_T(c) \leq \mathsf{rd}(C_0)$. Since $c \sim c$, the base case is proved.

2. $\mathsf{rang}(c) = n, n \geq 1$. In this case, c is created due to the application of the *Rskey* rule to nodes $a, b \in V_a$ and a strong functional dependency $(u_1, ..., u_n \text{ skeyfor } C, u) \in \mathcal{K}$. The node a has a u-successor x created due to the application of $R\exists_c$ or an earlier application of *Rskey*. The node c is u-successor of b, $c \sim x$ and $\mathsf{rang}(x) < \mathsf{rang}(c) = n$. By induction hypothesis, there is a $c' \in V_c$ such that $x \sim c'$ and $\mathsf{lev}_T(c') \leq \mathsf{rd}(C_0)$. Due to the transitivity of $\sim$, we get $c \sim c'$ and the induction step is proved.

According to (a), the out-degree of T is bounded by $|C_0| + |\mathcal{K}|$. Using the fact that $\mathsf{rd}(C_0) \leq |C_0|$ and that concrete nodes are leaves of the tree T, we get:

$$\#\{c \in V_c \mid \mathsf{lev}_T(c) \leq \mathsf{rd}(C_0)\} \leq (|C_0| + |\mathcal{K}|)^{|C_0|};$$

Finally, combining this result with $(*)$ we complete the proof of (b).

(c) Using the definition of $\approx$ and the fact that $\sim$ is an equivalence relation, it is easy to see that $\approx$ is also an equivalence relation on V_a. Due to (b), $V_c/_\sim$ is of bounded size. Let $V_c/_\sim = \{v_1, v_2, \ldots, v_n\}$. Furthermore, $\mathsf{suff}(C_0, \mathcal{K})$ is of bounded size and $m = |\mathsf{suff}(C_0, \mathcal{K})| \leq |C_0| + |\mathcal{K}|$. Let us define a mapping $\phi : V_a \to 2^{\mathsf{suff}(C_0,\mathcal{K}) \times V_c/_\sim}$ in the following way:

$$\phi(a) = \{(u, v_k) \mid u \in \mathsf{suff}(C_0, \mathcal{K}), a \text{ has a } u\text{-successor } x, \text{ and } x \in v_k\}$$

Then the following holds:

$$a \approx b \quad \text{iff} \quad \phi(a) = \phi(b) \text{ and } \mathcal{L}(a) = \mathcal{L}(b) \tag{$**$}$$

Due to the definition of $\oplus$, the functionality of paths is respected and therefore, there are no $v_i, v_j \in V_c/_\sim$, $i \neq j$ such that $\{(u, v_i), (u, v_j)\} \subseteq \phi(a)$ for some $a \in V_a$ and $u \in \mathsf{suff}(\mathcal{K})$. Thus, we have

$$|\{\phi(a) \mid a \in V_a\}| \leq \sum_{i=0}^{m} \binom{m}{i} \cdot n^i.$$

In the above sum, every argument $\binom{m}{i} \cdot n^i$ corresponds to the situation when an abstract node has exactly i successors for paths from $\mathsf{suff}(C_0, \mathcal{K})$. Since $\sum_{i=0}^{m} \binom{m}{i} \cdot n^i = (n+1)^m$, and $m \leq |C_0| + |\mathcal{K}|$ we get $|\{\phi(a) \mid a \in V_a\}| \leq (n+1)^{|C_0|+|\mathcal{K}|}$. Finally, using $(**)$ and the fact that $|\{\mathcal{L}(a) \mid a \in V_a\}| \leq 2^{|\mathsf{cl}(C_0,\mathcal{K})|}$ we obtain

$$|V_a/_\approx| \leq |\{\phi(a) \mid a \in V_a\}| \cdot |\{\mathcal{L}(a) \mid a \in V_a\}| \leq (n+1)^{|C_0|+|\mathcal{K}|} \cdot 2^{|\mathsf{cl}(C_0,\mathcal{K})|}$$

(d) Let $M = |V_a/_\approx|$. Assume that there is a node $a \in V_a \cup V_c$ such that $\mathsf{lev}_T(a) > M + \mathsf{mpl}(C_0, \mathcal{K})$. Then a is created due to the application of a completion rule to an ancestor $b \in V_a$ of a for which it holds that $k = \mathsf{lev}_T(b) > M$.

Then there is sequence of abstract nodes $a_0, a_1, \ldots, a_k$ such that a_i is a successor of a_{i-1} for $1 \leq i \leq k$, a_0 is the root node and $a_k = b$. Since $k > M$, we have that $a_i \approx a_j$ for some i, j with $0 \leq i < j \leq k$. This means that b is blocked and contradicts the assumption that a completion rule was applied to b. $\square$

Lemma 5 *(Termination) When started with an $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$ concept C_0 and a safe key box $\mathcal{K}$, both in NNF, the tableau algorithm terminates.*

Proof. The tableau algorithm terminates due to the following reasons:

- It constructs a finitely labelled completion tree T of bounded out-degree and depth (by Lemma 4 (a) and (d)) in a monotonic way, i.e., no nodes are removed from T, and no concepts are removed from node labels.

- The concrete equivalence relation $\sim$ is updated only finitely often: Let $\sim_1$, $\sim_2$, ... be the sequence of concrete equivalences computed during the run of the tableau algorithm. The relation $\sim$ is updated exclusively due to the application of the rules *Rwkey*, *Rskey*, and $R\sim$. By the definition of these rules and ζ_S, we have:

$$\sim_1 \subsetneq \sim_2 \subsetneq \cdots$$

  Moreover, from Lemma 4 (a) and (d) we can conclude that the number of the concrete nodes $\#V_c$ is finite, more precisely, double-exponential in $|C_0| + |\mathcal{K}|$. Since $\sim \subseteq V_c \times V_c$, the relation $\sim$ is updated no more than $(\#V_c)^2$ times.

- $\mathcal{P}$ is updated only finitely often. This can be shown reasoning similarly as in the case of $\sim$.

$\square$

The tableau algorithm algorithm we presented runs in 2NExpTime. This is due to the fact that the algorithm is non-deterministic and it constructs a completion tree whose number of nodes is (in the worst case) double-exponential in $|C_0| + |\mathcal{K}|$ (since the depth of the tree may be exponential – c.f. Lemma 4 (d)).

Now we prove that the tableau algorithm is sound, i.e., that if it returns satisfiable then the input concept C_0 and key box $\mathcal{K}$ have a common model $\mathcal{I}$. This model can be constructed by "putting" all unblocked abstract nodes of the constructed completion tree into the interpretation domain $\Delta_{\mathcal{I}}$. However, in the soundness proof that we will present, we do not put *all* unblocked abstract nodes into the domain, but rather use a filtration technique and in this way obtain a model whose size is only exponential in $|C_0| + |\mathcal{K}|$. This will help us, along with the completeness lemma, to show a bounded model property of $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$ which induces a NExpTime upper complexity bound.

Lemma 6 *(Soundness) If the tableau algorithm returns* satisfiable, *then the input concept C_0 is satisfiable w.r.t. the input safe key box $\mathcal{K}$ in a model whose size is not greater than M, where*

$$M = ((|C_0| + |\mathcal{K}|)^{|C_0|} + 1)^{|C_0|+|\mathcal{K}|} \cdot 2^{|\mathsf{cl}(C_0,\mathcal{K})|}.$$

Proof. If the tableau algorithm returns satisfiable, then there exists a complete and clash-free completion system $S = (T, \mathcal{P}, \sim)$ for C_0 and $\mathcal{K}$. Let $T = (V_a, V_c, E, \mathcal{L})$. Since the rule $R\sim$ is not applicable and S does not contain a clash, we have that $\mathsf{check}(\zeta_S) \subseteq \sim$. By the definition of ζ_S, it always holds that $\sim \subseteq \mathsf{check}(\zeta_S)$. Therefore, we have that $\sim = \mathsf{check}(\zeta_S)$ and there exists a solution δ for ζ_S such that

$$\delta(x) = \delta(y) \text{ iff } x \sim y.$$

We now use S and δ to construct an interpretation $\mathcal{I}$. In order to get an exponential bound on the size of the interpretation domain, we introduce a linear ordering $\prec$ on V_a such that $a \prec b$ implies $\mathsf{lev}_T(a) \leq \mathsf{lev}_T(b)$. Using $\prec$, we make sure that $\Delta_\mathcal{I}$ contains no more than one representative of each $V_a/_\approx$ (c.f. Lemma 4):

$$\Delta_\mathcal{I} = \{a \in V_a \mid a \text{ is not blocked and there is no unblocked } b \in V_a \text{ such that } a \approx b \text{ and } b \prec a\}$$

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

$$R^\mathcal{I} = \{(a,b) \in \Delta_\mathcal{I} \times \Delta_\mathcal{I} \mid \text{ there is a } b' \in V_a \text{ such that } b \approx b' \text{ and } b' \text{ is an } R\text{-successor of } a\}, \text{ for all } R \in N_R$$

$$g^\mathcal{I} = \{(a, \delta(x)) \in \Delta_\mathcal{I} \times \Delta_\mathcal{D} \mid x \text{ is } g\text{-successor of } a\}, \text{ for all } g \in N_{cF}$$

$\mathcal{I}$ is well defined since $f^\mathcal{I}$ is functional for each $f \in N_{aF}$ and $g^\mathcal{I}$ is functional for each $g \in N_{cF}$. This follows from the definition of the operation $\oplus$.

In order to show that $\mathcal{I}$ is a model for C_0 and $\mathcal{K}$, we prove the following claims:

Claim 1: If $a \in V_a$ is unblocked or directly blocked, then there exists an $a' \in \Delta_\mathcal{I}$ such that $a \approx a'$.

Proof: Let us define $\mathsf{p}(a)$ for $a \in V_a$ as follows:

$$\mathsf{p}(a) := \#\{b \in V_a \mid b \prec a\}$$

We prove the claim by induction on $\mathsf{p}(a)$:

1. $\mathsf{p}(a) = 0$. Then a is the root node and $a \in \Delta_{\mathcal{I}}$. Since $a \approx a$, the base case is proved.

2. $\mathsf{p}(a) = n, n \geq 1$. We distinguish two cases:

- a is unblocked and there is no unblocked $b \in V_a$ with $a \approx b$ and $b \prec a$. By definition of $\Delta_{\mathcal{I}}$, we have that $a \in \Delta_{\mathcal{I}}$. Due to the reflexivity of $\approx$, it holds that $a \approx a$.
- There is an unblocked $\bar{a} \in V_a$ such that $a \approx \bar{a}$ and $\bar{a} \prec a$ (this includes the case when a is directly blocked). Then $\mathsf{p}(\bar{a}) < \mathsf{p}(a)$ and by inductional hypothesis there exists an $a' \in \Delta_{\mathcal{I}}$ such that $\bar{a} \approx a'$. The transitivity of $\approx$ implies $a \approx a'$.

Claim 2: If $a \in \Delta_{\mathcal{I}}$ and $u \in \mathsf{suff}(C_0, \mathcal{K})$, then the following holds:

(a) If $u^{\mathcal{I}}(a)$ is defined, then a has a u-successor x and $u^{\mathcal{I}}(a) = \delta(x)$.

(b) If a has a u-successor x, then $u^{\mathcal{I}}(a) = \delta(x)$.

Proof: We prove (a) and (b) by induction on the length of u:

(a) $|u| = 1$: True by construction of $\mathcal{I}$.

$|u| = n+1, n \geq 1$: Let $u = f_1 \cdots f_n g$. Since $u^{\mathcal{I}}(a)$ is defined, by definition of $\mathcal{I}$ there is a $b \in \Delta_{\mathcal{I}}$ such that $f_1^{\mathcal{I}}(a) = b$ and $(f_2 \cdots f_n g)^{\mathcal{I}}(b)$ is defined. By induction hypothesis, b has an $f_2 \cdots f_n g$-successor y and $(f_2 \cdots f_n g)^{\mathcal{I}}(b) = \delta(y)$. Furthermore, $f_1^{\mathcal{I}}(a) = b$ implies that there is a $b' \in V_a$ which is an f_1-successor of a and $b \approx b'$. By definition of $\approx$ (since $f_2 \cdots f_n g \in \mathsf{suff}(C_0, \mathcal{K})$), b' has an $f_2 \cdots f_n g$-successor x such that $y \sim x$. Then x is a u-successor of a and $u^{\mathcal{I}}(a) = \delta(y) = \delta(x)$.

(b) $|u| = 1$: True by construction of $\mathcal{I}$.

$|u| = n+1, n \geq 1$: Let $u = f_1 \cdots f_n g$ and $b \in V_a$ an f_1-successor of a. Then x is an $f_2 \cdots f_n g$-successor of b. Since $a \in \Delta_{\mathcal{I}}$, b is either unblocked or directly blocked. Due to Claim 1, there is a $b' \in \Delta_{\mathcal{I}}$ such that $b \approx b'$ and $b' \prec b$. By definition of $\mathcal{I}$ we have that $f_1^{\mathcal{I}}(a) = b'$. By definition of $\approx$, b' has a $f_2 \cdots f_n g$-successor y such that $x \sim y$. Since $b' \in \Delta_{\mathcal{I}}$ and $f_2 \cdots f_n g \in \mathsf{sub}(C_0, \mathcal{K})$, by induction hypothesis we obtain $(f_2 \cdots f_n g)^{\mathcal{I}}(b') = \delta(y)$. Finally, we get $u^{\mathcal{I}}(a) = \delta(y) = \delta(x)$.

Claim 3: For all $a \in \Delta_{\mathcal{I}}$ and $C \in \mathsf{cl}(C_0, \mathcal{K})$, if $C \in \mathcal{L}(a)$, then $a \in C^{\mathcal{I}}$.

Proof: We prove the claim by structural induction:

- C is a concept name. By construction of $\mathcal{I}$.
- $C = \neg D$. Since C is in NNF, D is a concept name. Clash-freeness of S implies $D \notin \mathcal{L}(a)$. The construction of $\mathcal{I}$ implies $a \notin D^{\mathcal{I}}$ which yields $a \in (\neg D)^{\mathcal{I}}$.
- $C = D \sqcap E$. The completeness of S implies $\{D, E\} \subseteq \mathcal{L}(a)$. The induction hypothesis yields $a \in D^{\mathcal{I}}$ and $a \in E^{\mathcal{I}}$, therefore $a \in (D \sqcap E)^{\mathcal{I}}$.
- $C = D \sqcup E$. The completeness of S implies $\{D, E\} \cap \mathcal{L}(a) \neq \emptyset$. By induction hypothesis it holds that $a \in D^{\mathcal{I}}$ or $a \in E^{\mathcal{I}}$, and therefore $a \in (D \sqcup E)^{\mathcal{I}}$.
- $C = \exists R.D$. Since the $R\exists$ rule is not applicable, a has an R-successor b such that $D \in \mathcal{L}(b)$. Then b is either unblocked or directly blocked. Due to Claim 1, there is a $b' \in \Delta_{\mathcal{I}}$ such that $b \approx b'$. Due to the definition of $\approx$, we have $\mathcal{L}(b) = \mathcal{L}(b')$ and therefore $D \in \mathcal{L}(b')$. By induction hypothesis, it holds that $b' \in D^{\mathcal{I}}$. By the definition of $\mathcal{I}$, we have $(a, b') \in R^{\mathcal{I}}$ and this implies $a \in C^{\mathcal{I}}$.
- $C = \forall R.D$. Let $(a, b) \in R^{\mathcal{I}}$. By construction of $\mathcal{I}$, there is an R-successor b' of a with $b \approx b'$. Non-applicability of $R\forall$ yields $D \in \mathcal{L}(b')$. Due to the definition of $\approx$, we have $\mathcal{L}(b) = \mathcal{L}(b')$ and therefore $D \in \mathcal{L}(b')$. By induction hypothesis, we get $b \in D^{\mathcal{I}}$. Since this holds independently of the choice of b, we obtain $a \in C^{\mathcal{I}}$.
- $C = \exists u_1, ..., u_n.P$. Since the $R\exists_c$ rule is not applicable, there exist $x_1, ..., x_n \in V_c$ such that x_i is a u_i-successor of a for $1 \leq i \leq n$ and $(x_1, ..., x_n) \in \mathcal{P}(P)$. By Claim 1 we get $u_i^{\mathcal{I}}(a) = \delta(x_i)$ for $1 \leq i \leq n$. Since $(x_1, ..., x_n) \in \mathcal{P}(P)$ and δ is a solution for ζ_S, we have $(\delta(x_1), ..., \delta(x_n)) \in P^{\mathcal{D}}$ 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 a g-successor of a. Thus, by the construction of $\mathcal{I}$, there is no $\alpha \in \Delta_{\mathcal{D}}$ such that $(a, \alpha) \in g^{\mathcal{I}}$.

Since C_0 is in the label of the root node a_0 and $a_0 \in \Delta_{\mathcal{I}}$, Claim 3 implies that $\mathcal{I}$ is a model of C_0. We use Claim 2 and Claim 3 to show that $\mathcal{I}$ satisfies all strong and weak functional dependencies in $\mathcal{K}$:

- $\mathcal{I}$ satisfies all $(u_1, ..., u_n$ skeyfor $C, u) \in \mathcal{K}$: take $a, b \in C^{\mathcal{I}}$ such that $u_i^{\mathcal{I}}(a) = u_i^{\mathcal{I}}(b)$ for $1 \leq i \leq n$ and $u^{\mathcal{I}}(a)$ is defined. Non-applicability of Rch yields $\{C, \dot\neg C\} \cap \mathcal{L}(a) \neq \emptyset$. If $\dot\neg C \in \mathcal{L}(a)$, then Claim 2 implies $a \in (\dot\neg C)^{\mathcal{I}}$ which contradicts $a \in C^{\mathcal{I}}$. Therefore, $C \in \mathcal{L}(a)$ and for the same reason $C \in \mathcal{L}(b)$. Using the Claim 2(a) and the fact that $u_i^{\mathcal{I}}(a)$ and $u_i^{\mathcal{I}}(b)$ are defined for

$1 \leq i \leq n$, we conclude that a has a u_i-successor x_i and b a u_i-successor y_i such that $u_i^{\mathcal{I}}(a) = \delta(x_i)$ and $u_i^{\mathcal{I}}(b) = \delta(y_i)$ for $1 \leq i \leq n$. Moreover, $\delta(x_i) = \delta(y_i)$ implies $x_i \sim y_i$ for $1 \leq i \leq n$. Reasoning similarly, we get that a has a u-successor x such that $u^{\mathcal{I}}(a) = \delta(x)$. Non-applicability of *Rskey* yields that b has a u-successor y such that $x \sim y$. Using Claim 2(b), we get that $u^{\mathcal{I}}(b) = \delta(y)$, and since $\delta(x) = \delta(y)$ we obtain $u^{\mathcal{I}}(a) = u^{\mathcal{I}}(b)$.

- $\mathcal{I}$ satisfies all $(u_1, ..., u_n \text{ wkeyfor } C, u) \in \mathcal{K}$: take $a, b \in C^{\mathcal{I}}$ such that $u_i^{\mathcal{I}}(a) = u_i^{\mathcal{I}}(b)$ for $1 \leq i \leq n$ and $u^{\mathcal{I}}(a)$ and $u^{\mathcal{I}}(b)$ are defined. Reasoning similarly as in the previous case, we obtain $C \in \mathcal{L}(a) \cap \mathcal{L}(b)$. Using Claim 2(a) and the definition of δ, we conclude that a has a u_i-successor x_i and b a u_i-successor y_i such that $x_i \sim y_i$ for $1 \leq i \leq n$. Moreover, a has a u-successor x and b has a u-successor y. Non-applicability of *Rwkey* yields that $x \sim y$. Using Claim 2(b), we get that $u^{\mathcal{I}}(a) = \delta(x)$ and $u^{\mathcal{I}}(b) = \delta(y)$, and since $\delta(x) = \delta(y)$ we obtain $u^{\mathcal{I}}(a) = u^{\mathcal{I}}(b)$.

Let us now determine the size of $\Delta_{\mathcal{I}}$. From the construction of $\mathcal{I}$ it follows that $|\Delta_{\mathcal{I}}| \leq |V_a/_{\approx}|$. Using Lemma 4, we obtain

$$|V_a/_{\approx}| \leq ((|C_0| + |\mathcal{K}|)^{|C_0|} + 1)^{|C_0| + |\mathcal{K}|} \cdot 2^{|\mathsf{cl}(C_0, \mathcal{K})|}.$$

□

Finally, we prove that the tableau algorithm is complete.

Lemma 7 *(Completeness) If the input concept C_0 is satisfiable w.r.t. the safe 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 tableau algorithm in such a way that it constructs a complete and clash-free completion system. A completion system $S = (T, \mathcal{P}, \sim)$ with $T = (V_a, V_c, E, \mathcal{L})$ is called *$\mathcal{I}$-compatible* if there exist mappings $\pi : V_a \rightarrow \Delta_{\mathcal{I}}$ and $\tau : V_c \rightarrow \Delta_{\mathcal{D}}$ such that:

(C1) $C \in \mathcal{L}(a) \Rightarrow \pi(a) \in C^{\mathcal{I}}$

(C2) b is a R-successor of $a \Rightarrow (\pi(a), \pi(b)) \in R^{\mathcal{I}}$

(C3) x is a g-successor of $a \Rightarrow g^{\mathcal{I}}(\pi(a)) = \tau(x)$

(C4) $(x_1, ..., x_n) \in \mathcal{P}(P) \Rightarrow (\tau(x_1), ..., \tau(x_n)) \in P^{\mathcal{D}}$

(C5) $x \sim y \Rightarrow \tau(x) = \tau(y)$

We prove the following claim to show that completion rules can be applied in such a way that $\mathcal{I}$-compatibility is preserved.

Claim 1: 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 functions satisfying conditions (C1)-(C5), and R a completion rule applicable to S. We make a case analysis according to the type of R.

$R\sqcap$ The rule is applied to $C_1 \sqcap C_2 \in \mathcal{L}(a)$. By (C1), $C_1 \sqcap C_2 \in \mathcal{L}(a)$ 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 $\mathcal{L}(a)$, the obtained completion system is $\mathcal{I}$-compatible via the same π and τ.

$R\sqcup$ The rule is applied to $C_1 \sqcup C_2 \in \mathcal{L}(a)$. By (C1), $C_1 \sqcup C_2 \in \mathcal{L}(a)$ implies $\pi(a) \in (C_1 \sqcup C_2)^{\mathcal{I}}$ and hence $\pi(a) \in C_1^{\mathcal{I}}$ or $\pi(a) \in C_2^{\mathcal{I}}$. Since the rule adds C_1 or C_2 to $\mathcal{L}(a)$, it can be applied in such a way that the obtained completion system is $\mathcal{I}$-compatible via the same π and τ.

$R\exists$ The rule is applied to $\exists R.C \in \mathcal{L}(a)$. By (C1), $\pi(a) \in (\exists R.C)^{\mathcal{I}}$ and hence there exists a $d \in \Delta_{\mathcal{D}}$ such that $(\pi(a), d) \in R^{\mathcal{I}}$ and $d \in C^{\mathcal{I}}$. Since the rule adds a new R-successor b of a and sets $\mathcal{L}(b) = \{D\}$, the resulting completion system is $\mathcal{I}$-compatible via $\pi' = \pi \cup \{b \mapsto d\}$ and τ.

$R\forall$ The rule is applied to $\forall R.C \in \mathcal{L}(a)$ and it adds C to the label $\mathcal{L}(b)$ of an existing R-successor of a. By (C1), $\pi(a) \in (\forall R.C)^{\mathcal{I}}$ and by (C2), $(\pi(a), \pi(b)) \in R^{\mathcal{I}}$. Therefore, $\pi(b) \in C^{\mathcal{I}}$ and the resulting completion system is $\mathcal{I}$-compatible via π and τ.

$R\exists_c$ The rule is applied to $\exists u_1, ..., u_n.P \in \mathcal{L}(a)$ 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 concrete nodes $x^{(i)}$ (or reuses existing ones and renames them) 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 $2 \leq j \leq k_i$,
- $x^{(i)}$ is $g^{(i)}$-successor of $b_{k_i}^{(i)}$ for $1 \leq i \leq n$, and
- $(x^{(1)}, ..., x^{(n)}) \in \mathcal{P}(P)$.

Due to (C1), $\pi(a) \in (\exists u_1, ..., u_n.P)^{\mathcal{I}}$ and therefore there exist $d_j^{(i)} \in \Delta_{\mathcal{I}}$ for $1 \leq i \leq n$ and $1 \leq j \leq k_i$ and $\alpha_1, ..., \alpha_n \in \Delta_D$ such that:

– $(\pi(a), d_1^{(i)}) \in (f_1^{(i)})^{\mathcal{I}}$ for $1 \leq i \leq n$,
– $(d_j^{(i)}, d_{j-1}^{(i)}) \in (f_j^{(i)})^{\mathcal{I}}$ for $1 \leq i \leq n$ and $2 \leq j \leq k_i$,
– $(g^{(i)})^{\mathcal{I}}(d_{k_i}^{(i)}) = \alpha_i$ for $1 \leq i \leq n$, and
– $(\alpha_1, ..., \alpha_n) \in P^{\mathcal{D}}$.

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

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

.

Rch The rule is applied to an abstract node a and a functional dependency $(u_1, ..., u_n \text{ depfor } C, u) \in \mathcal{K}$ where depfor $\in$ {skeyfor, wkeyfor} and non-deterministically adds C or $\dot\neg C$ to $\mathcal{L}(a)$. By definition of semantics, $\pi(a) \in C^{\mathcal{I}}$ or $\pi(a) \in (\dot\neg C)^{\mathcal{I}}$. Therefore, *Rch* can be applied in such a way that the obtained completion system is $\mathcal{I}$-compatible via π and τ.

Rwkey The rule is applied to nodes $a, b \in V_a$ and a functional dependency $(u_1, ..., u_n \text{ wkeyfor } C, u) \in \mathcal{K}$. Then $C \in \mathcal{L}(a) \cap \mathcal{L}(b)$, a has u_i-successor x_i, b has u_i-successor y_i, and $x_i \sim y_i$ for $1 \leq i \leq n$, a has a u-successor x and b has a u-successor y. Due to (C1), $\pi(a), \pi(b) \in C^{\mathcal{I}}$. Due to (C2) and (C3), $u_i^{\mathcal{I}}(a) = u_i^{\mathcal{I}}(b)$ for $1 \leq i \leq n$, $u^{\mathcal{I}}(a) = \tau(x)$ and $u^{\mathcal{I}}(b) = \tau(y)$. Since $\mathcal{I}$ is a model of $\mathcal{K}$, $u^{\mathcal{I}}(a) = u^{\mathcal{I}}(b)$ which implies $\tau(x) = \tau(y)$. The rule sets $\sim := (\sim \cup (x, y))^*$. Obviously, (C5) is satisfied and the resulting completion system is $\mathcal{I}$-compatible via π and τ.

Rskey The rule is applied to nodes $a, b \in V_a$ and a functional dependency $(u_1, ..., u_n \text{ skeyfor } C, u) \in \mathcal{K}$ with $u = f_1 \cdots f_n g$. Then $C \in \mathcal{L}(a) \cap \mathcal{L}(b)$, a has u_i-successor x_i, b has u_i-successor y_i, and $x_i \sim y_i$ for $1 \leq i \leq n$ and a has a u-successor x. Due to (C1), $\pi(a), \pi(b) \in C^{\mathcal{I}}$. Due to (C2) and (C3), $u_i^{\mathcal{I}}(a) = u_i^{\mathcal{I}}(b)$ for $1 \leq i \leq n$, and $u^{\mathcal{I}}(a) = \tau(x)$. Since $\mathcal{I}$ is a model of $\mathcal{K}$, $u^{\mathcal{I}}(b)$ is defined and $u^{\mathcal{I}}(a) = u^{\mathcal{I}}(b)$. Hence, there exist $d_1, ..., d_n \in \Delta_{\mathcal{I}}$ and $\alpha \in \Delta_D$ such that: $(\pi(b), d_1) \in f_1^{\mathcal{I}}$, $(d_{i-1}, d_i) \in f_i^{\mathcal{I}}$ for $2 \leq i \leq n$ and $g^{\mathcal{I}}(d_n) = \alpha = \tau(x)$. The rule application generates new abstract nodes $b_1, ..., b_n$ and a new concrete node y (or reuses existing ones and renames them) such that: b_1 is an f_1-successor of b, b_i is an f_i-successor of b_{i-1} for $2 \leq i \leq n$ and y is g-successor of b_n. The rule also sets $\sim := (\sim \cup (x, y))^*$. If we set $\pi' := \pi \cup \bigcup_{1\leq i\leq n} \{b_i \mapsto d_i\}$ and $\tau' := \tau \cup \{y \mapsto \alpha\}$, the resulting completion system is $\mathcal{I}$-compatible via π' and τ'.

$R\!\sim$ The properties (C4) and (C5) imply that τ is a solution for the predicate conjunction ζ_S. We guide the non-deterministic check function such that, when given ζ_S and set of variables $V_c \subseteq O_c$ as input, it returns the relation $\sim'$ defined by setting $x \sim' y$ iff $\tau(x) = \tau(y)$ for all $x, y \in V_c$. Obviously, $\sim'$ is a concrete equivalence relation. The rule sets $\sim := \sim'$, the condition (C4) and (C5) is satisfied, and the resulting completion system is $\mathcal{I}$-compatible via π and τ.

With the second claim, we show that $\mathcal{I}$-compatibility implies clash-freeness:

Claim 2: Every $\mathcal{I}$-compatible completion system is clash-free.

Proof: Let $S = (T, \mathcal{P}, \sim)$ with $T = (V_a, V_c, E, \mathcal{L})$ be an $\mathcal{I}$-compatible completion system. We show that S is clash-free by case distinction:

- Assume that there is an $a \in V_a$ and $A \in N_C$ such that $\{A, \neg A\} \subseteq \mathcal{L}(a)$. Due to (C1), $\pi(a) \in A^{\mathcal{I}}$ and $\pi(a) \in (\neg A)^{\mathcal{I}}$, which is a contradiction.
- Assume that there are $a \in V_a$ and $x \in V_c$ such that x is g-successor of a and $g\uparrow \in \mathcal{L}(a)$. Due to (C1) and (C3), $g^{\mathcal{I}}(\pi(a)) = x$ and $\pi(a) \in (g\uparrow)^{\mathcal{I}}$, which is a contradiction.
- According to the properties (C4) and (C5), τ is a solution for ζ_S. Thus, S is concrete domain satisfiable.

Let $S_{C_0} = (T_{C_0}, \mathcal{P}_0, \emptyset)$ with $T_{C_0} = (\{a_0\}, \emptyset, \emptyset, \{a_0 \mapsto \{C_0\}\})$ be the initial completion system. Obviously, S_{C_0} is $\mathcal{I}$-compatible. According to Claim 1, we can apply completion rules in such a way that $\mathcal{I}$-compatibility is preserved. By Lemma 5, the algorithm always terminates and by Claim 2, no clash will be found and the algorithm returns $\mathsf{satisfiable}$. $\square$

As an immediate consequence of Lemmas 5, 6 and 7, we get the following theorem:

Theorem 6 *If $\mathcal{D}$ is a key-admissible concrete domain, the tableau algorithm decides satisfiability of $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$-concepts w.r.t. safe key boxes.*

Lemmas 6 and 7 yield a *bounded-model property* for $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$: if an $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$-concept C_0 is satisfiable w.r.t. a safe key box $\mathcal{K}$, Lemma 7 implies that the tableau algorithm returns $\mathsf{satisfiable}$. Then Lemma 6 implies that C_0 and $\mathcal{K}$ have a model $\mathcal{I}$ such that $|\Delta_{\mathcal{I}}| \leq \mathsf{b}(C_0, \mathcal{K})$, where

$$\mathsf{b}(C_0, \mathcal{K}) = ((|C_0| + |\mathcal{K}|)^{|C_0|} + 1)^{|C_0| + |\mathcal{K}|} \cdot 2^{|\mathsf{cl}(C_0, \mathcal{K})|} \tag{4.1}$$

Since $|\mathsf{cl}(C_0, \mathcal{K})|$ is linear in $|C_0| + |\mathcal{K}|$, there is an $m \in \mathbb{N}$ such that $\mathsf{b}(C_0, \mathcal{K}) \leq 2^{(|C_0| + |\mathcal{K}|)^m}$. To sum up, every $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$-concept C_0 that is satisfiable w.r.t. $\mathcal{K}$

has a model $\mathcal{I}$ of size $|\Delta_{\mathcal{I}}| \leq 2^{(|C_0|+|\mathcal{K}|)^m}$. This observation yields a NExpTime upper complexity bound:

Theorem 7 *If $\mathcal{D}$ is a key-admissible concrete domain and extended $\mathcal{D}$-satisfiability is in* NP*, then $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$-concept satisfiability w.r.t. safe key boxes is in* NExpTime.

Proof. Let us consider an alternative algorithm for deciding whether a given $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$-concept C_0 is satisfiable w.r.t. a safe key box $\mathcal{K}$. We first introduce the notion of a quasi-model:

Let $\mathsf{b}(C_0, \mathcal{K})$ be defined as in (4.1) and $V = \{v_1, \ldots, v_n\}$ be a finite set of variables, where $n \leq (|C_0| + |\mathcal{K}|) \cdot \mathsf{b}(C_0, \mathcal{K})$. A *quasi-interpretation* for C_0 and $\mathcal{K}$ is a tuple $\mathcal{M} = (\Delta_{\mathcal{M}}, \mathcal{P}^{\mathcal{M}}, \cdot^{\mathcal{M}})$, where

- $\Delta_{\mathcal{M}}$ is a non-empty set such that $|\Delta_{\mathcal{M}}| \leq \mathsf{b}(C_0, \mathcal{K})$;
- $\mathcal{P}^{\mathcal{M}}$ is a function mapping each predicate P of arity n used in C_0 to a subset of V^n, and an equality predicate $=$ to a subset of V^2;
- $\cdot^{\mathcal{M}}$ is a quasi-interpretation function which maps
  - each concept name C to a subset $C^{\mathcal{M}}$ of $\Delta_{\mathcal{M}}$,
  - each role name R to a subset $R^{\mathcal{M}}$ of $\Delta_{\mathcal{M}} \times \Delta_{\mathcal{M}}$,
  - each abstract feature f to a partial function $f^{\mathcal{M}}$ from $\Delta_{\mathcal{M}}$ to $\Delta_{\mathcal{M}}$, and
  - each concrete feature g to a partial function $g^{\mathcal{M}}$ from $\Delta_{\mathcal{M}}$ to V.

The quasi-interpretation function is extended to paths and arbitrary concepts in the same way as interpretation functions, with the following exception:

$$\begin{aligned}(\exists u_1, \ldots, u_n.P)^{\mathcal{M}} \ := \ \{d \in \Delta_{\mathcal{M}} \mid \exists x_1, \ldots, x_n \in V : u_i^{\mathcal{M}}(d) = x_i \\ \text{and } (x_1, \ldots, x_n) \in \mathcal{P}^{\mathcal{M}}(P)\}\end{aligned}$$

A quasi-interpretation $\mathcal{M}$ is called a *quasi-model* for C_0 and $\mathcal{K}$ if it satisfies the following conditions:

(C1) the finite predicate conjunction

$$\zeta_{\mathcal{M}} = \bigwedge_{(x_1,\ldots,x_n)\in\mathcal{P}^{\mathcal{M}}(P)} P(x_1, \ldots, x_n) \wedge \bigwedge_{(x,y)\in\mathcal{P}^{\mathcal{M}}(=)} =(x, y) \wedge \bigwedge_{(x,y)\notin\mathcal{P}^{\mathcal{M}}(=)} \neq(x, y)$$

is satisfiable;

(C2) there exists an $a \in \Delta_{\mathcal{M}}$ such that $a \in C_0^{\mathcal{M}}$;

(C3) the functional dependencies from $\mathcal{K}$ are satisfied via the mapping of $=$ by $\mathcal{P}^{\mathcal{M}}$.

It is not difficult to see that there is a quasi-model for an $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$-concept C_0 and a safe key box $\mathcal{K}$ iff C_0 and $\mathcal{K}$ are satisfiable. This follows immediately from the definition of quasi-models, the bounded-model property for $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$ w.r.t. safe key boxes obtained in Lemma 6, and the fact that the number of different concrete features appearing in C_0 and $\mathcal{K}$ is bounded by $|C_0| + |\mathcal{K}|$.

Now we present an alternative decision procedure for $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$ with safe key boxes, based on quasi-models. We first "guess" a quasi-interpretation for C_0 and $\mathcal{K}$ (clearly, there are only finitely many such quasi-interpretations), and then we check whether it is a quasi-model for C_0 and $\mathcal{K}$. The latter can be done via the following algorithm:

Algorithm: Let $D_1, \ldots, D_m$ be all concepts from $\mathsf{cl}(C_0, \mathcal{K})$, listed in order of length. Thus we have that if D_i is a subconcept of D_j, then $i < j$. The algorithm labels every node a of $\Delta_{\mathcal{M}}$ with $\mathcal{L}(a)$ – a set of concepts from $\mathsf{cl}(C_0, \mathcal{K})$. Initially, all node labels $\mathcal{L}(a)$ are set to the empty set. In the i-th step of the algorithm, $1 \leq i \leq m$, the following rule is applied to all $a \in \Delta_{\mathcal{M}}$:

- if $D_i = A$ for A a concept name, and $a \in A^{\mathcal{M}}$, then add A to $\mathcal{L}(a)$;
- if $D_i = \neg A$ for A a concept name, and $a \notin A^{\mathcal{M}}$, then add the tuple $\neg A$ to $\mathcal{L}(a)$;
- if $D_i = \exists u_1, \ldots, u_n.P$ and $a \in (D_i)^{\mathcal{M}}$, then add $\exists u_1, \ldots, u_n.P$ to $\mathcal{L}(a)$;
- if $D_i = g\uparrow$ and $g^{\mathcal{M}}(a)$ undefined, then add $g\uparrow$ to $\mathcal{L}(a)$;
- if $D_i = B \sqcap C$ and $\{B, C\} \subseteq \mathcal{L}(a)$, then add $B \sqcap C$ to $\mathcal{L}(a)$;
- if $D_i = B \sqcup C$ and $\{B, C\} \cap \mathcal{L}(a) \neq \emptyset$, then add $B \sqcup C$ to $\mathcal{L}(a)$;
- if $D_i = \exists R.B$, and there exists a $b \in \Delta_{\mathcal{M}}$ such that $(a, b) \in R^{\mathcal{M}}$ and $B \in \mathcal{L}(b)$, then add $\exists R.B$ to $\mathcal{L}(a)$;
- if $D_i = \forall R.B$, and for all $b \in \Delta_{\mathcal{M}}$ such that $(a, b) \in R^{\mathcal{M}}$, it holds that B in $\mathcal{L}(b)$, then add $\forall R.B$ to $\mathcal{L}(a)$;

An easy induction shows that, after the m-th step of the algorithm, the following holds: $a \in D^{\mathcal{M}}$ for $D \in \mathsf{cl}(C_0, \mathcal{K})$ iff $D \in \mathcal{L}(a)$. Moreover, every step of the algorithm can be carried out in time $O(|\Delta_{\mathcal{M}}|)$, which is at most exponential in $|C_0| + |\mathcal{K}|$. Since the number of steps $m = |\mathsf{cl}(C_0, \mathcal{K})|$ is linear in $|C_0| + |\mathcal{K}|$, the algorithm can be carried out in time exponential in $|C_0| + |\mathcal{K}|$.

Now we can check whether $\mathcal{M}$ satisfies conditions (C2) and (C3): (C2) is satisfied iff there is an $a \in \Delta_{\mathcal{M}}$ such that $C_0 \in \mathcal{L}(a)$; similarly, (C3) is satisfied if for every functional dependency $(u_1, \ldots, u_n$ depfor $C, u)$, every two nodes $a, b \in \Delta_{\mathcal{M}}$ such that $C \in \mathcal{L}(a) \cap \mathcal{L}(b)$ and $(u_i^{\mathcal{M}}(a), u_i^{\mathcal{M}}(b)) \in \mathcal{P}^{\mathcal{M}}(=)$ $(i = 1..n)$, have "correct" u-successors. Obviously, (C2) and (C3) can be checked in exponential time, since $|\Delta_{\mathcal{M}}|$ is exponential in $|C_0| + |\mathcal{K}|$ and the number of functional dependencies in $\mathcal{K}$ is not greater than $|\mathcal{K}|$.

Finally, we can employ a concrete domain reasoner to check whether $\zeta_{\mathcal{M}}$ is satisfiable (C1). Alternatively, if a concrete domain $\mathcal{D}$ does not provide for an equality and inequality predicate, the algorithm for extended $\mathcal{D}$-satisfiability can be employed to check the satisfiability of the $\mathcal{D}$-conjunction

$$\bigwedge_{(x_1,\ldots,x_n) \in \mathcal{P}^{\mathcal{M}}(P)} P(x_1, \ldots, x_n) \wedge \bigwedge_{v \in V} \top_{\mathcal{D}}(v).$$

If successful, it returns an equivalence relation $\sim \subseteq V \times V$, and in exponential time it can be checked whether $\sim = \mathcal{P}^{\mathcal{M}}(=)$.

Since all described parts of the (non-deterministic) algorithm run in exponential time, and every $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$-concept C_0 and a safe key box $\mathcal{K}$ have a model of size bounded by $\mathsf{b}(C_0, \mathcal{K})$, we conclude that $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$-concept satisfiability w.r.t. safe key boxes can be decided in NExpTime if extended $\mathcal{D}$-satisfiability is in NP. $\square$

Together with the lower complexity bound from Theorem 4 we get the following result:

Theorem 8 *If $\mathcal{D}$ is a key-admissible concrete domain and extended $\mathcal{D}$-satisfiability is in* NP*, then $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$-concept satisfiability w.r.t. safe key boxes is* NExpTime*-complete.*

Since the concept subsumption can be reduced to the concept unsatisfiability in the standard way, we obtain that the $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$-concepts subsumption is co-NExpTime-complete.

Chapter 5

Conclusion

In this thesis, we have completed the investigation of description logics with concrete domains and key constraints, by considering a type of keys called functional dependencies. Uniqueness key constraints have already been extensively analyzed in [Lutz et al. 2003], in combination with DLs with concrete domains.

Let us summarize the results obtained in these two investigations. In [Lutz et al. 2003] it is proved that providing $\mathcal{ALC}(\mathcal{D})$– the basic DL with concrete domains – with uniqueness constraints leads to undecidability in the general case. The decidability can be preserved if the considered key boxes are Boolean, namely if they contain only concepts which are Boolean combinations of concept names. Even in this restricted form, uniqueness constraints produce dramatic jumps in the complexity of reasoning – from PSpace-complete to NExpTime-complete. Uniqueness constraints turned ut to be a very powerful expressive means, since they can "capture" nominals and thus have a strong effect on the structure of logical models.

Quite surprisingly, we were able to show in this thesis that seemingly weaker functional dependencies have an equally severe impact on decidability and complexity of reasoning. In their weak form, functional dependencies only allow to state constraints on concrete data and do not affect the logical objects. However, it turns out that providing $\mathcal{ALC}(\mathcal{D})$ even only with weak functional dependencies leads to undecidability. In order to regain decidability, we have found a safe class of key boxes, in which we disallow the concepts that have subconcepts of the form $\exists u_1, \ldots, u_n.P$. However, even safe key boxes with weak functional dependencies affect dramatically the complexity of reasoning - $\mathcal{ALC}(\mathcal{D})$ equipped with such key boxes is shown to be NExpTime-hard. The matching upper complexity bound is obtained with help of a tableau algorithm for $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$ with safe key boxes. Due to strong functional dependencies, which might influence the structure of the logical domain, we had to come up with a complicated blocking mechanism in order to ensure the termination of the algorithm; this was unnecessary in the case of uniqueness constraints. After the investigation we performed in this thesis, we can conclude that both types of keys produce the similar complexity jumps from PSpace-completeness to undecidability/ NExpTime-completeness when added to $\mathcal{ALC}(\mathcal{D})$.

For the future work, it would be interesting to combine both uniqueness constraints and functional dependencies in a single DL with concrete domains. We believe that the upper complexity bounds can be preserved and that Boolean key boxes can be replaced with less restricted safe ones.

It would be also useful to integrate functional dependencies into more expressive DLs with concrete domains. For example, it would be interesting to consider $\mathcal{SHOQ}(\mathcal{D})$, since this logic has found applications in the semantic web as underlying logic for web ontology languages. Moreover, functional dependencies can be combined with other extensions of DLs with concrete domains, for example with inverse roles or acyclic TBoxes.

Finally, it would be a challenge to implement the tableau algorithm for $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$ we presented in this thesis. We believe that it is amenable to known optimization techniques and thus can be implemented efficiently.

Bibliography

[Areces et al. 1999] C. Areces, P. Blackburn, and M. Marx. A road-map on complexity for hybrid logics. In *Proceedings of CSL'99*, volume 1683 of *LNCS*, pages 307-321, Springer-Verlag, 1999.

[Baader et al. 2003a] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P.F. Patel Schneider, Eds. *The Description Logic Handbook: Theory, Implementation, and Applications.* Cambridge University Press, 2003.

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

[Baader et al. 2003b] F. Baader, I. Horrocks, and U. Sattler. Description logics as ontology languages for the semantic web. In *Festschrift in honor of Jörg Siekmann*, D. Hutter and W. Srephan, editors, Lecture Notes in Artificial Intelligence. Springer-Verlag, 2003.

[Baader & Sattler 2001] F. Baader and U. Sattler. An overview of tableau algorithms for description logics. *Studia Logica*, 69:5-40, 2001.

[Borgida & Weddell 1997] A. Borgida and G.E. Weddell. Adding uniqueness constraints to description logics (preliminary report). In *Proceedings of the 5th International Conference on Deductive and Object-Oriented Databases (DOOD97)*, volume 1341 of *LNCS*, pages 85-102, Springer, 1997.

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

[Calvanese et al. 2001] D. Calvanese, G. De Giacomo, and M. Lenzerini. Identification Constraints and Functional Dependencies in Description Logics. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI'01)*, pages 155-160, Morgan Kaufmann, 2001.

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

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

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

[Lutz 2002c] C. Lutz. Reasoning about entity relationship daigrams with complex attribute dependencies. In *Proceedings of the International Workshop on Description Logics 2002 (DL2002)*, number 53 in CEUR-WS (http://ceur-ws.org/), pages 185-194, 2002.

[Lutz 2003a] C. Lutz. Description logics with concrete domains - a survey. In *Advances in Modal Logics (AiML) Volume 4*, pages 265-296, King's College Publications, 2003.

[Lutz 2003b] C. Lutz. NEXPTIME-complete description logics with concrete domains. *ACM Transactions on Computational Logic, volume V*, pages 1-36, 2003.

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

[Lutz et al. 2003] C. Lutz, C. Areces, I. Horrocks, and U. Sattler. Keys, nominals, and concrete domains. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI'03)*, pages 349-354, Morgan Kaufmann, 2003.

[Lutz & Sattler 2001] C. Lutz and U. Sattler. The complexity of reasoning with boolean modal logics. In F. Wolter, H. Wansing, M. de Rijke, and M. Zakharyaschev, editors, *Advances in Modal Logics 3*, CSLI Publications, Stanford, 2001.

[Haarslev & Möller 2001] V. Haarslev and R. Möller. RACER system description. In *IJCAR-01*, vol. 2083 of *LNAI*. Springer-Verlag, 2001.

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

[Horrocks 1998] I. Horrocks. Using an Expressive Description Logic: FaCT or Fiction? In *Proceedinigs of the 6th International Conference on Principles of Knowledge Representation and Reasoning (KR'98)*, Morgan Kaufmann, 1998.

[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 *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI 2002)*, 2002.

[Horrocks & Sattler 1999] I. Horrocks and U. Sattler. A description logic with transitive and inverse roles and role hierarchies. *Journal of Logic and Computation*, 9(3):385-410, 1999.

[Horrocks et al. 1999] I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for expressive description logics. In *Proceedings of of the 6th International Conference on Logic for Programming and Automated Reasoning (LPAR'99)*, volume 1705 of *LNAI*, pages 161-180, Springer-Verlag, 1999.

[Khizder et al. 2001] V. L. Khizder, D. Toman, and G.E. Weddell. On decidability and complexity of decription logics with unique constraints. In *Proceedings of the 8th International Conference on Database Theory (ICDT 2001)*, volume 1973 of *LNCS*, pages 54-67, Springer, 2001.

[Minsky 1975] M. L. Minsky. A framework for representing knowledge. In Winston, editor, *The Psychology of Computer Vision*, pages 211-277. McGraw-Hill, 1975.

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

[Schmidt-Schauß & Smolka 1991] M. Schmidt-Schauß and G. Smolka. Attributive concept decsriptions with complements. *Artificial Intelligence*, 48(1):1-26, 1991.

[Quillian 1968] R. M. Quillian. Semantic Memory. In Minsky, editor, *Semantic Information Processing*, pages 216-270. MIT Press, 1968.

Statement of Academical Honesty

I hereby declare that I have not used any auxiliary sources for my thesis work other than have been cited in my thesis.

Dresden, 27$^{\text{th}}$ February 2004

Maja Miličić
Matr No: 2879792