TECHNISCHE UNIVERSITÄT DRESDEN

INTERNATIONAL CENTER FOR COMPUTATIONAL LOGIC

Master's Thesis on

Two Types of Key Constraints in Description Logics with Concrete Domains

by

Dũng Dinh-Khac

born on 31st August 1980 in Hanoi, Vietnam

Advisor: Maja Miličić

Supervising Professor: Prof. Dr. Franz Baader

Dresden, September 2006

Technische Universität Dresden

Author:	**Dung Dinh-Khac**
Matrikel-Nr.:	**3172493**
Title:	**Two Types of Key Constraints in Description Logics with Concrete Domains**
Degree:	**Master of Science**
Date of submission:	**14/09/2006**

Declaration

Hereby I certify that the thesis has been written by me and I have not used any auxiliary sources for my thesis work other than those have been cited.

Signature of Author

Abstract

Description Logics (DLs) are a family of logic-based knowledge representation formalisms for representing and reasoning about conceptual knowledge. To represent and reason about concrete qualities of real-world entities such as size, duration, or amounts, DLs are equipped with concrete domains. Interestingly, DLs and DLs with concrete domains are useful in many applications, such as modelling database schemas and the semantic web.

Recently, it has been suggested that the expressive power of DLs with concrete domains can be further extended by adding database-like key constraints. In database schemas, key constraints can be a source of additional inconsistencies, which DLs used in reasoning about database schemas should be able to capture. Up to now, two different types of key constraints, namely uniqueness constraints and funtional dependencies, have been considered in the context of DLs with concrete domains. Surprisingly, to the best of our knowledge the two types of key constraints have not been investigated in one single logic despite the fact that in some applications, both of them are needed.

In this paper, we consider the first description logic with concrete domains, uniqueness constraints, and functional dependencies. It it obtained by extending the description logic $\mathcal{ALC}(\mathcal{D})$ (the basic propositional closed description logic $\mathcal{ALC}$ equipped with a concrete domain $\mathcal{D}$) with key boxes consisting of key constraints. More precisely, we analyze the impact of the presence of both types of key constraints on decidability and complexity of reasoning. Following from previous results, uniqueness constraints and functional dependencies bring undecidability in the general case and in order to preserve decidability we need to consider a slightly restricted form of key constraints. In the restricted form, we are able to show that reasoning w.r.t. both types of key constraints is not harder than reasoning w.r.t. each of them individually. Furthermore, several extensions with acyclic TBoxes, general TBoxes, and inverse roles are also discussed.

Contents

Acknowledgements

First of all, I would like to express my gratitude and appreciation to Maja Miličić, my scientific advisor. Without her advice, I could not have done this thesis in time. Additionally, I feel grateful for her patience in discussing and giving me lots of useful criticisms.

I am very thankful to Prof. Franz Baader, for giving very motivating lectures "Logic-based Knowledge Representation" and "Complexity and Logic" from which I gain background knowledge for my thesis, and Prof. Steffen Hölldobler, for his support and advice during my study. Besides, many thanks go to Dr. Khang Tran-Dinh, my former supervisor in Hanoi University of Technology, who introduced to me the world of theoretical computer science and without his constant support, I could not have joint the CL Programme.

Moreover, I would like to acknowledge the German Academic Exchange Service (DAAD) for the scholarship I received during my study. This scholarship has made my life in Dresden and especially my study much easier.

I must not forget to mention my friends, particularly Yusri, my roomate, and Sebastian, my football partner, who always bring me joy in their own way.

Most importantly, I would like to thank my parents, my sister, and Phuong for understanding, sharing, encouraging without ever pushing, and giving me infinite love.

Chapter 1

Introduction

Description Logics (DLs) are a family of logic-based knowledge representation formalisms for representing and reasoning about conceptual knowledge in a structured and semantically well-understood manner [2, 4]. In DLs, knowledge is represented based on the notion of *concept descriptions*, which are built from atomic *concept names* (unary predicates) and *role names* (binary predicates) by means of concept and role constructors provided by a particular description logic. For example, the basic propositionally closed description logic is called $\mathcal{ALC}$ (Attribute Language with Complements) [24]. Intuitively, the following $\mathcal{ALC}$-concept:

$$\mathsf{Professor} \sqcap \exists \mathsf{interested\_in}.\mathsf{DLs} \sqcap \forall \mathsf{give\_lecture}.\neg \mathsf{Database}$$

describes "The set of professors who are interested in DLs and do not give any lecture on Database subject". In the above expression, Professor, DLs, and Database are atomic concepts, whereas interested_in and give_lecture are roles.

Based on $\mathcal{ALC}$, more expressive description logics are built by considering addtional expressiveness means. Among those, the following are considered important from both theoretical and practical point of view: nominals [1]; number restrictions [3]; inverse and transitive roles, and role hierarchies [11]. Moreover, DLs are usually equipped with components expressing terminological knowledge, namely *TBoxes*, and assertional knowledge, namely *ABoxes*.

Most DLs are actually fragments of first order logic where the expressivity is reduced in order to ensure decidability of reasoning. In particular, $\mathcal{ALC}$ corresponds to a first order logic fragment with only two variables, e.g., the above concept can be translated into the following first order logic formula:

$$\begin{aligned} \mathsf{Professor}(x) \wedge \exists y.(&\mathsf{interested\_in}(x,y) \wedge \mathsf{DLs}(y)) \\ &\wedge \forall y.(\mathsf{give\_lecture}(x,y) \rightarrow \neg \mathsf{Database}(y)) \end{aligned}$$

To deduce implicit knowledge from the explicitly represented one, knowledge representation systems based on DLs provide various reasoning services. The most important ones are *satisfiability* and *subsumption* of concepts. Informally, a concept is said to be satisfiable if it is consistent whereas deciding concept subsumption means to determine subconcept-superconcept relationships. In propositionally closed description logics, those that provide all Boolean connectives, concept satisfiability can be reduced to concept subsumption and vice

versa. Therefore, in such logics it suffices to develop a procedure deciding concept satisfiability. Usually, satisfiability problem is solved by a *tableau algorithm* [6]. In general, tableau algorithms decide satisfiability of a concept by trying to construct a model for it. Considering complexity of reasoning in various DLs, it becomes harder as expressivity of DLs grows. For example, it is well-known that reasoning in $\mathcal{ALC}$ is PSPACE-complete, but reasoning in extensions of $\mathcal{ALC}$ is PSPACE-, EXPTIME-, NEXPTIME-complete, and even undecidable.

In the last decade, DLs have been used in various applications including reasoning about database conceptual models expressed in entity-relationship diagrams or object-oriented schemas [9] and reasoning about ontologies for the semantic web [4]. However, it turns out that even very expressive description logics are not expressive enough for applications in which it is necessary to represent information of concrete or quantitative nature, such as age, numbers, or durations. For example, we want to describe employees whose age is below 55 and whose income is greater than 50000 Euros/year. In order to do this, obviously it is needed to represent natural numbers (e.g., $55, 50000$) and compare them. Moreover, in reasoning about database schemas, concrete "datatypes" are needed to capture integrity constraints.

The standard way of integrating numbers and other datatypes into description logics is to extend DLs with so-called *concrete domains* [3, 17]. Informally, a concrete domain $\mathcal{D}$ consists of a set and predicates associated with a fixed extension over this set. The integration of concrete domains into description logics is achieved by adding

1. *abstract features*, which are functional roles;
2. *concrete features*, which are (partial) functions that map logical objects to values from the concrete domain;
3. a concrete domain-based concept constructor of the form $\exists u_1, \ldots, u_n.P$, in which u_i, called *paths*, are sequences $f_1 \ldots f_k g$ of k abstract features $f_1, \ldots, f_k$ followed by a concrete feature g and P is an n-ary predicate from the concrete domain.

The logic obtained by integrating a concrete domain $\mathcal{D}$ with $\mathcal{ALC}$ is called $\mathcal{ALC}(\mathcal{D})$. For example, using a concrete domain $\mathcal{D}$ based on natural numbers, the following concept

$$\mathsf{Employee} \sqcap \exists \mathsf{age}.{<_{55}} \sqcap \exists(\mathsf{income}), (\mathsf{spouse\ income}).{>}$$

describes employees whose age is below 55 and whose income is greater than their spouses' income. In the example, Employee is a concept, spouse is an abstract feature, age, income are concrete features, and $<_{55}, >$ respectively are unary and binary predicates with the obvious extension. Besides, the term $\exists(\mathsf{income}), (\mathsf{spouse\ income}).{>}$ is an instance of the concrete domain-based concept constructor.

Concerning complexity of reasoning, it is proved in [15] that adding concrete domains to $\mathcal{ALC}$ does not make reasoning harder, i.e., reasoning in $\mathcal{ALC}(\mathcal{D})$ is PSPACE-complete, if reasoning in the concrete domain $\mathcal{D}$ is in PSPACE. Some further extensions of $\mathcal{ALC}(\mathcal{D})$ are considered in [14].

Unsurprisingly, extending description logics with concrete domains is very useful in applications. In reasoning about conceptual database models, concrete

domains are necessary since most databases store "concrete" data, e.g., numbers and strings, and the used description logic should be able to capture integrity constraints on such data which are part of the conceptual model [16]. Furthermore, as description logics are languagues for describing ontologies, concrete domains are also important. For example, to construct an employee ontology, we need a proper way to formulate concrete datatypes such as age, salary, hiring years. In fact, the basic description logic $\mathcal{SHOQ}(\mathcal{D})$ [12] underlying the web ontology language OWL is equipped with concrete datatypes.

Recently, it has been proposed in [19, 20] to further extend the expressive power of DLs with concrete domains by adding *key constraints*, a popular notion in databases. It is known that in database schemas, key constraints play an important role, and thus DLs used in reasoning about conceptual database models should be able to capture such constraints. Among various types of key constraints, the most important ones are uniqueness constraints, which describe a set of properties uniquely identifying objects, and functional dependencies, which describe that a property is functionally determined by a set of other properties. For example,

> *Employees are uniquely identified by their branches' IDs and personel-IDs*

is a uniqueness constraint whereas the following is a functional dependency:

> *The extra pay for Microsoft's employees is detemined by their positions and the amount of overtime they have worked.*

In literature, DLs with uniqueness constraints and functional dependencies have been analyzed in [7, 8, 13, 25, 26], but in these DLs, concrete domains have not been taken into account. In the context of DLs with concrete domains, uniqueness constraints and functional dependencies have separatedly been considered in [19] and [20] respectively. Formally, in [19] DLs with concrete domains are equipped with *key boxes* consisting of uniqueness constraints of the form

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

and in [20] $\mathcal{ALC}(\mathcal{D})$ is extended with key boxes containing functional dependencies of the form

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

in which $u_1, \ldots, u_n$ and u are paths, and C is a concept. The former key constraint says that each pair of instances of the class C that have the same u_i-value, $1 \leq i \leq n$, must be the same individual, whereas the latter states that each pair of instances of the class C that have the same u_i-value, $1 \leq i \leq n$, must have the same u-value.

It has been shown that uniqueness constraints and functional dependencies have a dramatic impact on decidability and complexity of reasoning. In the general case, although satisfiability of $\mathcal{ALC}(\mathcal{D})$-concepts is PSPACE-complete, adding either uniqueness constraints or functional dependencies makes it undecidable. Decidability is recovered if we restrict ourselves to a certain form of key boxes. For DLs with concrete domains and functional dependencies, decidability is preserved when we disallow sub-concepts of the form $\exists u_1, \ldots, u_n.P$ in concepts in key boxes whereas when uniqueness constraints are considered, the condition

is even more strict. In [19, 20], decidability of reasoning in DLs with concrete domains and restricted forms of key constraints is shown via tableau algorithms. Although decidability is preserved, adding restricted uniqueness constraints or functional dependencies makes reasoning become NExpTime-complete.

Although in many applications both types of key constraints are needed, for example an employee ontology may need to represent both key constraints mentioned earlier, surprisingly, to the best of our knowledge they have not been treated in a single logic. That fact motivates us to integrate both uniqueness constraints and functional dependencies into DLs with concrete domains.

Moreover, as discussed in [19], there exists a close connection between uniqueness constraints and so-called *nominals*, i.e., concept names that have at most one instance, such as US_President. It is not hard to see that uniqueness constraints can "simulate" nominals: for example, if we use a concrete domain based on natural numbers that has unary predicates $=_n$ for equality with $n \in \mathbb{N}$, then the uniqueness constraint (g keyfor $\top$), where $\top$ stands for logical truth, obviously makes the concept $\exists g.=_3$ behave like a nominal. For this reason, we explicitly enrich our logic with nominals as they cannot bring more "harm" to complexity of reasoning than uniqueness constraints do.

Our aim is to perform an analysis on the impact of the presence of both types of key constraints on decidability and computational complexity of reasoning. Intuitively, reasoning becomes undecidable in the general case and in order to regain decidability, a restriction on key boxes is needed. Although restricting key boxes recovers decidability, it makes reasoning become NExpTime-complete, thus not harder than that when only one type of key constraints is considered.

The rest of the paper is organized as follows.

In Chapter 2, we formally introduce the syntax and semantics of the logic $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$, which is obtained by adding uniqueness constraints, functional dependencies, and nominals to $\mathcal{ALC}(\mathcal{D})$. Besides, we give the formal definition of safe key boxes, which help us regain decidability of reasoning. Furthermore, in Section 2.2 we discuss the undecidability result of reasoning in the general case, and give an NExpTime lower complexity bound for reasoning w.r.t. safe key boxes in $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$.

In Chapter 3, it is shown that admitting only safe key boxes ensures that reasoning in $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$ is decidable. First, decidability is shown in Section 3.1 by a tableau algorithm deciding $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$-concept satisfiability w.r.t. safe key boxes. It turns out that a naive combination of the tableau algorithms, which deal with uniqueness constraints and functional dependencies in [19] and [20] respectively, does not work. Instead, the two types of key constraints need to be treated carefully as both uniqueness constraints and strong functional dependencies may influence the structure of logical models. Then, termination, soundness, and completeness of the algorithm are proved in Section 3.2. Moreover, these proofs yield an NExpTime upper complexity bound for $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$-concept satisfiability w.r.t. safe key boxes, which coincides with the NExpTime lower complexity bound in Chapter 2.

In Chapter 4, we give a discussion on extending $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$ with acyclic TBoxes, general TBoxes, and inverse roles. We first show that adding acyclic TBoxes does not make reasoning in $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$ harder. Then we give a discusssion about the fact that for a large class of concrete domains, adding either general TBoxes or inverse roles leads to undecidability of reasoning.

Finally, Chapter 5 gives concluding remarks.

Chapter 2

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

In this chapter, we introduce the description logic $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$ with concrete domains, uniqueness constraints, and functional dependencies. First, its syntax, semantics, and reasoning problems in it are defined in Section 2.1. And then, in Section 2.2 we discuss lower complexity bounds for $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$.

2.1 $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$

In this section, syntax and semantics of the description logic $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$ are formally introduced. We start with the definition of a concrete domain [3, 17].

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

Based on concrete domains, the description logic $\mathcal{ALCOK}(\mathcal{D})$ with concrete domains and uniqueness constraints is proposed in [19], whereas the description logic $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$ which includes concrete domains and functional dependencies is treated in [20]. In the following, we combine the two logics together by taking into account both uniqueness constraints and functional dependencies in one logic. First, we define $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$-concepts, functional dependencies, uniquess constraints, and key boxes.

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

- *every concept name and every nominal is a concept, and*

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

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

A weak functional dependency *is an expression*

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

a strong functional dependency *is an expression*

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

a uniqueness constraint *is an expression*

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

where $u_1, \ldots, u_k$ ($k \geq 1$) and u are paths, and C is a concept. A finite set of functional dependencies and uniqueness constraints is called a key box.

As usual, we use $\top$ as abbreviation for an arbitrary propositional tautology, $\bot$ as abbreviation for $\neg\top$, and $C \to D$ as abbreviation for $\neg C \sqcup D$. It is easily seen that both $\mathcal{ALCOK}(\mathcal{D})$ and $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$ are fragments of the description logic $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$. Beside the two fragments, throughout this paper, we will also consider several fragments of $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$ such as $\mathcal{ALCO}(\mathcal{D})$, obtained from $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$ by admitting empty key boxes, $\mathcal{ALCK}(\mathcal{D})$ and $\mathcal{ALC}(\mathcal{D})$ obtained from $\mathcal{ALCOK}(\mathcal{D})$ and $\mathcal{ALCO}(\mathcal{D})$ respectively by disallowing nominals, etc.

Before actually giving the formal semantics of $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$, let us consider the following examples showing uniqueness constraints and functional dependencies together in applications.

Example 1.

1. *In a bookstore ontology, we assume that bookstores are uniquely identified by their identification numbers, which can be expressed by the following uniqueness constraint:*

$$(\mathsf{storeID} \textit{ keyfor } \mathsf{Bookstore})$$

whereas "All books with the same ISBN and sold in the same bookstore have the same price" can be modelled by

$$(\mathsf{isbn}, \mathsf{sold\_in}\ \mathsf{storeID} \textit{ skeyfor } \mathsf{Book}, \mathsf{price})$$

where Bookstore, Book *are concept names,* sold_in *is an abstract feature, and* isbn, storeID *and* price *are concrete features.*

2. *Again, we reconsider the example about an employee ontology in Chapter 1. We assume that "employees are uniquely determined by their branches' IDs and their personel IDs", which can be expressed by the following n-ary uniqueness constraint:*

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

Furthermore, it is assumed that "the extra pay for Microsoft's employees is determined by their positions and the amount of overtime they have worked". This is represented as follows:

$$(\mathsf{overtime\_hours}, \mathsf{position} \quad \textit{wkeyfor}\ \mathsf{Employee} \sqcap \exists \mathsf{work\_for}.\mathsf{Microsoft}, \mathsf{extra\_pay})$$

The concepts Employee *and* $\mathsf{Employee} \sqcap \exists \mathsf{work\_for}.\mathsf{Microsoft}$ *respectively refer to employees and those who work for Microsoft,* branch *is an abstract feature, and* id, position, overtime_hours *and* extra_pay *are concrete features.*

In the functional dependency, it is natural to use weak form, since we want to allow for the case that an arbitrary employee gives up on his extra pay.

3. *In an ontology about car dealers, we assume that cars and manufacturers are equipped with identification numbers and that every car is uniquely identified by the combination of its own identification number and its manufacturer's one. The following uniqueness constraint expresses this.*

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

Moreover, we assume that "the price of a used car is determined by its model and its mileage". That constraint is described by the following functional dependency:

$$(\mathsf{model},\ \mathsf{manufacturer\ id},\ \mathsf{mileage} \quad \textit{skeyfor}\ \mathsf{Car} \sqcap \exists \mathsf{formerOwner}.\mathsf{Human}, \mathsf{price})$$

In the above key constraints, $\mathsf{Car}, \mathsf{Human}$ *are concept names, the concept* $\mathsf{Car} \sqcap \exists \mathsf{formerOwner}.\mathsf{Human}$ *refers to used cars,* formerOwner *is a role name,* manufacturer *is an abstract feature, and* id, model, mileage, price *are concrete features.*

We now define the semantics of $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$, together with the most common decision problems.

Definition 3 ($\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$ 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 nominal* N *to a singleton subset* $N^{\mathcal{I}}$ *of* $\Delta_{\mathcal{I}}$*,*
- *each role name* R *to a subset* $R^{\mathcal{I}}$ *of* $\Delta_{\mathcal{I}} \times \Delta_{\mathcal{I}}$*,*
- *each abstract feature* f *to a partial function* $f^{\mathcal{I}}$ *from* $\Delta_{\mathcal{I}}$ *to* $\Delta_{\mathcal{I}}$*, and*
- *each concrete feature* g *to a partial function* $g^{\mathcal{I}}$ *from* $\Delta_{\mathcal{I}}$ *to* $\Delta_{\mathcal{D}}$*.*

If $u = f_1 \dots f_n g$ *is a path and* $d \in \Delta_{\mathcal{I}}$*, then* $u^{\mathcal{I}}(d)$ *is defined as* $g^{\mathcal{I}}(f_n^{\mathcal{I}}(\dots(f_1^{\mathcal{I}}(d))\dots))$*. The interpretation is extended to arbitrary concepts as*

follows:

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

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

$\mathcal{I}$ satisfies *a uniqueness constraint $(u_1, \ldots, u_k$ keyfor $C)$ if, for all $a, b \in C^{\mathcal{I}}$ the following holds: if, for $1 \leq i \leq k, u_i^{\mathcal{I}}(a) = u_i^{\mathcal{I}}(b)$, then $a = b$.*

$\mathcal{I}$ is a model *of a key box $\mathcal{K}$ iff $\mathcal{I}$ satisfies all functional dependencies and uniqueness constraints in $\mathcal{K}$. A concept C is* satisfiable w.r.t. a key box $\mathcal{K}$ *iff C and $\mathcal{K}$ has 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 $\mathcal{K}$.*

Like in any description logic that provides for negation and conjunction, in $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$ 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 allows us to concentrate only on concept satisfiability when investigating the comlexity of $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$.

In order not to commit decision procedures to any particular concrete domain, we need a well-defined interface between the decision procedure and the concrete domain reasoner. Usually, concrete domains are required to be admissible [3, 14], i.e., satisfiability of finite predicate conjunctions in concrete domains is decidable.

Definition 4 ($\mathcal{D}$-conjunction, admissibility)**.** *Let $\mathcal{D}$ be a concrete domain and V be 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 $x_j^{(i)}$ are variables from V. A $\mathcal{D}$-conjunction 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;*

3. *satisfiability of* $\mathcal{D}$*-conjunctions is decidable.*

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

As we will see in the subsequent section, reasoning w.r.t. key boxes introduced so far (i.e., general ones) in $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$ is undecidable. Fortunately, when restricting ourselves to a certain class of restricted key boxes, we preserve decidability. This class of key boxes is introduced in the following definition.

Definition 5 (Safe key box)**.** *A key box* $\mathcal{K}$ *is called* safe *if none of concepts apprearing in functional dependencies and uniqueness constraints in* $\mathcal{K}$ *has a subconcept of the form* $\exists u_1, \ldots, u_n.P$*.*

Note that requiring a key box to be safe is not a very severe restriction. As we can see, safe key boxes are expressive enough to model all the discussed examples.

2.2 Lower Complexity Bounds

In this section, we present lower complexity bounds for reasoning problems in the description logic $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$. First of all, it follows from undecidability of reasoning w.r.t. general key boxes in fragments of $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$ that satisfiability of $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$-concepts w.r.t. general key boxes is undecidable.

Before actually discussing complexity results, let us introduce some notations. A key box is called *unary* if it contains functional dependencies of the form (u depfor C, u') and uniqueness constraints of the form (u keyfor C) only. It is called *Boolean* if all concepts appearing in it are Boolean combinations of concept names. It is called *path-free* if it contains paths of length one only, and it is called *weak* if it contains weak functional dependencies only. Finally, a concept is called *path-free* if it contains paths of length one only.

Now, for $\mathcal{ALCOK}(\mathcal{D})$, it is proved in [19] that even when considering unary key boxes, concept satisfiability w.r.t. general unary key boxes in $\mathcal{ALCK}(\mathcal{D})$, a fragment of $\mathcal{ALCOK}(\mathcal{D})$, is undecidable.

Lemma 6 (Theorem 3.3 in [19])**.** *There exists a concrete domain* $\mathcal{D}$ *such that* $\mathcal{D}$*-satisfiability is in* PTime *and satisfiability of* $\mathcal{ALCK}(\mathcal{D})$*-concepts w.r.t. path-free unary key boxes is undecidable.*

Moreover, a similar result is established in [19] for a large class of concrete domains, which in fact is quite relevant in applications. These are arithmetic concrete domains.

Definition 7. *A concrete domain* $\mathcal{D}$ *is called* arithmetic *iff* $\mathbb{N} \subseteq \Delta_{\mathcal{D}}$ *and* $\Phi_{\mathcal{D}}$ *contains the following predicates:*

1. *unary predicates* $=_0$ *with* $(=_0)^{\mathcal{D}} = \{0\}$ *and* $\neq_0$ *with* $(\neq_0)^{\mathcal{D}} = \Delta_{\mathcal{D}} \backslash \{0\}$*,*

2. *binary equality and inequality predicates,*

3. *a ternary predicate* $+$ *with* $(+)^{\mathcal{D}} \cap \{(k_1, k_2, x) \mid k_1, k_2 \in \mathbb{N} \text{ and } x \in \Delta_{\mathcal{D}}\} = \{(k_1, k_2, k_1 + k_2) \mid k_1, k_2 \in \mathbb{N}\}$*, and*

4. *a ternary predicate* $*$ *with* $(*)^{\mathcal{D}} \cap \{(k_1, k_2, x) \mid k_1, k_2 \in \mathbb{N} \text{ and } x \in \Delta_{\mathcal{D}}\} = \{(k_1, k_2, k_1 * k_2) \mid k_1, k_2 \in \mathbb{N}\}$.

Lemma 8 (Theorem 3.4 in [19]). *For an arithmetic concrete domain* $\mathcal{D}$*, satisfiability of* $\mathcal{ALCK}(\mathcal{D})$*-concepts w.r.t. path-free unary key boxes is undecidable.*

For $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$, even similar results holds when we admit only unary path-free weak key boxes.

Lemma 9 (Corollary 1 in [21]). *There exists a concrete domain* $\mathcal{D}$ *such that* $\mathcal{D}$*-satisfiability is in* PTime *and* $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$*-concept satisfiability w.r.t. unary path-free weak key boxes is undecidable.*

Lemma 10 (Theorem 2 in [21]). *For an arithmetic concrete domain* $\mathcal{D}$*, satisfiability of* $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$*-concepts w.r.t. (general) weak key boxes is undecidable.*

Based on above lemmas and the fact that $\mathcal{ALCK}(\mathcal{D})$ and $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$ are fragments of $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$, we get unsatisfiability of reasoning w.r.t. general key boxes in $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$.

Theorem 11. *There exists a concrete domain* $\mathcal{D}$ *such that* $\mathcal{D}$*-satisfiability is in* PTime *and* $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$*-satisfiability w.r.t. general key boxes is undecidable.*

Theorem 12. *Let* $\mathcal{D}$ *be an arithmetic concrete domain. Then, satisfiability of* $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$*-concepts w.r.t. general key boxes is undecidable.*

Obviously, Theorems 11 and 12 establish a discouraging undecidability result about reasoning in $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$. Fortunately, as discussed in [19] and [21], restricting to certain classes of key boxes regains decidability of reasoning in $\mathcal{ALCOK}(\mathcal{D})$ and $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$. For $\mathcal{ALCOK}(\mathcal{D})$, the proposed class of key boxes is Boolean ones whereas for $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$ these are safe key boxes. Furthermore, [19] and [21] show the existence of a concrete domain $\mathcal{D}$ such that reasoning w.r.t. Boolean key boxes in $\mathcal{ALCOK}(\mathcal{D})$ and reasoning w.r.t. safe key boxes in $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$ become NExpTime-hard.

Lemma 13 (Theorem 3.7 in [19]). *There exists a concrete domain* $\mathcal{D}$ *such that* $\mathcal{D}$*-satisfiability is in* PTime *and satisfiability of path-free* $\mathcal{ALCK}(\mathcal{D})$*-concepts w.r.t. Boolean unary key boxes is* NExpTime*-hard.*

Lemma 14 (Corollary 2 in [21]). *There exists a concrete domain* $\mathcal{D}$ *such that* $\mathcal{D}$*-satisfiability is in* PTime *and* $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$*-concept satisfiability w.r.t. safe key boxes is* NExpTime*-hard.*

Furthermore, [19] and [21] also point out a large class of concrete domains such that the NExpTime lower complexity holds for reasoning w.r.t. Boolean key boxes in $\mathcal{ALCK}(\mathcal{D})$ and reasoning w.r.t. safe key boxes in $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$, respectively.

Lemma 15 (Theorem 3.8 in [19]). *Let* $\mathcal{D}$ *be a concrete domain. If there exists* $a, b \in \Delta_{\mathcal{D}}$ *with* $a \neq b$ *and* $P_1, P_2 \in \Phi_{\mathcal{D}}$ *such that* $P_1^{\mathcal{D}} = \{a\}$ *and* $P_2^{\mathcal{D}} = \{b\}$*, then satisfiability of path-free* $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$*-concepts w.r.t. unary Boolean key boxes is* NExpTime*-hard.*

Lemma 16 (Theorem 4 in [21]). *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.*

Note that safe key boxes are more general than Boolean key boxes, i.e., they additionally allow for sub-concepts of forms $\exists R.C, \forall R.C$, or $g\uparrow$ in key boxes. We will show that even when we allow for safe key boxes in the presence of uniqueness constraints, reasoning in $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$ remains decidable. Therefore, from now on we focus ourselves to reasoning w.r.t. safe key boxes only.

Since $\mathcal{ALCK}(\mathcal{D})$ and $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$ are fragments of $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$, the following theorem immediately follows from the above lemmas.

Theorem 17. *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 $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$-concepts w.r.t. safe key boxes is* NExpTime*-hard.*

It is easily seen that conditions on the concrete domain $\mathcal{D}$ in Lemmas 15 and 16 are equivalent in the sense that we can do an appropriate renaming to obtain one from the other. Therefore, Theorem 17 already covers the concrete domain considered in Lemma 15.

Since concept subsumption is reduced to concept satisfiability using negation, from Theorem 17 we get the co-NExpTime-hardness of concept subsumption in $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$.

Chapter 3

A Reasoning Procedure

In this chapter, we present a decision procedure for $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$. As discussed in Section 2.2, $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$-concept satisfiability w.r.t. general key boxes is undecidable. Fortunately, when restricting ourselves to safe key boxes, we regain decidability. In Section 3.1, the decision procedure is presented. Then we prove the termination, soundness, and completeness of the algorithm in Section 3.2. Based on the bounded model property for $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$ induced by the proofs, we show an alternative algorithm that yields a NExpTime upper complexity bound for $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$ with a restricted key box $\mathcal{D}$.

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

In this section, a tableau algorithm for $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$-concept satisfiability w.r.t. safe key boxes is presented. Like other tableau algorithms (see, e.g., [6]), the algorithm decides satisfiability of a concept by trying to construct a model for it. The algorithm starts with an initial structure induced by the input concept and the input key box, and applies so-called completion rules step by step until an obvious contradiction is found or no more rules is applicable. In the former case, the input concept is not satisfiable whereas in the latter case it is satisfiable.

To deal with concrete domains, the algorithm works on trees which have two types of nodes: abstract ones representing individuals of the logical domain, and concrete ones mapped to values of the concrete domain. In order not to commit the algorithm to any particular concrete domain, an interface between the tableau algorithm and a concrete domain reasoner is needed. Usually, admissibility of a concrete domain $\mathcal{D}$, which guarantees that $\mathcal{D}$-satisfiability is decidable (c.f. Definition 4), is enough.

However, due to functional dependencies and uniqueness constraints, besides satisfiability of a given $\mathcal{D}$-conjunction, additionally we need information about which concrete nodes have to be mapped to the same concrete domain value. Thus, a stronger condition for concrete domains from [19, 20], called *key-admissibility*, is needed for $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$. We define key-admissibility in a non-deterministic way since the algorithm for $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$ is non-deterministic.

Definition 18 (key-admissible)**.** *A concrete domain $\mathcal{D}$ is* key-admissible *iff it satisfies the following properties:*

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

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

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

Note that key-admissibility is not significantly stronger than admissibility since it is shown in [19] that every admissible concrete domain that provides for an equality predicate is also key-admissible. Therefore, throughout this paper, we assume that the concrete domain $\mathcal{D}$ provides for an equality predicate.

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

$$\begin{gathered}
\neg(C \sqcap D) \rightsquigarrow \neg C \sqcup \neg D \qquad \neg(C \sqcup D) \rightsquigarrow \neg C \sqcap \neg D \\
\neg(\exists R.C) \rightsquigarrow \forall R.\neg C \qquad \neg(\forall R.C) \rightsquigarrow \exists R.\neg C \\
\neg\neg C \rightsquigarrow C \\
\neg(\exists u_1, \ldots, u_n.P) \rightsquigarrow \exists u_1, \ldots, u_n.\bar{P} \sqcup u_1{\uparrow} \sqcup \ldots \sqcup u_n{\uparrow} \\
\neg(g{\uparrow}) \rightsquigarrow \exists g.\top_{\mathcal{D}}
\end{gathered}$$

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

We use $\dot{\neg} C$ to denote the result of converting $\neg C$ into NNF. A keybox $\mathcal{K}$ is in NNF if all concepts occurring in functional dependencies and uniqueness constraints 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 $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$-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 and uniqueness constraints in $\mathcal{K}$. For a set of concepts Γ, $\mathsf{sub}(\Gamma)$ denotes $\bigcup_{C \in \Gamma} \mathsf{sub}(C)$. Moreover, 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

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

Let us now introduce the underlying data structure for the tableau algorithm.

Definition 19 (Completion system)**.** *Let O_a and O_c be disjoint and countably infinite sets of* abstract *and* concrete nodes. *A* completion tree *for an $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$-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 $a \in V_a, x \in V_c$ is labelled with a concrete feature $\mathcal{L}(a, x)$ occurring in C or $\mathcal{K}$.*

For $a \in V_a$, 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 $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$-concept C and a key box $\mathcal{K}$ is a tuple $(T, \mathcal{P}, \prec, \sim)$, 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 $P \in \Phi_{\mathcal{D}}$ of arity n in C to a subset of V_c^n,*
- *$\prec$ is a strict, total ordering on V_a such that $a \prec b$ implies $\mathsf{lev}_T(a) \leq \mathsf{lev}_T(b)$, and*
- *$\sim$ is an equivalence relation on V_c.*

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

Intuitively, the relation $\sim$ records equalities between concrete nodes that are found by the tableau algorithm. The relation $\sim$ then induces an equivalence relation $\approx_a$ between abstract nodes, which results in another equivalence relation $\approx_c \supseteq \sim$ between concrete nodes.

Definition 20 ($\approx_a$ and $\approx_c$ Relations)**.** *Let $S = (T, \mathcal{P}, \prec, \sim)$ be a completion system for a concept C and a key box $\mathcal{K}$ with $T = (V_a, V_c, E, \mathcal{L})$, and $\approx$ an equivalence relation on V_a. For each $R \in \mathsf{N_R}$, a node $b \in V_a$ is an $R/\approx$-neighbor of a node $a \in V_a$ if there exists a node $c \in V_a$ such that $a \approx c$ and b is an R-successor of c. Similarly, for each $g \in \mathsf{N_{cF}}$ a node $x \in V_c$ is a $g/\approx$-neighbor of a if there exists a node $c \in V_a$ such that $a \approx c$ and x is a g-successor of c. For paths u, the notion $u/\approx$-neighbor is defined inductively as follows:*

- *if $u = g$, then x is a $u/\approx$-neighbor of a iff x is a $g/\approx$-neighbor of a.*
- *if $u = f_1 \dots f_n g$, then x is a $u/\approx$-neighbor of a iff there exists a node $b \in V_a$ such that b is an $f_1/\approx$-neighbor of a and x is an $f_2 \dots f_n g/\approx$-neighbor of b.*

We define a sequence of equivalence relations $\approx_a^0 \subseteq \approx_a^1 \subseteq \ldots$ *on* V_a *as follows:*

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

Finally, set $\approx_a = \bigcup_{i \geq 0} \approx_a^i$. *Then define*

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

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

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

with the initial completion tree

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

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 the rules are not applicable anymore. The completion rules add new nodes to completion trees by using an operation defined as follows.

Definition 21 ($\oplus$ operation)**.** *An abstract concrete node is called* fresh *w.r.t. a completion tree* T *if it does not appear in* T. *Let* $S = (T, \mathcal{P}, \prec, \sim)$ *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, $R \in \mathsf{N_R}$*) yields a completion system obtained from* S *in the following way:*
  - *if* $R \notin \mathsf{N_{aF}}$ *or* $R \in \mathsf{N_{aF}}$ *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* $R \in \mathsf{N_{aF}}$ *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.

  Moreover, b *is inserted into* $\prec$ *such that* $b \prec c$ *implies* $\mathsf{lev}_T(b) \leq \mathsf{lev}_T(c)$.
- $S \oplus agb$ *(*$a \in V_a, x \in O_c$ *fresh in* T, $g \in \mathsf{N_{cF}}$*) yields a completion system obtained from* S *in the following way:*

- *if a has no g-successors, then add x to V_a, (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 $u = f_1 \dots f_n g$ be a path. With $S \oplus aux$, where $a \in V_a$ and $x \in O_c$ fresh in T, we denote the completion system obtained from S by taking distinct nodes $b_1, \dots, b_n \in O_a$ which are fresh in T and setting

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

Note that since the tableau algorithm keeps track of equivalences between abstract and concrete nodes, when adding new nodes to completion trees, it does not really need to respect the functionality of abstract and concrete features as it does by the $\oplus$ operation. In fact, we can use the non-functional "+" operation from [19] instead. However, in order to be consistent with the tableau algorithm for $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$ in [21], and more importantly, to simplify the proof of the bounded model property, the tableau algorithm in this paper uses the $\oplus$ operation.

Let us now define what an obvious clash means.

Definition 22 (Clash)**.** *Let $S = (T, \mathcal{P}, \prec, \sim)$ be a completion system for a concept C_0 and a key box $\mathcal{K}$ with $T = (V_a, V_c, E, \mathcal{L})$. We say that the completion system S is* concrete domain satisfiable *iff the conjunction*

$$\zeta_S = \bigwedge_{P \text{ used in } C_0} \bigwedge_{(x_1,\dots,x_n) \in \mathcal{P}(P)} P(x_1, \dots, x_n) \wedge \bigwedge_{x \approx_c y} = (x, y)$$

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

1. *there is an $a \in V_a$ and an $A \in \mathsf{N_C}$ such that $\{A, \neg A\} \subseteq \mathcal{L}(a)$,*
2. *there are $a \in V_a$ and $x \in V_c$ such that $g\!\uparrow \in \mathcal{L}(a)$ and x is a g-successor of a,*
3. *S is not concrete domain satisfiable.*

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

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

Definition 23 ($\approx_u, \approx$ Relations, Blocking)**.** *Let $S = (T, \mathcal{P}, \prec, \sim)$ 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 $u/\approx_a$-neighbors (written $a \approx_u b$) if the following holds:*

- *if a has a $u/\approx_a$-neighbor x, then b has a $u/\approx_a$-neighbor y and $x \sim y$;*
- *if b has a $u/\approx_a$-neighbor x, then a has a $u/\approx_a$-neighbor y and $x \sim y$.*

Let $\mathsf{suff}(C_0, \mathcal{K})$ *be the set of all suffixes of paths that appear in a subconcept* $\exists u_1, \ldots, u_n.P \in \mathsf{sub}(C_0)$*, in a functional dependency, or in a uniqueness constraint (either on the left- and right-hand side) in the key box* $\mathcal{K}$*. Moreover, given a node* $a \in V_a$*, let*

$$\mathbb{L}(a) := \bigcup_{b \approx_a a} \mathcal{L}(b)$$

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

1. $\mathbb{L}(a) = \mathbb{L}(b)$*, 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.*

The blocking condition here is similar to that in [21] except that in this paper we consider labels of all equivalent abstract nodes according to the relation $\approx_a$ instead of labels of each node only, and $u/\approx_a$-neighbors instead of u-successors. The reason is that in our context, abstract nodes that are equivalent according to the relation $\approx_a$ describe the same node in the model. Thus, their node labels and their successors should be identical.

The first part of the blocking condition, which requires each pair of blocking and directly blocked nodes to have the same labels, is known from tableau algorithms for other DLs. The second part is introduced due to the presence of uniqueness constraints and functional dependencies. It guarantees that while blocking we consider not only abstract nodes and their labels, but also concrete ones and the concrete equivalence relation. The blocking mechanism is explained by an example after the completion rules and the algorithm are given. In the completion rules, if ρ is a binary relation on $V_c \times V_c$, we denote by ρ^* the reflexive, symmetric and transitive closure of ρ.

Completion rules:

R⊓ 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⊔ 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∃ if $\exists R.C \in \mathcal{L}(a)$, a is not blocked, and there is no R-successor b of a such that $C \in \mathcal{L}(b)$, then set $S := S \oplus aRb$ for a fresh $b \in \mathsf{O_a}$ and $\mathcal{L}(b) := \{C\} \cup \mathcal{L}(b)$[1].

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

[1] In the case when R is an abstract feature f and a has already had a f-successor c, then according to the definition of the $\oplus$ operation, c is renamed with b, and hence the rule must take into account concepts already in $\mathcal{L}(b)$. In other cases, we have $\mathcal{L}(b) = \emptyset$, and thus there is no problem with the assignment.

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

Rwkey if $C \in \mathcal{L}(a) \cap \mathcal{L}(b)$, $(u_1, \ldots, u_n$ wkeyfor $C, u) \in \mathcal{K}$, a and b are not blocked, a has a u_i-successor x_i, b has a u_i-successor y_i, $x_i \sim y_i$ 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 $C \in \mathcal{L}(a) \cap \mathcal{L}(b)$, $(u_1, \ldots, u_n$ skeyfor $C, u) \in \mathcal{K}$, a and b are not blocked, a has a u_i-successor x_i, b has a u_i-successor y_i, $x_i \sim y_i$ 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 $x \sim z$, then set $S := S \oplus buy$ with $y \in \mathsf{O_c}$ fresh and $\sim := (\sim \cup \{(x, y)\})^*$.

Rp if $\mathcal{L}(b) \not\subseteq \mathcal{L}(a)$, a is not blocked, and $a \in V_a$ is minimal w.r.t. $\prec$ such that $a \approx_a b$ then set $\mathcal{L}(a) := \mathcal{L}(a) \cup \mathcal{L}(b)$.

Rcp if $a \in V_a$ is not blocked, a is minimal w.r.t. $\prec$ in the set $\{b \in V_a \mid b \approx_a a\}$, a has a $u/\approx_a$-neighbor x for some $u \in \mathsf{suff}(C_0, \mathcal{K})$, and a does not have a u-successor, then set $S := S \oplus auy$ with $y \in \mathsf{O_c}$ fresh and $\sim := (\sim \cup \{(x, y)\})^*$.

Let us give some further remarks on the completion rules. The $R\sqcup$ and *Rch* are non-deterministic. The first five rules, namely $R\sqcap, R\sqcup, R\exists, R\forall$, and $R\exists c$, are known from the existing algorithm for $\mathcal{ALC}(\mathcal{D})$-concept satisfiability (see, e.g., [14]). The other five rules are introduced to deal with key boxes.

The *Rch* rule is a so-called "choice rule": intuitively, for an abstract node a and a functional dependency $(u_1, \ldots, u_k$ depfor $C, u)$ or a uniqueness constraint $(u_1, \ldots, u_k$ keyfor $C)$ in the key box $\mathcal{K}$ such that a has neighbors for all paths u_i, it guesses whether or not a satisfies C. This is necessary since both possibilities may have ramifications.

The *Rwkey* rule dealing with weak functional dependencies and *Rskey* dealing with strong functional dependencies are similar to those in [20]. We will demonstrate later that without blocking, the *Rskey* rule can cause infinite runs of the algorithm.

The *Rp* rule dealing with equalities between abstract nodes recorded by the $\approx_a$ relation is similar to that in [19]. Intuitively, if $a \approx_a b$ then a and b describe the same node in the model, and thus their node labels should be identical. However, it suffices to choose one representative among the equivalent nodes and make sure that this representative's node labels contains the labels of all equivalent ones. We choose the node that is minimal w.r.t. the ordering $\prec$ as the representative. The *Rp* does the appropriate copying of node labels.

The *Rcp* rule, which is new, also deals with equalities between abstract nodes recorded by the $\approx_a$ relation. Let us explain the necessity of the *Rcp* by the following example. Let a be an abstract node of the constructed tree such that a is blocked. Obviously, successors of a are also blocked and are not considered in the model. However, it may happen that a u-successor of a, for some $u \in \mathsf{suff}(C_0, \mathcal{K})$, is created by an application of the rule $R\exists c$ or *Rskey* to an ancestor b of a, and thus, we must "save" information about the u-successor in order to make b satisfy its labels in the model. A proper way to do it is to add a new u-successor to an arbitrary node c such that $c \approx_a a$ because actually

a and c refers to the same individual in the model. However, to simplify the copying, we choose the minimal node w.r.t. $\prec$ such that it is equivalent to a to add the u-successor to. While the Rp rule does the copying of node labels, the Rcp does the appropriate copying of paths.

Let us now present the tableau algorithm in pseudo-code notation. It is started with $\mathsf{sat}(S_{C_0})$. In the algorithm, check refers to the function computing a concrete equivalence for a given $\mathcal{D}$-conjunction.

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

Before proving termination, soundness, and completeness of the tableau algorithm, let us show an example explaining the need for the blocking mechanism.

Example 2. *Let us reconsider an example in [21], which shows that the Rskey rule endangers the termination of the algorithm. Consider satisfiability of*

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

Without blocking, applications of Rskey generate an infinite f-chain such that each element has a g-successor that is zero as shown in Figure 3.1. The second part of the blocking condition is introduced to deal with this effect.

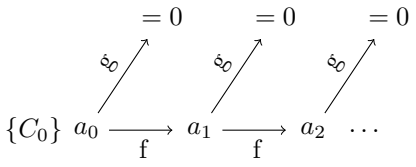

Figure 3.1: An infinite chain created by the tableau algorithm

Note that the tableau algorithm for $\mathcal{ALCOK}(\mathcal{D})$-concept satisfiability introduced in [19] does not need a blocking mechanism. The reason is that it works only with Boolean key boxes, i.e., those in which concepts are Boolean combinations of concept names. Since there is no strong functional dependency in key boxes, the problem discussed in Example 2 does not occur, and thus, blocking mechanism is not necessary.

3.2 Termination, soundness, and completeness

In this section, we prove termination, soundness, and completeness of the tableau algorithm given in Section 3.1. We first introduce a few notions.

Definition 24 (Path length, Role depth)**.** *Let $u = f_1 \dots f_n g$ be a path. The* length *of u is defined as $|u| = n + 1$.*

The role depth *of concepts is defined inductively as follows:*

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

Let C be a concept and $\mathcal{K}$ be a key box. By $\mathsf{feat}(C)$ and $\mathsf{feat}(\mathcal{K})$ we denote the set of abstract and concrete features appearing in C and the set of those appearing in functional dependencies and uniqueness constraints in $\mathcal{K}$, respectively. Besides, we use $|C|$ to denote the length of C, i.e., the number of symbols used to write it down, and $|\mathcal{K}|$ to denote

$$\begin{aligned}
&\sum_{(u_1,\dots,u_k \text{ keyfor } C)\in\mathcal{K}} (|u_1| + \dots + |u_n| + |C|)\\
+\ &\sum_{(u_1,\dots,u_k \text{ depfor } C,u)\in\mathcal{K}} (|u_1| + \dots + |u_n| + |C| + |u|)
\end{aligned}$$

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

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

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

1. $n = 0$. Then $C \equiv C_0$ and a is the root node, i.e., $\mathsf{lev}_T(a) = 0$.
2. $n \ge 1$. Assume $C \notin \mathsf{cl}(\mathcal{K})$. Since $C \in \mathsf{cl}(C_0, \mathcal{K}) = \mathsf{sub}(C_0) \cup \mathsf{cl}(\mathcal{K})$, we have that $C \in \mathsf{sub}(C_0)$. We distinguish three cases:
   - C is added to $\mathcal{L}(a)$ via an application of $R\sqcup$ or $R\sqcap$ rule. Then there exists an $E \in \mathcal{L}(a)$ such that $E = C \sqcup D$ or $E = C \sqcap D$ for some concept $D \in \mathsf{cl}(C_0, \mathcal{K})$. Since $C \notin \mathsf{cl}(\mathcal{K})$, we have that $E \notin \mathsf{cl}(\mathcal{K})$. Since E is added to $\mathcal{L}(a)$ by an earlier rule application, by induction hypothesis we obtain that $\mathsf{lev}_T(a) \le \mathsf{rd}(C_0) - \mathsf{rd}(E)$. Because $\mathsf{rd}(E) = \mathsf{rd}(C)$, we have that $\mathsf{lev}_T(a) \le \mathsf{rd}(C_0) - \mathsf{rd}(C)$.

- 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 $D \in \{\exists R.C, \forall R.C\}$. Since $C \notin \mathsf{cl}(\mathcal{K})$, we have that $D \notin \mathsf{cl}(\mathcal{K})$. Since D is added to $\mathcal{L}(b)$ by an earlier rule application, the induction hypothesis implies that $\mathsf{lev}_T(b) \leq \mathsf{rd}(C_0) - \mathsf{rd}(D)$. Using $\mathsf{rd}(D) = \mathsf{rd}(C)+1$ and $\mathsf{lev}_T(b) = \mathsf{lev}_T(a)-1$, we obtain that $\mathsf{lev}_T(a) \leq \mathsf{rd}(C_0) - \mathsf{rd}(C)$.
- C is added to $\mathcal{L}(a)$ via an application of Rp rule. Then there is a $b \in V_a$ with $C \in \mathcal{L}(b)$, and a is minimal w.r.t. $\prec$ such that $a \approx_a b$. Since $C \notin \mathsf{cl}(\mathcal{K})$ and C is added to $\mathcal{L}(b)$ by an earlier rule application, the induction hypothesis yields that $\mathsf{lev}_T(b) \leq \mathsf{rd}(C_0) - \mathsf{rd}(C)$. Moreover, since a is minimal w.r.t. $\prec$ such that $a \approx_a b$, we have that $\mathsf{lev}_T(a) \leq \mathsf{lev}_T(b)$. Therefore, $\mathsf{lev}_T(a) \leq \mathsf{rd}(C_0) - \mathsf{rd}(C)$.

□

Similarly to [21], we show in the following lemma and corollary that the completion tree is of bounded size.

Lemma 26. *Let $S = (T, \mathcal{P}, \prec, \sim)$ with $T = (V_a, V_c, E, \mathcal{L})$ be a completion system constructed during the run of the tableau algorithm started on an input concept C_0 and a safe key box $\mathcal{K}$. Then the following holds:*

(a) The out-degree of T is bounded by $|\mathsf{sub}(C_0) \cup \mathsf{sub}(\mathsf{con}(\mathcal{K}))| + |\mathsf{feat}(C_0) \cup \mathsf{feat}(\mathcal{K})|$;

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

$$|V_c/_\sim| \leq \#\{c \in V_c \mid \mathsf{lev}_T(c) \leq \mathsf{rd}(C_0)\} \leq N^{\mathsf{rd}(C_0)}, \text{ where}$$

$$N = |\mathsf{sub}(C_0) \cup \mathsf{sub}(\mathsf{con}(\mathcal{K}))| + |\mathsf{feat}(C_0) \cup \mathsf{feat}(\mathcal{K})|;$$

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

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

(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 applications of the rules $R\exists, R\exists c$, $Rskey$, and Rcp. The rule $R\exists$ generates at most one successor for each $\exists R.C \in \mathsf{sub}(C_0) \cup \mathsf{cl}(\mathcal{K})$. From the fact that $\mathsf{cl}(\mathcal{K}) = \mathsf{sub}(\mathsf{con}(\mathcal{K})) \cup \{\dot\neg D \mid D \in \mathsf{sub}(\mathsf{con}(\mathcal{K}))\}$ and the rewrite rule $\neg(\forall R.C) \rightsquigarrow \exists R.\neg C$, we have that

$$\begin{aligned} & |\{\exists R.C \mid \exists R.C \in \mathsf{sub}(C_0)\} \cup \{\exists R.C \mid \exists R.C \in \mathsf{cl}(\mathcal{K})\}| \\ \leq\ & |\mathsf{sub}(C_0) \\ & \quad \cup \{\exists R.C \mid \exists R.C \in \mathsf{sub}(\mathsf{con}(\mathcal{K}))\} \cup \{\forall R.C \mid \forall R.C \in \mathsf{sub}(\mathsf{con}(\mathcal{K}))\}| \\ \leq\ & |\mathsf{sub}(C_0) \cup \mathsf{sub}(\mathsf{con}(\mathcal{K}))|. \end{aligned}$$

Therefore, the number of successors of $a \in V_a$ created by applying the rule $R\exists$ is bounded by $|\mathsf{sub}(C_0) \cup \mathsf{sub}(\mathsf{con}(\mathcal{K}))|$.

The rule $R\exists c$ generates at most one successor for every abstract or concrete feature appearing in some $\exists u_1, \ldots, u_n.P \in \mathsf{sub}(C_0)$ (because $\mathcal{K}$ is safe,

there is no subconcept of the form $\exists u_1, \ldots, u_n.P$ in $\mathcal{K}$). Besides, the rule *Rskey* generates at most one successor for every abstract or concrete feature appearing in paths in $\mathcal{K}$, whereas the rule *Rcp* creates at most one successor for every abstract or concrete feature appearing in paths in C_0 and $\mathcal{K}$. Due to the definition of the operation $\oplus$, when dealing with abstract and concrete features, it does not create any successor if the successor already exists. Therefore, the number of successors of an abstract node a created by appling the rules $R\exists c$, *Rskey*, and *Rcp* is bounded by $|\mathsf{feat}(C_0) \cup \mathsf{feat}(\mathcal{K})|$.

To sum up, the number of successors of $a \in V_a$ is bounded by $|\mathsf{sub}(C_0) \cup \mathsf{sub}(\mathsf{con}(\mathcal{K}))| + |\mathsf{feat}(C_0) \cup \mathsf{feat}(\mathcal{K})|$.

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

$$(\forall c \in V_c)(\exists c' \in V_c)(\mathsf{lev}_T(c') \leq \mathsf{rd}(C_0) \wedge c \sim c') \tag{3.1}$$

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

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

Now, we prove (3.1) by strong induction on $\mathsf{rang}(c)$:

1. $\mathsf{rang}(c) = 0$. Then c is created due to an application of the $R\exists c$ rule to a node $a \in V_a$ and a concept $C \in \mathcal{L}(a), C = \exists u_1, \ldots, u_n.P$. Since $\mathcal{K}$ is safe, $C \notin \mathsf{cl}(\mathcal{K})$, and thus by Lemma 25, we have that $\mathsf{lev}_T(a) \leq \mathsf{rd}(C_0) - \mathsf{rd}(C)$. Due to the definition of $\mathsf{rd}(\exists u_1, \ldots, u_n.P)$, we obtain that $\mathsf{lev}_T(c) - \mathsf{lev}_T(a) \leq \mathsf{rd}(C)$. Combining these two inequalities yields that $\mathsf{lev}_T(c) \leq \mathsf{rd}(C_0)$. Since $c \sim c$, the base case is proved.
2. $\mathsf{rang}(c) = n, n \geq 1$. We distinguish two cases:
   - c is created due to an application of the *Rskey* rule to nodes $a, b \in V_a$ and a strong functional dependency $(u_1, \ldots, u_n \text{ skeyfor } C, u) \in \mathcal{K}$. The node a has a u-successor x created due to an application of $R\exists c$ or an earlier application of *Rskey* or *Rcp*. The node c is a u-successor of b such that $c \sim x$ 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.
   - c is created due to an application of the *Rcp* rule to node $a \in V_a$ and a path $u \in \mathsf{suff}(C_0, \mathcal{K})$. The node a has a $u/{\approx_a}$-neighbor x created due to an application of $R\exists c$ or an earlier application of *Rskey* or *Rcp*. Therefore, $\mathsf{rang}(x) < \mathsf{rang}(c) = n$, and thus by induction hypothesis, there exists a $c' \in V_c$ such that $x \sim c'$ and $\mathsf{lev}_T(c') \leq \mathsf{rd}(C_0)$. From the *Rcp* rule, we have that $c \sim x$. 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 $N = |\mathsf{sub}(C_0) \cup \mathsf{sub}(\mathsf{con}(\mathcal{K}))| + |\mathsf{feat}(C_0) \cup \mathsf{feat}(\mathcal{K})|$. Using the fact that concrete nodes are leaves of the tree T, we get

$$\#\{c \in V_c \mid \mathsf{lev}_T(c) \leq \mathsf{rd}(C_0)\} \leq N^{\mathsf{rd}(C_0)}$$

Finally, combining this result with (3.1), we complete the proof of (b).

(c) Using the definition of $\approx$ and the fact that $\sim$ and $\approx_a$ are equivalence relations, it is not difficult to show 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, obviously $\mathsf{suff}(C_0, \mathcal{K})$ is of bounded size. Let $m = |\mathsf{suff}(C_0, \mathcal{K})|$. Let us define a mapping $\phi : V_a \to 2^{\mathsf{suff}(C_0,\mathcal{K}) \times V_c/_\sim}$ in the following way:

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

Then, the following holds:

$$a \approx b \text{ iff } \phi(a) = \phi(b) \text{ and } \mathbb{L}(a) = \mathbb{L}(b) \tag{3.2}$$

Due to the definition of $\oplus$, the functionality of paths is respected and therefore, there are no $v_i, v_j \in V_c/_\sim$, for $i \neq j$, such that $\{(u, v_i), (u, v_j)\} \subseteq \phi(a)$ for some $a \in V_a$ and $u \in \mathsf{suff}(C_0, \mathcal{K})$. Thus, we have that

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

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 $\sum_{i=0}^{m} \binom{m}{i} \cdot n^i = (n+1)^m$, we get that $|\{\phi(a) \mid a \in V_a\}| \leq (n+1)^m = (n+1)^{|\mathsf{suff}(C_0,\mathcal{K})|}$. Finally, using (3.2) and the fact that $|\{\mathbb{L}(a) \mid a \in V_a\}| \leq 2^{|\mathsf{cl}(C_0,\mathcal{K})|}$, we obtain that

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

(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 an 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.

□

Corollary 27. *Let $S = (T, \mathcal{P}, \prec, \sim)$ with $T = (V_a, V_c, E, \mathcal{L})$ be a completion system constructed during the run of the tableau algorithm started on an 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) $|V_a/_\approx| \leq ((|C_0| + |\mathcal{K}|)^{|C_0|} + 1)^{|C_0|+|\mathcal{K}|} . 2^{2 \cdot (|C_0|+|\mathcal{K}|)}$;

(c) There exists a constant k such that $\#V_c + \#V_a$ is bounded by $2^{2^{(|C_0|+|\mathcal{K}|)^k}}$.

Proof. (a) By Lemma 26(a), the out-degree of T is bounded by $|\mathsf{sub}(C_0) \cup \mathsf{sub}(\mathsf{con}(\mathcal{K}))| + |\mathsf{feat}(C_0) \cup \mathsf{feat}(\mathcal{K})| \leq |\mathsf{sub}(C_0)| + |\mathsf{sub}(\mathsf{con}(\mathcal{K}))| + |\mathsf{feat}(C_0)| + |\mathsf{feat}(\mathcal{K})|$. Moreover, it is easily proved by structural induction that $|\mathsf{sub}(C_0)| + |\mathsf{feat}(C_0)| \leq |C_0|$ and $|\mathsf{sub}(\mathsf{con}(\mathcal{K}))| + |\mathsf{feat}(\mathcal{K})| \leq |\mathcal{K}|$. Therefore, the out-degree of T is bounded by $|C_0| + |\mathcal{K}|$.

(b) From Lemma 26(c) we have that $|V_a/{\approx}| \leq (n+1)^{|\mathsf{suff}(C_0,\mathcal{K})|}.2^{|\mathsf{cl}(C_0,\mathcal{K})|}$, where $n = |V_c/{\sim}|$.

By Lemma 26(b), $|V_c/{\sim}| \leq N^{\mathsf{rd}(C_0)}$ in which $N = |\mathsf{sub}(C_0) \cup \mathsf{sub}(\mathsf{con}(\mathcal{K}))| + |\mathsf{feat}(C_0) + \mathsf{feat}(\mathcal{K})|$. As shown in Part (a), $N \leq |C_0| + |\mathcal{K}|$. Moreover, obviously we have that $\mathsf{rd}(C_0) \leq |C_0|$. Therefore, $|V_c/{\sim}| \leq (|C_0| + |\mathcal{K}|)^{|C_0|}$.

It is easily seen that $|\mathsf{suff}(C_0, \mathcal{K})| \leq |C_0| + |\mathcal{K}|$. Furthermore, by definition of $\mathsf{cl}(C_0, \mathcal{K})$, we have that $|\mathsf{cl}(C_0, \mathcal{K})| \leq |\mathsf{sub}(C_0)| + |\mathsf{cl}(\mathcal{K})| \leq |\mathsf{sub}(C_0)| + 2 \cdot |\mathsf{sub}(\mathsf{con}(\mathcal{K}))| \leq |C_0| + 2 \cdot \sum_{C \in \mathsf{con}(\mathcal{K})} |C| \leq |C_0| + 2 \cdot |\mathcal{K}| \leq 2 \cdot (|C_0| + |\mathcal{K}|)$.

Summing it up, we obtain that

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

(c) According to Lemma 26(d), we have that the depth of T is bounded by $|V_a/{\approx}| + \mathsf{mpl}(C_0, \mathcal{K})$. From Part (b), we obtain that

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

Furthermore, it is easy to see that $\mathsf{mpl}(C_0, \mathcal{K}) \leq |C_0| + |\mathcal{K}|$. Therefore, there exists a constant m such that the depth of T is bounded by $2^{(|C_0|+|\mathcal{K}|)^m}$.

Since the out-degree of T is bounded by $|C_0| + |\mathcal{K}|$ by Part (a) and the depth of T is bounded by $2^{(|C_0|+|\mathcal{K}|)^m}$, we have that the number of nodes in T is bounded by

$$\frac{(|C_0| + |\mathcal{K}|)^{2^{(|C_0|+|\mathcal{K}|)^m}-1} - 1}{|C_0| + |\mathcal{K}| - 1}.$$

Therefore, there is a constant k such that the number of nodes in T is bounded by $2^{2^{(|C_0|+|\mathcal{K}|)^k}}$. The fact that $V_a \cup V_c$ is the set of nodes in T, together with that V_a and V_c are disjoint, finishes the proof.

□

Lemma 28. *There is a constant k such that, if the tableau algorithm is started with $C_0, \mathcal{K}$, then, in every recursion step, the while loop terminates after at most $2^{2^{(|C_0|+|\mathcal{K}|)^k}}$ steps.*

Proof. Let $S = (T, \mathcal{P}, \prec, \sim)$ with $T = (V_a, V_c, E, \mathcal{L})$ be the argument of the sat function, $\sim_1, \sim_2, \ldots$ be the sequence of concrete equivalences computed in the while loop, and $\approx_c^1, \approx_c^2, \ldots$ be the corresponding $\approx_c$ relations.

It is not difficult to see that $\sim_1 \subsetneq \sim_2 \subsetneq \ldots$: If the while loop reaches the i-th step, we have that $\sim_{i-1} \neq \approx_c^{i-1}$ after step $i-1$. By definition, $\sim_{i-1} \subseteq \approx_c^{i-1}$, and thus $\sim_{i-1} \subsetneq \approx_c^{i-1}$. By definition of ζ_S, we have that $\approx_c^{i-1} \subseteq \sim_i$ for $i \geq 0$. Therefore, $\sim_{i-1} \subsetneq \sim_i$.

By Corollary 27(c), there exists a constant k such that $\#V_c \leq \#V_a + \#V_c \leq 2^{2^{(|C_0|+|\mathcal{K}|)^k}}$. Therefore, $\#{\sim} \leq 2^{2\cdot 2^{(|C_0|+|\mathcal{K}|)^k}}$ which, together with the above claim, implies that the number of steps performed by the while loop is also bounded by $2^{2\cdot 2^{(|C_0|+|\mathcal{K}|)^k}}$. □

Lemma 29. *There is a constant k such that, if the tableau algorithm is started with an $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$-concept C_0 and a safe key box $\mathcal{K}$, then the number of recursion calls is bounded by $2^{2^{(|C_0|+|\mathcal{K}|)^k}}$.*

Proof. It suffices to establish an appropriate upper bound on the number of rule applications.

- The $R\sqcap, R\sqcup, R\exists$, and $R\exists c$ rules can be applied at most once for each concept in a node label;
- the rules $R\forall$ and Rp can be applied at most once for every concept $C \in \mathsf{cl}(C_0, \mathcal{K})$ and every pair of (abstract) nodes;
- the rule Rch can be applied at most once for every abstract node and every functional dependency or uniqueness constraint in $\mathcal{K}$;
- the rules $Rskey$ and $Rwkey$ can be applied at most once for every functional dependency in $\mathcal{K}$ and every pair of abstract nodes;
- the rule Rcp can be applied at most once for every path $u \in \mathsf{suff}(C_0, \mathcal{K})$ and every abstract node.

Furthermore, Corollary 27(c) shows that the number of nodes in T is at most double exponential in $|C_0| + |\mathcal{K}|$. Since neither nodes nor concepts in node labels are ever deleted, the fact that node labels are subsets of $\mathsf{cl}(C_0, \mathcal{K})$ and the number of paths in $\mathsf{suff}(C_0, \mathcal{K})$ is bounded by $|C_0| + \mathcal{K}|$ thus implies that the number of applications of these rules is at most double exponential in $|C_0| + |\mathcal{K}|$. □

Termination of the algorithm follows immediately from Lemmas 28 and 29.

Lemma 30 (Termination)**.** *When started with an $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$-concept C_0 and a safe key box $\mathcal{K}$, both in NNF, the tableau algorithm terminates.*

It is easily seen that the tableau algorithm presented in the previous section is in 2NExpTime. The reason is that the algorithm is non-deterministic and it constructs in a monotonic way a completion tree whose number of nodes in the worst case is double-exponential in $|C_0| + |\mathcal{K}|$.

Let us now show that the algorithm is sound, i.e., the input concept C_0 is satisfiable w.r.t. the key box $\mathcal{K}$ if the algorithm returns satisfiable. It is done by extracting a model $\mathcal{I}$ from the constructed completion tree for C_0 and $\mathcal{K}$. The point is that not all nodes of the completion tree but only "representatives" of equivalence classes are put into the domain. Doing this way, we obtain a model whose size is only exponential in $|C_0| + |\mathcal{K}|$. This property helps us to show a NExpTime upper complexity bound in Section 3.3.

Lemma 31 (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*

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

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

$$\delta(x) = \delta(y) \text{ iff } x \approx_c y. \tag{3.3}$$

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

$$\bigwedge_{P \text{ used in } C_0} \bigwedge_{(x_1,\ldots,x_n)\in\mathcal{P}(P)} P(x_1,\ldots,x_n)$$

of ζ_S. Moreover, for each conjunct $= (x, y)$ from the second part of ζ_S, we have $x \approx_c y$ by definition of ζ_S and thus $\delta(x) = \delta(y)$ by (3.3).

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

$$\begin{aligned}
\Delta_\mathcal{I} =& \{a \in V_a \mid a \text{ is not blocked and there is no unblocked } b \in V_a \text{ such that} \\
& \qquad a \approx b \wedge b \prec a\} \cup \{w\} \\
A^\mathcal{I} =& \{a \in \Delta_\mathcal{I} \mid A \in \mathcal{L}(a)\} \\
N^\mathcal{I} =& \begin{cases} \{a \in \Delta_\mathcal{I} \mid N \in \mathcal{L}(a) & \text{if there is an } a \in \Delta_\mathcal{I} \text{ such that } N \in \mathcal{L}(a)\} \\ \{w\} & \text{otherwise} \end{cases} \\
R^\mathcal{I} =& \{(a, b) \in \Delta_\mathcal{I} \times \Delta_\mathcal{I} \mid \text{there is a } b' \in V_a \text{ such that } b' \text{ is an } R\text{-successor of } a, \\
& \qquad \text{and } b \approx b'\} \\
g^\mathcal{I} =& \{(a, \delta(x)) \in \Delta_\mathcal{I} \times \Delta_D \mid x \text{ is a } g\text{-successor of } a\}
\end{aligned}$$

for all $A \in \mathsf{N_C}, N \in \mathsf{N_O}, R \in \mathsf{N_R}$, and $g \in \mathsf{N_{cF}}$.

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

Claim 1. $\mathcal{I}$ is well-defined.

Proof.

- $N^\mathcal{I}$ is a singleton for each $N \in \mathsf{N_O}$. Assume that there exist $a, b \in \Delta_\mathcal{I}$ such that $a \neq b$ and $N \in \mathcal{L}(a) \cap \mathcal{L}(b)$. By definition of $\approx_a$, $N \in \mathcal{L}(a) \cap \mathcal{L}(b)$ implies that $a \approx_a b$. It is easily seen that $\approx_a \subseteq \approx$, and thus $a \approx b$. From $a, b \in \Delta_\mathcal{I}$, we obtain that a, b are not blocked. Since $a \in \Delta_\mathcal{I}, a \approx b$, and b is unblocked, the construction of $\mathcal{I}$ yields that $b \not\prec a$. Similarly, we have that $a \not\prec b$. Since $\prec$ is strict and total, $a \not\prec b$, $b \not\prec a$, and $a \neq b$ yield a contradiction.
- for an abstract feature $f \in \mathsf{N_{aF}}$ and a concrete feature $g \in \mathsf{N_{cF}}$, thanks to the $\oplus$ operation, we always have that $f^\mathcal{I}$ and $g^\mathcal{I}$ are functional.

Claim 2. If $a \in V_a$ is unblocked or directly blocked, then there exists exactly one $b \in \Delta_\mathcal{I}$ such that $a \approx b$. Furthermore, it holds that

(a) $\mathcal{L}(a) \subseteq \mathcal{L}(b)$, and

(b) if a has a $u/\approx_a$-neighbor x, for some $u \in \mathsf{suff}(C_0, \mathcal{K})$, then b has a u-successor y such that $x \sim y$.

Proof. For each $a \in V_a$, let $U_a := \{c \in V_a \mid c \approx a\}$ and b be the minimal node in U_a w.r.t. $\prec$. Obviously, $a \in U_a$, and thus U_a is non-empty, which guarantees the existence of b. It is easily seen that b is not blocked, and thus due to the construction of $\mathcal{I}$ we have that $b \in \Delta_\mathcal{I}$. Moreover, since $\prec$ is total and strict, for all $a' \in U_a$ it holds that if $a' \neq b$ then $b \prec a'$. Therefore, due to the construction of $\mathcal{I}$, we obtain that for all $a' \in U_a$, if $a' \neq b$ then $a' \notin \Delta_\mathcal{I}$. In other words, there is only one $b \in U_a$ such that $b \in \Delta_\mathcal{I}$.

From the definition of $\approx$, we have that $\approx_a \subseteq \approx$. The equivalence relation $\approx_a$ partitions U_a into equivalence classes $U_a^1, \ldots, U_a^n$. Let $a_1, \ldots, a_n$ be minimal nodes w.r.t. $\prec$ in $U_a^1, \ldots, U_a^n$ respectively. Since b is minimal in U_a w.r.t. $\prec$, we have that $b \in \{a_1, \ldots, a_n\}$. Moreover, because the relation $\approx$ is transitive, $a_i \approx a_j$ for all $1 \leq i, j \leq n$.

(a) Due to non-applicability of the *Rp* rule, for each $1 \leq i \leq n$ we have $\mathcal{L}(a_i) = \cup_{c \in U_a^i} \mathcal{L}(c)$, and thus $\mathcal{L}(a_i) = \mathbb{L}(c)$ for all $c \in U_a^i$.

Let c_1, c_2 be two arbitrary nodes in U_a, and $1 \leq i, j \leq n$ such that $c_1 \in U_a^i, c_2 \in U_a^j$. Because $c_1, c_2 \in U_a$ and the relation $\approx$ is transitive, we have $c_1 \approx c_2$. Therefore, $\mathbb{L}(c_1) = \mathbb{L}(c_2)$, and thus $\mathcal{L}(a_i) = \mathcal{L}(a_j)$. Because c_1 and c_2 are arbitrary nodes, it holds that $\mathcal{L}(a_i) = \mathcal{L}(a_j)$ for all $1 \leq i, j \leq n$.

Because $b \in \{a_1, \ldots, a_n\}$, it holds that $\mathcal{L}(b) = \mathcal{L}(a_i)$ for all $1 \leq i \leq n$. Since $a \in U_a$, there is i_0, $1 \leq i_0 \leq n$, such that $a \in U_a^{i_0}$. Therefore, $\mathcal{L}(a) \subseteq \mathbb{L}(a) = \mathcal{L}(a_{i_0}) = \mathcal{L}(b)$.

(b) Due to non-applicability of the *Rcp* rule, for each $1 \leq i \leq n$ and $c \in U_a^i$ we have that if c has a $u/{\approx_a}$-neighbor x, for some $u \in \mathsf{suff}(C_0, \mathcal{K})$ then a_i has a u-successor y such that $x \sim y$.

Suppose that $a \in U_a^{i_0}$, for some $1 \leq i_0 \leq n$, and a has a $u/{\approx_a}$-neighbor x, for some $u \in \mathsf{suff}(C_0, \mathcal{K})$. Then for each $1 \leq i \leq n$, we have that $a \approx a_i$, and thus by definition of the relation $\approx$ we obtain that a_i has a $u/{\approx_a}$-neighbor z such that $x \sim z$. Hence, due to the above argument, a_i has a u-successor z' such that $z \sim z'$, and thus $x \sim z'$ since $\sim$ is transitive.

Because $b \in \{a_1, \ldots, a_n\}$, the above argument also holds for b. Therefore, b has a u-successor y such that $x \sim y$.

Claim 3. 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 the claim by induction on the length of u.

(a)
- $|u| = 1$: true by the construction of $\mathcal{I}$.
- $|u| = n+1, n \geq 1$: Let $u = f_1 \ldots f_n g$. Since $u^\mathcal{I}$ is defined, by definition of $\mathcal{I}$ there exists a $b \in \Delta_\mathcal{I}$ such that $f_1^\mathcal{I}(a) = b$ and $(f_2 \ldots f_n g)^\mathcal{I}(b)$ is defined. By induction hypothesis, b has an $f_2 \ldots f_n g$-successor y such that $(f_2 \ldots f_n g)^\mathcal{I}(b) = \delta(y)$. Furthermore, since $f_1^\mathcal{I}(a) = b$, there exists a $b' \in V_a$ such that b' is an f_1-successor of a and $b' \approx b$, and thus $b \approx_u b'$ for all $u \in \mathsf{suff}(C_0, \mathcal{K})$. Hence, b' has an $f_2 \ldots f_n g/{\approx_a}$-neighbor z such that $z \sim y$. Since b' is an f_1-successor of a, we have that z is an $f_1 \ldots f_n g/{\approx_a}$-neighbor of a. Because $a \in \Delta_\mathcal{I}$, Claim 2(b) yields that a has an $f_1 \ldots f_n g$-successor x such that $x \sim z$. Since $\sim$ is an equivalence relation, we have that $x \sim y$. Therefore, x is a u-successor of a such that $u^\mathcal{I}(a) = \delta(y) = \delta(x)$.

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

- $|u| = n + 1, n \geq 1$: Let $u = f_1 \dots f_n g$ and $b \in V_a$ an f_1-successor of a. Since x is an $f_1 \dots f_n g$-successor of a, we have that x is an $f_2 \dots f_n g$-successor of b. Since $a \in \Delta_\mathcal{I}$, we have that b is either unblocked or directly blocked. Due to Claim 2, there exists a $b' \in \Delta_\mathcal{I}$ such that $b \approx b'$. Thus, definition of $\mathcal{I}$ yields that $f_1^\mathcal{I}(a) = b'$. Since $b \approx b'$, we have that $b \approx_u b'$ for all $u \in \mathsf{suff}(C_0, \mathcal{K})$. Therefore, b' has an $f_2 \dots f_n g/\approx_a$-neighbor y such that $x \sim y$. Moreover, since $b' \in \Delta_\mathcal{I}$, Claim 2(b) yields that b' has an $f_2 \dots f_n g$-successor z such that $y \sim z$. Because $\sim$ is an equivalence relation, we have that $x \sim z$. Since $b' \in \Delta_\mathcal{I}$ and $f_2 \dots f_n g \in \mathsf{suff}(C_0, \mathcal{K})$, by induction hypothesis we obtain that $(f_2 \dots f_n g)^\mathcal{I}(b') = \delta(z)$. Therefore, we have that $u^\mathcal{I}(a) = \delta(z) = \delta(x)$.

Claim 4. 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. True by the construction of $\mathcal{I}$.
- $C = \neg D$. Since C is in NNF, D is a concept name. Clash-freeness of S implies that $D \notin \mathcal{L}(a)$. The construction of $\mathcal{I}$ implies that $a \notin D^\mathcal{I}$, which yields that $a \in (\neg D)^\mathcal{I}$.
- $C = D \sqcap E$. The completeness of S implies that $\{D, E\} \subseteq \mathcal{L}(a)$. By induction hypothesis, we have that $a \in D^\mathcal{I}$ and $a \in E^\mathcal{I}$. Therefore, $a \in (D \sqcap E)^\mathcal{I}$.
- $C = D \sqcup E$. The completeness of S implies that $\{D, E\} \cap \mathcal{L}(a) \neq \emptyset$. By induction hypothesis, we have that $a \in D^\mathcal{I}$ or $a \in E^\mathcal{I}$. Therefore, $a \in (D \sqcup E)^\mathcal{I}$.
- $C = \exists R.D$. Non-applicability of the $R\exists$ rule implies that a has an R-successor b such that $D \in \mathcal{L}(b)$. Then either b is unblocked or directly blocked. Due to Claim 2, there is a $b' \in \Delta_\mathcal{I}$ such that $b \approx b'$. By the construction of $\mathcal{I}$, we have that $(a, b') \in R^\mathcal{I}$. Furthermore, Claim 2(a) yields that $\mathcal{L}(b) \subseteq \mathcal{L}(b')$, and thus $D \in \mathcal{L}(b')$. By induction hypothesis, we have that $b' \in D^\mathcal{I}$, and thus $a \in C^\mathcal{I}$.
- $C = \forall R.D$. Let $(a, b) \in R^\mathcal{I}$. By the construction of $\mathcal{I}$, there is an R-successor b' of a such that $b' \approx b$. Since the $R\forall$ rule is not applicable, we have that $D \in \mathcal{L}(b')$. Due to Claim 2(a), we obtain that $\mathcal{L}(b') \subseteq \mathcal{L}(b)$, and thus $D \in \mathcal{L}(b)$. By induction hypothesis, we have that $b \in D^\mathcal{I}$. Because it holds for all b such that $(a, b) \in R^\mathcal{I}$, we have that $a \in C^\mathcal{I}$.
- $C = \exists u_1, \dots, u_n.P$. Since the $R\exists_c$ rule is not applicable, there exist $x_1, \dots, x_n \in V_c$ such that x_i is a u_i-successor of a for $1 \leq i \leq n$ and $(x_1, \dots, x_n) \in \mathcal{P}(P)$. By Claim 3(b), we have that $u_i^\mathcal{I}(a) = \delta(x_i)$ for $1 \leq i \leq n$. Since $(x_1, \dots, x_n) \in \mathcal{P}(P)$ and δ is a solution for ζ_S, we obtain that $(\delta(x_1), \dots, \delta(x_n)) \in P^\mathcal{D}$, and thus $a \in C^\mathcal{I}$.
- $C = g\!\uparrow$. Since S is clash-free, there is 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 $C_0 \in \mathcal{L}(a_0)$ and $a_0 \in \Delta_\mathcal{I}$, Claim 4 implies that $\mathcal{I}$ is a model of C_0. The satisfaction of all functional dependencies and uniqueness constraints in $\mathcal{K}$ by $\mathcal{I}$ is shown as follows:

- $\mathcal{I}$ satisfies all $(u_1, \ldots, u_n$ skeyfor $C, u) \in \mathcal{K}$: let $a, b \in C^\mathcal{I}$ 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. Since the *Rch* rule is not applicable, we have that $\{C, \dot{\neg}C\} \cap \mathcal{L}(a) \neq \emptyset$. If $\dot{\neg}C \in \mathcal{L}(a)$, then Claim 4 implies that $a \in (\dot{\neg}C)^\mathcal{I}$, which contradicts $a \in C^\mathcal{I}$. Therefore, $C \in \mathcal{L}(a)$ and for the same reason $C \in \mathcal{L}(b)$. Using Claim 3(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 has 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, the fact that $\delta(x_i) = \delta(y_i)$ implies that $x_i \sim y_i$, for $1 \leq i \leq n$. Similarly, we have 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 $x \sim y$, and thus $\delta(x) = \delta(y)$. Claim 3(b) yields that $u^\mathcal{I}(b) = \delta(y)$, and since $\delta(x) = \delta(y)$ we obtain that $u^\mathcal{I}(a) = u^\mathcal{I}(b)$.

- $\mathcal{I}$ satisfies all $(u_1, \ldots, u_n$ wkeyfor $C, u) \in \mathcal{K}$: let $a, b \in C^\mathcal{I}$ 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. Similarly to the previous case, we obtain that $C \in \mathcal{L}(a) \cap \mathcal{L}(b)$. Using Claim 3(a) and the definition of δ, we have that a has a u_i-successor x_i and b has 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* implies that $x \sim y$. Using Claim 3(b), we have that $u^\mathcal{I}(a) = \delta(x)$ and $u^\mathcal{I}(b) = \delta(y)$. Since $\delta(x) = \delta(y)$, we obtain that $u^\mathcal{I}(a) = u^\mathcal{I}(b)$.

- $\mathcal{I}$ satisfies all $(u_1, \ldots, u_n$ keyfor $C) \in \mathcal{K}$: let $a, b \in C^\mathcal{I}$ such that $u_i^\mathcal{I}(a) = u_i^\mathcal{I}(b)$ for $1 \leq i \leq n$. Similarly to previous cases, we obtain that $C \in \mathcal{L}(a) \cap \mathcal{L}(b)$. Using Claim 3(a) and the definition of δ, we have that a has a u_i-successor x_i and b has a u_i-successor y_i such that $x_i \sim y_i$ for $1 \leq i \leq n$. Therefore, we have that $a \approx_a b$, and thus $a \approx b$. Because $a, b \in \Delta_\mathcal{I}$ and $a \approx b$, we have that $a = b$.

Now, for the size of $\Delta_\mathcal{I}$, it follows from the construction of $\mathcal{I}$ that $|\Delta_\mathcal{I}| \leq |V_a/{\approx}|$. By Corollary 27, we have that

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

□

Finally, we show completeness of the algorithm.

Lemma 32 (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 $S = (T, \mathcal{P}, \prec, \sim)$ with $T = (V_a, V_e, E, \mathcal{L})$ is called *$\mathcal{I}$-compatible* if there exist mappings $\pi : V_a \to \Delta_\mathcal{I}$ and $\tau : V_c \to \Delta_\mathcal{D}$ such that:

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

(C2) b is an 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) $(x_1, \ldots, x_n) \in \mathcal{P}(P) \Rightarrow (\tau(x_1), \ldots, \tau(x_n)) \in P^{\mathcal{D}}$

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

Let us first prove the following technical claims.

Claim 1. If completion system S is $\mathcal{I}$-compatible, then

(i) $a \approx_a b$ implies $\pi(a) = \pi(b)$,

(ii) $x \approx_c y$ implies $\tau(x) = \tau(y)$.

Proof. We show by induction on i that $a \approx_a^i b$ implies that $\pi(a) = \pi(b)$, which yields (i).

- If $a \approx_a^0 b$, then there exists a nominal N such that $N \in \mathcal{L}(a) \cap \mathcal{L}(b)$. By (C1) we have that $\pi(a) \in N^{\mathcal{I}}$ and $\pi(b) \in N^{\mathcal{I}}$. Definition of the semantics yields that $\pi(a) = \pi(b)$.
- Assuming that $a \approx_a^i b$ for $i \geq 1$, we distinguish three cases:
  1. If $a \approx_a^{i-1} b$, then $\pi(a) = \pi(b)$ by induction hypothesis.
  2. There exist a $c \in V_a$ and an $f \in \mathsf{N_{aF}}$ such that both a and b are $f/\approx_a^{i-1}$-neighbors of c. Hence, there exist $c_1, c_2 \in V_a$ such that $c \approx_a^{i-1} c_1 \approx_a^{i-1} c_2$, a is an f-successor of c_1, and b is an f-successor of c_2. By induction hypothesis, we have that $\pi(c) = \pi(c_1) = \pi(c_2)$. Thus, (C2) yields that $\{(\pi(c), \pi(a)), (\pi(c), \pi(b))\} \subseteq f^{\mathcal{I}}$, which implies that $\pi(a) = \pi(b)$ by the definition of the semantics.
  3. There exist $(u_1, \ldots, u_n$ keyfor $C) \in \mathcal{K}$, a $u_i/\approx_a^{i-1}$-neighbors x_i of a, and a $u_i/\approx_a^{i-1}$-neighbors y_i of b, for $1 \leq i \leq n$, such that $C \in \mathcal{L}(a) \cap \mathcal{L}(b)$ and $x_i \sim y_i$ for $1 \leq i \leq n$. (C1) yields that $\{\pi(a), \pi(b)\} \subseteq C^{\mathcal{I}}$. Using induction hypothesis, (C2), and (C3), it is easy to show that $u_i^{\mathcal{I}}(\pi(a)) = \tau(x_i)$ and $u_i^{\mathcal{I}}(\pi(b)) = \tau(y_i)$ for $1 \leq i \leq n$. By (C5), this implies that $u_i^{\mathcal{I}}(\pi(a)) = u_i^{\mathcal{I}}(\pi(b))$ for $1 \leq i \leq n$. Since $\mathcal{I}$ is a model of the key box $\mathcal{K}$, this yields that $\pi(a) = \pi(b)$ by definition of the semantics.

For Part (ii). If $x \approx_c y$, then either $x \sim y$ or there is an $a \in V_a$ and a $g \in \mathsf{N_{cF}}$ such that both x and y are $g/\approx_a$-neighbors of a. In the former case, (C5) that yields $\tau(x) = \tau(y)$. In the latter case, Part (i) and (C3) yields that $\{(\pi(a), \tau(x)), (\pi(a), \tau(y))\} \subseteq g^{\mathcal{I}}$, which implies that $\tau(x) = \tau(y)$.

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

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

Proof. Let S be an $\mathcal{I}$-compatible completion system, π and τ mappings satisfying conditions (C1)-(C5), and R a completion rule appliable to S. We make a case analysis according to the type of R:

R⊓ 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 that $\pi(a) \in (C_1 \sqcap C_2)^{\mathcal{I}}$, 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⊔ 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 that $\pi(a) \in (C_1 \sqcup C_2)^{\mathcal{I}}$, 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 such that the obtained completion system is $\mathcal{I}$-compatible via the same π and τ.

R∃ 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 $d \in \Delta_{\mathcal{I}}$ such that $(\pi(a), d) \in R^{\mathcal{I}}$ and $d \in C^{\mathcal{I}}$. Since the rule uses the operation $\oplus$ to add a new R-successor b of a and sets $\mathcal{L}(b) = \{C\} \cup \mathcal{L}(b)$, the obtained completion system is $\mathcal{I}$-compatible via $\pi' = \pi \cup \{b \mapsto d\}$ and τ.

R∀ The rule is applied to $\forall R.C \in \mathcal{L}(a)$ and adds C to the label $\mathcal{L}(b)$ of an existing R-successor b of a. By (C1), we have that $\pi(a) \in (\forall R.C)^{\mathcal{I}}$. Since b is an R-successor of a, (C2) yields that $(\pi(a), \pi(b)) \in R^{\mathcal{I}}$. By definition of the semantics, we have that $\pi(b) \in C^{\mathcal{I}}$, and thus the obtained completion system is $\mathcal{I}$-compatible via π and τ.

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

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

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

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

The obtained 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 $(u_1, \ldots, u_n \text{ depfor } C, u) \in \mathcal{K}$, where depfor $\in$ {skeyfor, wkeyfor}, or a uniqueness constraint $(u_1, \ldots, u_n \text{ keyfor } C) \in \mathcal{K}$, and non-deterministically adds C or $\dot{\neg} C$ to $\mathcal{L}(a)$. By definition of semantics, $\pi(a) \in C^{\mathcal{I}}$ or $\pi(a) \in (\dot{\neg} C)^{\mathcal{I}}$. Thus, *Rch* can be applied such 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 $(u_1, \ldots, u_n$ wkeyfor $C, u) \in \mathcal{K}$. Then $C \in \mathcal{L}(a) \cap \mathcal{L}(b)$, a has a u_i-successor x_i, b has a u_i-successor y_i such that $x_i \sim y_i$ for $1 \leq i \leq n$, a has a u-successor x, and b has a u-successor y. Due to (C1), $\{\pi(a), \pi(b)\} \subseteq C^{\mathcal{I}}$. By (C2), (C3), and an easy induction on the length of u_i and u, we have that $u_i^{\mathcal{I}}(\pi(a)) = \tau(x_i), u_i^{\mathcal{I}}(\pi(b)) = \tau(y_i)$ for $1 \leq i \leq n$, and $u^{\mathcal{I}}(\pi(a)) = \tau(x)$, $u^{\mathcal{I}}(\pi(b)) = \tau(y)$. Due to (C5), $\tau(x_i) = \tau(y_i)$ and thus $u_i^{\mathcal{I}}(\pi(a)) = u_i^{\mathcal{I}}(\pi(b))$ for $1 \leq i \leq n$. Since $\mathcal{I}$ is a model of $\mathcal{K}$, we have that $u^{\mathcal{I}}(\pi(a)) = u^{\mathcal{I}}(\pi(b))$, which implies that $\tau(x) = \tau(y)$. The rule sets $\sim := (\sim \cup (x, y))^*$. Obviously, (C5) is satisfied and the obtained 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 $(u_1, \ldots, u_n$ skeyfor $C, u) \in \mathcal{K}$ with $u = f_1 \ldots f_n g$. Then $C \in \mathcal{L}(a) \cap \mathcal{L}(b)$, a has a u_i-successor x_i, b has a u_i-successor y_i such that $x_i \sim y_i$ for $1 \leq i \leq n$, and a has a u-successor x. Due to (C1), $\{\pi(a), \pi(b)\} \subseteq C^{\mathcal{I}}$. By (C2), (C3), and an easy induction on the length of u_i and u, we have that $u_i^{\mathcal{I}}(\pi(a)) = \tau(x_i), u_i^{\mathcal{I}}(\pi(b)) = \tau(y_i)$ for $1 \leq i \leq n$, and $u^{\mathcal{I}}(\pi(a)) = \tau(x)$. Due to (C5), $\tau(x_i) = \tau(y_i)$ and thus $u_i^{\mathcal{I}}(\pi(a)) = u_i^{\mathcal{I}}(\pi(b))$ for $1 \leq i \leq n$. Since $\mathcal{I}$ is a model of $\mathcal{K}$, $u^{\mathcal{I}}(\pi(b))$ is defined and $u^{\mathcal{I}}(\pi(a)) = u^{\mathcal{I}}(\pi(b))$. Hence, there exist $d_1, \ldots, d_n \in \Delta_{\mathcal{I}}$ and $\alpha \in \Delta_{\mathcal{D}}$ 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 $b_1, \ldots, b_n$ and a new concrete node y such that: b_1 is an f_1-successor of b, b_i is an f_i-successor of b_{i-1} for $1 < i \leq n$ and y is a 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 obtained completion system is $\mathcal{I}$-compatible via π' and τ'.

Rp The rule is applied to a concept $C \in \mathcal{L}(a)$ and adds C to the label $\mathcal{L}(b)$ of a node b with $a \approx_a b$. By (C1), we have that $\pi(a) \in C^{\mathcal{I}}$. Claim 1(i) implies that $\pi(a) = \pi(b)$, and thus $\pi(b) \in C^{\mathcal{I}}$. Therefore, the obtained completion system is $\mathcal{I}$-compatible via π and τ.

Rcp The rule is applied to a node a and a path $f_1 \ldots f_n g \in \mathsf{suff}(C_0, \mathcal{K})$. Since a has an $f_1 \ldots f_n g/\approx_a$-neighbor, there exist abstract nodes $a', b_1, \ldots b_n$, $b'_1, \ldots, b'_n$ and a concrete node x such that

- $a \approx_a a'$ and b_1 is an f_1-successor of a',
- $b'_i \approx_a b_i$ and b_{i+1} is an f_{i+1}-successor of b'_i, for $1 \leq i < n$, and
- $b'_n \approx_a b_n$ and x is a g-successor of b'_n.

The rule application generates new abstract nodes c_i, for $1 \leq i \leq n$, and an concrete node y such that

- c_1 is an f_1-successor of a,
- c_i is an f_i-successor of c_{i-1}, for $1 < i \leq n$, and
- y is a g-successor of c_n.

By definitions of $\approx_a$ and $\approx_c$, it is easy to see that $b_i \approx_a b'_i \approx_a c_i$ for all $1 \leq i \leq n$ and $x \approx_c y$. Therefore, Claim 1(i) yields that $\pi(a) =$

$\pi(a'), \pi(b_i) = \pi(b_i') = \pi(c_i)$ for all $1 \leq i \leq n$, whereas Claim 1(ii) yields that $\tau(x) = \tau(y)$.

Since S is $\mathcal{I}$-compatible, (C2) and (C3) yield that $(\pi(a'), \pi(b_1)) \in f_1^{\mathcal{I}}$, $(\pi(b_i'), \pi(b_{i+1})) \in f_{i+1}^{\mathcal{I}}$ for $1 \leq i < n$, and $g^{\mathcal{I}}(\pi(b_n)) = \tau(x)$. Therefore, we have that $(\pi(a), \pi(c_1)) \in f_1^{\mathcal{I}}$, $(\pi(c_i), \pi(c_{i+1})) \in f_{i+1}^{\mathcal{I}}$ for $1 \leq i < n$, and $g^{\mathcal{I}}(\pi(c_n)) = \tau(y)$. Thus, the obtained completion system is $\mathcal{I}$-compatible via π and τ.

Furthermore, we show in the following Claim that $\mathcal{I}$-compatability implies clash-freeness.

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

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

- Assume that there is an $a \in V_a$ and a concept name A 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 a g-successor of a and $g\uparrow \in \mathcal{L}(a)$. Then (C1) yields that $\pi(a) \in (g\uparrow)^{\mathcal{I}}$. Moreover, since x is a g-successor of a, (C3) implies that $g^{\mathcal{I}}(\pi(a)) = \tau(x)$, which contradicts $\pi(a) \in (g\uparrow)^{\mathcal{I}}$.
- According to the properties (C4) and (C5), and Claim 1(ii), τ is a solution for ζ_S. Thus S is concrete domain satisfiable.

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

$$S_{C_0} = (T_{C_0}, \mathcal{P}_0, \emptyset, \emptyset) \text{ with } T_{C_0} = (\{a_0\}, \emptyset, \emptyset, \{a_0 \mapsto C_0\})$$

We guide the non-deterministic check function such that, when given a predicate conjunction ζ_S with 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 $x, y \in V$. The relation $\sim$ is a concrete equivalence since τ is a solution for ζ_S. With this guidance (C5) is obviously satisfied after each call to check, and the other properties are not affected by such a call. According to Claim 2, we can apply the completion rules in such a way that the $\mathcal{I}$-compatibility is preserved. By Lemma 30, the algorithm always terminates, and thus by Claim 3, no clash will be found and the algorithm returns satisfiable. □

The following theorem follows immediately from Lemmas 30, 31, and 32.

Theorem 33. *If $\mathcal{D}$ is a key-admissible concrete domain, the tableau algorithm decides satisfiability of $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$-concepts w.r.t. safe key boxes.*

3.3 Upper Complexity Bound

As discussed in the previous section, the tableau algorithm runs in 2NExpTime. However, we show in this section that the upper complexity bound of reasoning w.r.t. safe key boxes in $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$ matches the NExpTime lower complexity bound established in Theorem 17.

The idea is based on the bounded model property for $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$ yielded by Lemmas 31 and 32: if an $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$-concept C_0 is satisfiable w.r.t. a safe key box $\mathcal{K}$, then Lemma 32 implies that the tableau algorithm returns satisfiable, whereas Lemma 31 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

$$\mathsf{b}(C_0, \mathcal{K}) = ((|C_0| + |\mathcal{K}|)^{|C_0|} + 1)^{|C_0|+|\mathcal{K}|}.2^{2\cdot(|C_0|+|\mathcal{K}|)}. \tag{3.4}$$

It is easily seen that there is an $m \in \mathbb{N}$ such that $\mathsf{b}(C_0, \mathcal{K}) \leq 2^{(|C_0|+|\mathcal{K}|)^m}$. Therefore, every $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$-concept C_0 that is satisfiable w.r.t. $\mathcal{K}$ has a model $\mathcal{I}$ of size $|\Delta_\mathcal{I}| \leq 2^{(|C_0|+|\mathcal{K}|)^m}$. Following this observation, we show that $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$-concept satisfiability w.r.t. safe key boxes is in NExpTime, which coincides with the NExpTime lower complexity bound established in Theorem 17.

Theorem 34. *For a key-admissible concrete domain $\mathcal{D}$ such that extended $\mathcal{D}$-satisfiability is in* NP*, $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$-concept satisfiability w.r.t. safe key boxes is in* NExpTime*.*

Proof. Let us consider an alternative algorithm for deciding satisfiability of a given $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$-concept C_0 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 (3.4) and $V = \{v_1, \ldots, v_n\}$ 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 functions 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:

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

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 and uniqueness constraints in $\mathcal{K}$ 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 $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$-concept C_0 and a safe key box $\mathcal{K}$ iff C_0 is satisfiable w.r.t. $\mathcal{K}$. This follows immediately from the definition of quasi-models, the bounded-model property for $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$ w.r.t. safe key boxes obtained in Lemma 31, 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 $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$-concept satisfiability w.r.t. 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 if 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 $\neg A$ 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 $\mathcal{O}(|\Delta_{\mathcal{M}}|)$, 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 $(u_1, \ldots, u_n \text{ depfor } C, u)$, every pair $a, b \in \Delta_{\mathcal{M}}$ such that $C \in \mathcal{L}(a) \cap \mathcal{L}(b)$ and $(u_i^{\mathcal{M}}(a), u_i^{\mathcal{M}}(b)) \in \mathcal{P}^{\mathcal{M}}(=)$ $(i = 1 \ldots n)$, have "correct" u-successors, and for every uniqueness constraint $(u_1, \ldots, u_n \text{ keyfor } C)$, for every pair $a, b \in \Delta_{\mathcal{M}}$ such that $C \in \mathcal{L}(a) \cap \mathcal{L}(b)$ and $(u_i^{\mathcal{M}}(a), u_i^{\mathcal{M}}(b)) \in \mathcal{P}^{\mathcal{M}}(=)$ $(i = 1 \ldots n)$, it holds that $a = b$. 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 and uniqueness constraints 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. Alternatively, if a concrete domain $\mathcal{D}$ does not provide for equality and inequality predicates, 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 $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$-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 $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$-concept satisfiability w.r.t. safe key boxes can be decided in NExpTime if extended $\mathcal{D}$-satisfiability is in NP. □

Since in $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$ concept subsumption can be polynomially reduced to concept satisfiability, from Theorem 34 we have that concept subsumption in $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$ can be decided in NExpTime.

Chapter 4

Extending $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$

In this chapter, several extensions of $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$ are discussed. First of all, in Section 4.1 we extend $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$ by taking into account so-called *acyclic TBoxes*. We show that admitting acyclic TBoxes does not make reasoning in $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$ harder. Generalizing acyclic TBoxes, we consider *general TBoxes* in Section 4.2. It turns out that for a large class of concrete domains, reasoning w.r.t. general TBoxes in $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$ becomes undecidable. Finally, we discuss an extension of $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$ with inverse roles in Section 4.3 and show a similar undecidability result.

4.1 $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$ with Acyclic TBoxes

Besides a concept language, most description logics provide means for expressing terminological knowledge and background knowledge about the application domain. It is done by introducing a TBox formalism. In this section, we investigate the complexity of reasoning in $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$ with the presence of a restricted form of TBoxes, called acyclic TBoxes. First of all, let us define acyclic TBoxes.

Definition 35 (Acyclic TBox)**.** *An expression of the form $A \doteq C$, where A is a concept name and C is a concept, is called a* concept definition*.*

Let $\mathcal{T}$ be a finite set of concept definitions. A concept name A directly uses *a concept name B in $\mathcal{T}$ if there is a concept definition $A \doteq C \in \mathcal{T}$ such that B appears in C. By* "uses"*, we denote the transitive closure of "directly uses". $\mathcal{T}$ is called an* acyclic TBox *if*

(i) there is no concept A such that A uses itself, and

(ii) the left-hand sides of all concept definitions in $\mathcal{T}$ are pairwise distinct.

The size $|\mathcal{T}|$ *of the TBox $\mathcal{T}$ is defined as*

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

An interpretation $\mathcal{I}$ is a model *of an acyclic TBox $\mathcal{T}$ iff $A^{\mathcal{I}} = C^{\mathcal{I}}$ for all $A \doteq C \in \mathcal{T}$. A concept C is* satisfiable w.r.t. a TBox $\mathcal{T}$ and a key box $\mathcal{K}$ *iff*

there exists a model of $C, \mathcal{T}$, and $\mathcal{K}$. A concept D subsumes *a concept C* w.r.t. a TBox $\mathcal{T}$ and a key box $\mathcal{K}$, *written $C \sqsubseteq_{\mathcal{T},\mathcal{K}} D$, iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for all models $\mathcal{I}$ of $\mathcal{T}$ and $\mathcal{K}$.*

The following is an example of an acyclic TBox:

$$\begin{aligned}
\mathsf{Male} &\doteq \neg\mathsf{Female}\\
\mathsf{Woman} &\doteq \mathsf{Human} \sqcap \mathsf{Female}\\
\mathsf{Man} &\doteq \mathsf{Human} \sqcap \mathsf{Male}\\
\mathsf{Mother} &\doteq \mathsf{Woman} \sqcap \exists\mathsf{has_child}.\mathsf{Human}\\
\mathsf{Father} &\doteq \mathsf{Male} \sqcap \exists\mathsf{has_child}.\mathsf{Human}
\end{aligned}$$

Obviously, acyclic TBoxes *define* concepts, i.e., each concept definition in acyclic TBoxes assigns a concept name to a complex concept. Therefore, they can be used to expand concepts such that satisfiability of concepts w.r.t. acyclic TBoxes is equivalent to satisfiability of concepts without reference to TBoxes. Before discussing the expansion, let us introduce some notations.

Definition 36. *Given an acyclic TBox $\mathcal{T}$, a concept name is called*

- defined in $\mathcal{T}$ *if it occurs on the left-hand side of a concept definition in $\mathcal{T}$;*
- primitive in $\mathcal{T}$ *if it does not occur on the left-hand side of any concept definition in $\mathcal{T}$.*

A concept C is called unfolded *w.r.t. $\mathcal{T}$ iff no concept name occurring in C is defined in $\mathcal{T}$.*

Given an acyclic TBox $\mathcal{T}$, a concept C, using the following *unfolding* algorithm, every concept C can be transformed into a concept C', which is unfolded w.r.t. $\mathcal{T}$.

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

Moreover, given a key box $\mathcal{K}$, we use $\mathsf{unfold}(\mathcal{K}, \mathcal{T})$ as abbreviation for the following key box:

$$\begin{aligned}
&\Big\{(u_1, \ldots, u_n \text{ keyfor } \mathsf{unfold}(C, \mathcal{T})) \mid (u_1, \ldots, u_n \text{ keyfor } C) \in \mathcal{K}\Big\}\\
\cup &\Big\{(u_1, \ldots, u_n \text{ depfor } \mathsf{unfold}(C, \mathcal{T}), u) \mid (u_1, \ldots, u_n \text{ depfor } C, u) \in \mathcal{K}\Big\}
\end{aligned}$$

where depfor $\in$ {skeyfor, wkeyfor}. It is easy to prove the following relation between reasoning w.r.t. acyclic TBoxes and reasoning without reference to them.

Lemma 37. *Given a concept C, an acyclic TBox $\mathcal{T}$, and a key box $\mathcal{K}$, we have that C is satisfiable w.r.t. $\mathcal{T}$ and $\mathcal{K}$ iff* $\mathsf{unfold}(C, \mathcal{T})$ *is satisfiable w.r.t.* $\mathsf{unfold}(\mathcal{K}, \mathcal{T})$.

Hence, using Lemma 37, $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$-concept satisfiability w.r.t. acyclic TBoxes and key boxes can be decided by first applying the "unfolding" to the input concept and key box, and then applying the tableau algorithm presented in Chapter 3 to the obtained concept and key box. However, as discussed in [22], unfolding concepts may lead to an exponential blowup in their size. To illustrate the exponential blowup, let us consider the following TBox $\mathcal{T}$:

$$\begin{aligned} &C_0 \doteq A \\ &C_1 \doteq \exists r_1.C_0 \sqcap \exists r_2.C_0 \\ &\dots \\ &C_n \doteq \exists r_1.C_{n-1} \sqcap \exists r_2.C_{n-1} \end{aligned}$$

It is easily seen that the size of the concept $\mathsf{unfold}(C_n, \mathcal{T})$ is exponential in n (we assume that unary coding is not being used). Therefore, although unfolding is an appropriate means for obtaining deciability results, it is not adequate to obtaining tight complexity bounds.

In [14], it is shown that adding acyclic TBoxes does not make reasoning in $\mathcal{ALC}$ harder, i.e., reasoning w.r.t. acyclic TBoxes in $\mathcal{ALC}$ is PSPACE-complete. The idea is that instead of using unfolding as a preprocessing step, unfolding is done *on demand*, i.e., if the tableau algorithm encounters a concept name A in the label of some node a, and in the TBox there is a concept definition $A \doteq C$, then it adds C to the label of a and does no further unfold at this stage. Although the technique does not work with every description logics, e.g., reasoning in $\mathcal{ALCF}$ is PSPACE-complete whereas reasoning w.r.t. acyclic TBoxes in $\mathcal{ALCF}$ is NEXPTIME-complete (see, e.g., [14, 15]), we show in the rest of the section that it can be used to prove that admitting acyclic TBoxes does not make reasoning in $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$ harder.

First, the input TBox is converted into a normal form defined as follows.

Definition 38 (Simple TBox)**.** *A TBox $\mathcal{T}$ is called* simple *iff it is acyclic and satisfies the following conditions:*

- *the right-hand side of each concept definition in $\mathcal{T}$ is of the form $\neg A, A_1 \sqcap A_2, A_1 \sqcup A_2, \exists R.A_1, \forall R.A_1, \exists u_1, \dots, u_n.P$, or $g\!\uparrow$ where $A, A_1, A_2 \in \mathsf{N_C}$, $g \in \mathsf{N_{cF}}$, $u_1, \dots, u_n$ are paths, and P is a concrete domain predicate;*
- *if the right hand side of a concept definition in $\mathcal{T}$ is $\neg A$, then A does not occur on the left hand side of any concept definition in $\mathcal{T}$.*

In fact, admitting only simple TBoxes does not reduce expressivity of acyclic TBoxes, since each acyclic TBox can be converted to an equivalent simple one, as shown in the following lemma.

Lemma 39. *Any acyclic TBox $\mathcal{T}$ can be converted into a simple one $\mathcal{T}'$ in polynomial time such that $\mathcal{T}'$ is equivalent to $\mathcal{T}$ in the following sense: any model of $\mathcal{T}'$ can be extended to a model of $\mathcal{T}$ and vice versa.*

Proof. It is proved similarly to Lemma 4.3 in [14]. □

Because concept names appearing in key boxes can be defined in TBoxes, we need to revise the definition of safe key boxes.

Definition 40. *A key box $\mathcal{K}$ is called* safe w.r.t. an acyclic TBox $\mathcal{T}$ *if the key box* $\mathsf{unfold}(\mathcal{K}, \mathcal{T})$ *is safe.*

A pair $\langle \mathcal{T}, \mathcal{K} \rangle$ is called simple *iff*

- *$\mathcal{T}$ is simple, $\mathcal{K}$ is safe w.r.t. $\mathcal{T}$,*
- *$\mathsf{con}(\mathcal{K})$ contains only concept names, and*
- *for each concept name B in $\mathsf{con}(\mathcal{K})$, there exists a concept definition $B' \doteq D$ in $\mathcal{T}$ such that for every model $\mathcal{I}$ of $\mathcal{T}$, it holds that $(B')^{\mathcal{I}} = \Delta_{\mathcal{I}} \backslash B^{\mathcal{I}}$. From now on, by $\bar{B}_{\mathcal{T}}$ we refer to the concept name B' which satisfies the above condition.*

We now modify the tableau algorithm presented in Chapter 3 to decide satisfiability of $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$-concept names w.r.t. simple pairs of TBoxes and key boxes. It is noticed that the algorithm also decides satisfiability of complex concepts w.r.t. acyclic TBoxes and "complex" safe key boxes by first applying the following preprocessing step: let the input be a concept C, an acyclic TBox $\mathcal{T}$, and a key box $\mathcal{K}$ safe w.r.t. $\mathcal{T}$.

1. introduce a new concept name A that does not appear in $\mathcal{T}$ previously and add a definition $A \doteq C$ to $\mathcal{T}$;
2. for each concept name $B \in \mathsf{con}(\mathcal{K})$, we introduce a new concept name B' that does not appear in $\mathcal{T}$ previously and add a definition $B' = \neg B$ to $\mathcal{T}$;
3. for each concept $D \in \mathsf{con}(\mathcal{K})$ that is not a concept name, we introduce two new concept names B, B' that do not appear in $\mathcal{T}$ previously, then add two definitions $B \doteq D$ and $B' \doteq \dot{\neg} D$ to $\mathcal{T}$. Let $\mathcal{K}'$ be the key box obtained from $\mathcal{K}$ by replacing each concept D by the concept name B;
4. convert the resulting TBox into an equivalent simple one $\mathcal{T}'$.

It is not hard to see that

- C is satisfiable w.r.t. $\mathcal{T}$ and $\mathcal{K}$ iff A is satisfiable w.r.t. $\mathcal{T}'$ and $\mathcal{K}'$;
- the pair $\langle \mathcal{T}', \mathcal{K}' \rangle$ is simple; and
- $|\mathcal{T}'| + |\mathcal{K}'|$ is obviously polynomial to $|C| + |\mathcal{T}| + |\mathcal{K}|$.

Moreover, given a concept name A, a TBox $\mathcal{T}$, and a key box $\mathcal{K}$, it is easily seen that if A is primitive in $\mathcal{T}$, then A is always satisfiable w.r.t. $\mathcal{T}$ and $\mathcal{K}$. Therefore, in what follows we assume that A is defined in $\mathcal{T}$.

The modified algorithm uses completion systems of a restricted form as the underlying data structure: every node label is a set of concept names. We call such completion systems *simple*.

Definition 41 (Modified Completion Algorithm)**.** *The* modified $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$-completion algorithm *is obtained from the tableau algorithm in Chapter 3 by the following modifications:*

1. *Let the input be a concept name A and a simple pair $\langle \mathcal{T}, \mathcal{K} \rangle$. The algorithm starts with the initial completion system $S_A = (T_A, \mathcal{P}_0, \emptyset, \emptyset)$ with the initial completion tree $T_A = (a_0, \emptyset, \emptyset, \{a_0 \mapsto \{A\}\})$ where $\mathcal{P}_0$ maps each predicate P to $\emptyset$.*

2. *The rules* $R\sqcap, R\sqcup, R\exists, R\forall$, *and* $R\exists c$ *are modified as follows: in the premise of each completion rule, substitute*

$$"C \in \mathcal{L}(a)" \quad \textit{with} \quad "B \in \mathcal{L}(a) \textit{ and } B \doteq C \in \mathcal{T}".$$

For example, in the $R\sqcap$ *rule, "*$C_1 \sqcap C_2 \in \mathcal{L}(a)$*" is replaced with "*$B \in \mathcal{L}(a)$ *and* $(B \doteq C_1 \sqcap C_2) \in \mathcal{T}$*".*

3. *The modified Rch rule: if* $(u_1, \ldots, u_n$ *keyfor* $B)$ *or* $(u_1, \ldots, u_n$ *depfor* $B, u)$*, where depfor* $\in \{$*skeyfor, wkeyfor*$\}$*, in* $\mathcal{K}$ *and there exist* $x_1, \ldots, x_n \in V_c$ *such that* x_i *is a* u_i*-successor of* a *(not blocked) for* $1 \leq i \leq n$ *and* $\{B, \bar{B}_{\mathcal{T}}\} \cap \mathcal{L}(a) = \emptyset$*, then set* $\mathcal{L}(a) := \mathcal{L}(a) \cup \{D\}$ *for some* $D \in \{B, \bar{B}_{\mathcal{T}}\}$*.*

4. *The completion rules Rwkey, Rskey, Rp, and Rcp are not modified;*

5. *The definition of a clash is modified as follows. Let* $S = (T, \mathcal{P}, \prec, \sim)$ *be a simple completion system with* $T = (V_a, V_c, E, \mathcal{L})$*.* S *is said to contain a clash iff the*

   - *there is an* $a \in V_a$ *and two concept names* $B, B' \in \mathcal{L}(a)$ *such that there is a concept definition* $B' \doteq \neg B$ *in the TBox* $\mathcal{T}$[1]*;*
   - *there are* $a \in V_a$ *and* $x \in V_c$*, and a concept name* $B \in \mathcal{L}(a)$ *such that there is a concept definition* $B \doteq g\!\uparrow$ *in* $\mathcal{T}$ *and* x *is a g-successor of* a*; or*
   - S *is not concrete domain satisfiable.*

We now show soundness, completeness, and termination of the modified completion algorithm based on a correspondence between runs of the modified algorithm on the concept name A, the simple pair $\langle \mathcal{T}, \mathcal{K} \rangle$ and runs of the original algorithm on input $\mathsf{unfold}(A, \mathcal{T}), \mathsf{unfold}(\mathcal{K}, \mathcal{T})$. Similarly to [14], the correspondence is defined based on the notion "variant" of a completion system defined in as follows.

Definition 42. *A simple completion system* $S = (T, \mathcal{P}, \prec, \sim)$*, in which* $T = (V_a, V_c, E, \mathcal{L})$*, is a* variant *of a completion system* $S' = (T', \mathcal{P}', \prec', \sim')$*, in which* $T' = (V'_a, V'_c, E', \mathcal{L}')$*, w.r.t. a TBox* $\mathcal{T}$ *iff the following holds:*

1. $V_a = V'_a, V_c = V'_c, \sim = \sim', \prec = \prec'$*, and* $\mathcal{P} = \mathcal{P}'$*,*
2. $A \in \mathcal{L}(a)$ *and* $\mathsf{unfold}(A, \mathcal{T}) = C$ *implies that* $C \in \mathcal{L}'(a)$*,*
3. $C \in \mathcal{L}'(a)$ *implies that there exists an* $A \in \mathsf{N_C}$ *such that* $A \in \mathcal{L}(a)$ *and* $\mathsf{unfold}(A, \mathcal{T}) = C$*,*
4. $R \in \mathcal{L}(a, b)$ *iff* $R \in \mathcal{L}'(a, b)$*, and*
5. x *is a g-successor of* a *in* T *iff* x *is a g-successor of* a *in* T'*.*

Now, the correspondence between runs of the modified algorithm and runs of the original one is established in the following lemma.

[1] From the definition of simple TBoxes and the fact that the TBox $\mathcal{T}$ is simple, the concept name B is primitive.

Lemma 43. *Let S_1 be a simple completion system that is a variant of a completion system S_1' w.r.t. a simple TBox $\mathcal{T}$.*

- *If the modified completion algorithm can apply a completion rule R to S_1 yielding a completion system S_2, then the original tableau algorithm can apply R to S_1' yielding a completion system S_2' such that S_2 is a variant of S_2' w.r.t. $\mathcal{T}$.*
- *Conversely, if the original algorithm can apply a completion rule R to S_1' yielding a completion system S_2', then the modified completion algorithm can apply R to S_1 yielding a variant S_2 of S_2' w.r.t. $\mathcal{T}$.*

Proof. The lemma is proved by case analysis similarly to Lemma 5.55 in [14]. □

Similarly to Lemma 26 and Corollary 27, we establish an upper bound for the size of the tree constructed by the modified algorithm. Before showing the upper bound, let us introduce a few notions. Let $\mathsf{CN}(\mathcal{T})$ be the set of concept names appearing in $\mathcal{T}$, $\mathsf{DC}(\mathcal{T})$ the set of defined concept names in $\mathcal{T}$, and $\mathsf{PC}(\mathcal{T}) = \mathsf{CN}(\mathcal{T}) \backslash \mathsf{DC}(\mathcal{T})$, i.e., the set of primitive concept names occurring in $\mathcal{T}$. Besides, we use $\mathsf{feat}(\mathcal{T})$ to denote the set of abstract and concrete features appearing in $\mathcal{T}$, $\mathsf{suff}(\mathcal{T}, \mathcal{K})$ to denote the set of all suffixes of paths appearing in $\mathcal{T}$ or $\mathcal{K}$, and $\mathsf{mpl}(\mathcal{T}, \mathcal{K})$ to denote $\max\{|u| \mid u \in \mathsf{suff}(\mathcal{T}, \mathcal{K})\}$.

Lemma 44. *Let A be a concept name and $\langle \mathcal{T}, \mathcal{K} \rangle$ a simple pair of a TBox $\mathcal{T}$ and a key box $\mathcal{K}$. Let $C = \mathsf{unfold}(A, \mathcal{T})$, $\mathcal{K}' = \mathsf{unfold}(\mathcal{K}, \mathcal{T})$. Then, the following holds:*

(a) $\mathsf{rd}(C) \leq |\mathcal{T}|$,

(b) $|\mathsf{sub}(C) \cup \mathsf{sub}(\mathsf{con}(\mathcal{K}'))| \leq |\mathsf{CN}(\mathcal{T})| + |\mathsf{con}(\mathcal{K})|$,

(c) $|\mathsf{cl}(C, \mathcal{K}')| \leq 2 \cdot (|\mathcal{T}| + |\mathcal{K}|)$,

(d) $\mathsf{feat}(C) \subseteq \mathsf{feat}(\mathcal{T})$ *and* $\mathsf{feat}(\mathcal{K}') \subseteq \mathsf{feat}(\mathcal{T}) \cup \mathsf{feat}(\mathcal{K})$,

(e) $\mathsf{suff}(C, \mathcal{K}') \subseteq \mathsf{suff}(\mathcal{T}, \mathcal{K})$ *and* $\mathsf{mpl}(C, \mathcal{K}') \leq \mathsf{mpl}(\mathcal{T}, \mathcal{K})$.

Proof. (a) Given a concept name B, by $\mathcal{T}_B$ we denote the set of concept definitions in $\mathcal{T}$ that are used by the run $\mathsf{unfold}(B, \mathcal{T})$. Then by an easy structural induction, we can prove that for each defined concept B in $\mathcal{T}$, we have $\mathsf{rd}(\mathsf{unfold}(B, \mathcal{T})) \leq |\mathcal{T}_B|$, which yields Part (a).

(b) Let $\mathsf{PC}(\mathcal{K}) := \{B \in \mathsf{con}(\mathcal{K}) \mid B \text{ is primitive in } \mathcal{T}\}$ and $\mathsf{DC}(\mathcal{K}) := \{B \in \mathsf{con}(\mathcal{K}) \mid B \text{ is defined in } \mathcal{T}\}$. Because $\langle \mathcal{T}, \mathcal{K} \rangle$ is simple, $\mathsf{con}(\mathcal{K})$ contains concept names only. Therefore, concepts in $\mathsf{con}(\mathcal{K})$ are either primitive in $\mathcal{T}$ or defined in $\mathcal{T}$, i.e., $\mathsf{con}(\mathcal{K}) = \mathsf{PC}(\mathcal{K}) \cup \mathsf{DC}(\mathcal{K})$. Obviously, $|\mathsf{PC}(\mathcal{K})| \leq |\mathsf{con}(\mathcal{K})|$.

Since $C = \mathsf{unfold}(A, \mathcal{T})$ and $\mathcal{K}' = \mathsf{unfold}(\mathcal{K}, \mathcal{T})$, we have that

$$\begin{aligned}
|\mathsf{sub}(C) \cup \mathsf{sub}(\mathsf{con}(\mathcal{K}'))| =& \Big|\mathsf{sub}(\mathsf{unfold}(A, \mathcal{T})) \cup \bigcup_{B \in \mathsf{con}(\mathcal{K})} \mathsf{sub}(\mathsf{unfold}(B, \mathcal{T}))\Big| \\
=& \Big|\mathsf{sub}(\mathsf{unfold}(A, \mathcal{T})) \cup \\
& \qquad \bigcup_{B \in \mathsf{DC}(\mathcal{K})} \mathsf{sub}(\mathsf{unfold}(B, \mathcal{T})) \cup \bigcup_{B \in \mathsf{PC}(\mathcal{K})} B\Big| \\
\leq& \Big|\mathsf{sub}(\mathsf{unfold}(A, \mathcal{T})) \cup \bigcup_{B \in \mathsf{DC}(\mathcal{K})} \mathsf{sub}(\mathsf{unfold}(B, \mathcal{T}))\Big| \\
& + |\mathsf{PC}(\mathcal{K})|.
\end{aligned}$$

Since A and every concept in $\mathsf{DC}(\mathcal{K})$ are defined in $\mathcal{T}$ and $\mathcal{T}$ is simple, it is not difficult to see that each concept definition $B \doteq D \in \mathcal{T}$ that is used by the run $\mathsf{unfold}(A, \mathcal{T})$ or the run $\mathsf{unfold}(B, \mathcal{T})$, for some $B \in \mathsf{DC}(\mathcal{K})$, and each primitive concept in $\mathsf{PC}(\mathcal{T})$ contribute at most one concept into $\mathsf{sub}(\mathsf{unfold}(A, \mathcal{T})) \cup \bigcup_{B \in \mathsf{DC}(\mathcal{K})} \mathsf{sub}(\mathsf{unfold}(B, \mathcal{T}))$. Furthermore, since $\mathcal{T}$ is acyclic, the number of concept definitions in $\mathcal{T}$ is equal to the number of defined concept names in $\mathcal{T}$, i.e., $|\mathsf{DC}(\mathcal{T})|$. Thus, $|\mathsf{sub}(\mathsf{unfold}(A, \mathcal{T})) \cup \bigcup_{B \in \mathsf{DC}(\mathcal{K})} \mathsf{sub}(\mathsf{unfold}(B, \mathcal{T}))|$ is bounded by $|\mathsf{DC}(\mathcal{T})| + |\mathsf{PC}(\mathcal{T})| = |\mathsf{CN}(\mathcal{T})|$.

Summing it up, we have that

$$\begin{aligned}
|\mathsf{sub}(C) \cup \mathsf{sub}(\mathsf{con}(\mathcal{K}'))| &\leq |\mathsf{CN}(\mathcal{T})| + |\mathsf{PC}(\mathcal{K})| \\
&\leq |\mathsf{CN}(\mathcal{T})| + |\mathsf{con}(\mathcal{K})|.
\end{aligned}$$

(c) Since

$$\begin{aligned}
\mathsf{cl}(C, \mathcal{K}') =& \mathsf{sub}(C) \cup \mathsf{cl}(\mathcal{K}') \\
=& \mathsf{sub}(C) \cup \mathsf{sub}(\mathsf{con}(\mathcal{K}')) \cup \{\doteq\neg D \mid D \in \mathsf{sub}(\mathsf{con}(\mathcal{K}'))\},
\end{aligned}$$

it is not difficult to see that $|\mathsf{cl}(C, \mathcal{K}')| \leq 2 \cdot |\mathsf{sub}(C) \cup \mathsf{sub}(\mathsf{con}(\mathcal{K}'))|$.

By Part (b), we have that $|\mathsf{sub}(C) \cup \mathsf{sub}(\mathsf{con}(\mathcal{K}'))| \leq |\mathsf{CN}(\mathcal{T})| + |\mathsf{con}(\mathcal{K})|$. Moreover, it is easily seen that $|\mathsf{CN}(\mathcal{T})| \leq |\mathcal{T}|$ and $|\mathsf{con}(\mathcal{K})| \leq |\mathcal{K}|$. Therefore, $|\mathsf{cl}(C, \mathcal{K}')| \leq 2 \cdot |\mathcal{T}| + |\mathcal{K}|$.

(d) Since A is a concept name and $C = \mathsf{unfold}(A, \mathcal{T})$, we have that every abstract feature and concrete feature appearing in C also appears in $\mathcal{T}$. Therefore, $\mathsf{feat}(C) \subseteq \mathsf{feat}(\mathcal{T})$.

Moreover, since $\langle \mathcal{T}, \mathcal{K} \rangle$ is simple and $\mathcal{K}' = \mathsf{unfold}(\mathcal{K}, \mathcal{T})$, we have that every abstract feature and concrete feature appearing in $\mathcal{K}'$ also appears in $\mathcal{K}$ or $\mathcal{T}$. Thus, $\mathsf{feat}(\mathcal{K}') \subseteq \mathsf{feat}(\mathcal{T}) \cup \mathsf{feat}(\mathcal{K})$.

(e) By a similar argument as above, we have that each path in C or $\mathcal{K}'$ must appear in $\mathcal{T}$ or $\mathcal{K}$. Therefore, obviously $\mathsf{suff}(C, \mathcal{K}') \subseteq \mathsf{suff}(\mathcal{T}, \mathcal{K})$. And as a consequence, $\mathsf{mpl}(C, \mathcal{K}') \leq \mathsf{mpl}(\mathcal{T}, \mathcal{K})$.

□

Corollary 45. *Let A be a concept name, $\langle \mathcal{T}, \mathcal{K} \rangle$ a simple pair of a TBox $\mathcal{T}$ and a key box $\mathcal{K}$, $C = \mathsf{unfold}(A, \mathcal{T})$, and $\mathcal{K}' = \mathsf{unfold}(\mathcal{K}, \mathcal{T})$. Let $S = (T, \mathcal{P}, \prec, \sim)$ with $T = (V_a, V_c, E, \mathcal{L})$ a completion system constructed during the run of the tableau algorithm in Chapter 3 started on the input concept C and the key box $\mathcal{K}'$. Then, the following holds:*

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

(b) $|V_a/{\approx}| \leq ((|\mathcal{T}| + |\mathcal{K}|)^{|\mathcal{T}|} + 1)^{|\mathcal{T}|+|\mathcal{K}|} \cdot 2^{2\cdot(|\mathcal{T}|+|\mathcal{K}|)}$;

(c) There exists a constant k such that $\#V_c + \#V_a$ is bounded by $2^{2^{(|\mathcal{T}|+|\mathcal{K}|)^k}}$.

Proof. (a) By Lemma 26(a), we have that the out-degree of T is bounded by $|\mathsf{sub}(C) \cup \mathsf{sub}(\mathsf{con}(\mathcal{K}'))| + |\mathsf{feat}(C) \cup \mathsf{feat}(\mathcal{K}')|$.

From Lemma 44(b), we have that $|\mathsf{sub}(C) \cup \mathsf{sub}(\mathsf{con}(\mathcal{K}'))| \leq |\mathsf{CN}(\mathcal{T})| + |\mathsf{con}(\mathcal{K})|$. Besides, Lemma 44(d) yields that $\mathsf{feat}(C) \cup \mathsf{feat}(\mathcal{K}') \subseteq \mathsf{feat}(\mathcal{T}) \cup \mathsf{feat}(\mathcal{K})$.

Therefore, we have that the out-degree of T is bounded by

$$\begin{aligned} &|\mathsf{sub}(C) \cup \mathsf{sub}(\mathsf{con}(\mathcal{K}'))| + |\mathsf{feat}(C) \cup \mathsf{feat}(\mathcal{K}')| \\ \leq &|\mathsf{CN}(\mathcal{T})| + |\mathsf{con}(\mathcal{K})| + |\mathsf{feat}(\mathcal{T}) \cup \mathsf{feat}(\mathcal{K})| \\ \leq &|\mathsf{CN}(\mathcal{T})| + |\mathsf{con}(\mathcal{K})| + |\mathsf{feat}(\mathcal{T})| + |\mathsf{feat}(\mathcal{K})| \end{aligned}$$

It is easily seen that $|\mathsf{CN}(\mathcal{T})| + |\mathsf{feat}(\mathcal{T})| \leq |\mathcal{T}|$ and $|\mathsf{con}(\mathcal{K})| + |\mathsf{feat}(\mathcal{K})| \leq |\mathcal{K}|$. Therefore, the out-degree of T is bounded by $|\mathcal{T}| + |\mathcal{K}|$.

(b) By Lemma 26(c), we have that $|V_a/{\approx}| \leq (n+1)^{|\mathsf{suff}(C,\mathcal{K}')|} \cdot 2^{|\mathsf{cl}(C,\mathcal{K}')|}$ where $n = |V_c/{\sim}|$.

According to Lemma 26(b), we have that $|V_c/{\sim}| \leq N^{\mathsf{rd}(C)}$ where $N = |\mathsf{sub}(C) \cup \mathsf{sub}(\mathsf{con}(\mathcal{K}'))| + |\mathsf{feat}(C) \cup \mathsf{feat}(\mathcal{K}')|$. Part (a) yields that $N \leq |\mathcal{T}| + |\mathcal{K}|$ whereas Lemma 44(a) implies that $\mathsf{rd}(C) \leq |\mathcal{T}|$. Therefore, $|V_c/{\sim}| \leq (|\mathcal{T}| + |\mathcal{K}|)^{|\mathcal{T}|}$.

Lemma 44(e) yields that $\mathsf{suff}(C, \mathcal{K}') \subseteq \mathsf{suff}(\mathcal{T}, \mathcal{K})$. Furthermore, it is not difficult to show that $|\mathsf{suff}(\mathcal{T}, \mathcal{K})| \leq |\mathcal{T}| + |\mathcal{K}|$. Thus, $|\mathsf{suff}(C, \mathcal{K}')| \leq |\mathcal{T}| + |\mathcal{K}|$.

Summing all the above up, together with the fact that $|\mathsf{cl}(C, \mathcal{K}')| \leq 2 \cdot (|\mathcal{T}| + |\mathcal{K}|)$ by Lemma 44(c), we obtain that

$$|V_a/{\approx}| \leq ((|\mathcal{T}| + |\mathcal{K}|)^{|\mathcal{T}|} + 1)^{|\mathcal{T}|+|\mathcal{K}|} \cdot 2^{2\cdot(|\mathcal{T}|+|\mathcal{K}|)}.$$

(c) According to Lemma 26(d), we have that the depth of T is bounded by $|V_a/{\approx}| + \mathsf{mpl}(C, \mathcal{K}')$. Part (b) implies that

$$|V_a/{\approx}| \leq ((|\mathcal{T}| + |\mathcal{K}|)^{|\mathcal{T}|} + 1)^{|\mathcal{T}|+|\mathcal{K}|} . 2^{2^{2\cdot(|\mathcal{T}|+|\mathcal{K}|)}}.$$

Furthermore, Lemma 44(e) yields that $\mathsf{mpl}(C_0, \mathcal{K}') \leq \mathsf{mpl}(\mathcal{T}, \mathcal{K})$. Obviously, $\mathsf{mpl}(\mathcal{T}, \mathcal{K}) \leq |\mathcal{T}| + |\mathcal{K}|$. Therefore, there exists a constant m such that the depth of T is bounded by $2^{(|\mathcal{T}|+|\mathcal{K}|)^m}$.

Since the out-degree of T is bounded by $|\mathcal{T}| + |\mathcal{K}|$ by Part (a) and the depth of T is bounded by $2^{(|\mathcal{T}|+|\mathcal{K}|)^m}$, we have that the number of nodes in T is bounded by

$$\frac{(|\mathcal{T}| + |\mathcal{K}|)^{2^{(|\mathcal{T}|+|\mathcal{K}|)^m}+1} - 1}{|\mathcal{T}| + |\mathcal{K}| - 1}.$$

Therefore, there is a constant k such that the number of nodes in T is bounded by $2^{2^{(|\mathcal{T}|+|\mathcal{K}|)^k}}$. The fact that $V_a \cup V_c$ is the set of nodes in T finishes the proof.

□

Based on the correspondence between runs of the original tableau algorithm and runs of the modified algorithm, established in Lemma 43, and the upper bound for the size of the tree T constructed by the algorithm, established in Corollary 45, we get the following.

Lemma 46. *The modified completion algorithm is sound, complete, and terminating. Moreover, if it is started on a concept name A and a simple pair $\langle \mathcal{T}, \mathcal{K} \rangle$, and returns* satisfiable*, then there exists a model for $A, \mathcal{T}$, and $\mathcal{K}$ whose size is not greater than M, where*

$$M = ((|\mathcal{T}| + |\mathcal{K}|)^{|\mathcal{T}|} + 1)^{|\mathcal{T}|+|\mathcal{K}|} \cdot 2^{2\cdot(|\mathcal{T}|+|\mathcal{K}|)}.$$

Proof. Soundness and completeness immediately follows from Lemma 43 and

1. the original tableau algorithm is sound, complete, and terminating;
2. the concept name A and the simple pair $\langle \mathcal{T}, \mathcal{K} \rangle$ have a model iff the concept $C = \mathsf{unfold}(A, \mathcal{T})$ and the keybox $\mathcal{K}' = \mathsf{unfold}(\mathcal{K}, \mathcal{T})$ have a model.

The termination is proved similarly to Lemmas 28 and 29, together with Corollary 45 which states that the completion tree T is of bounded size.

For the size of the constructed model, it is obvious that $|\Delta_{\mathcal{I}}| \leq |V_a/{\approx}|$. By Corollary 45, we have that

$$|V_a/{\approx}| \leq ((|\mathcal{T}| + |\mathcal{K}|)^{|\mathcal{T}|} + 1)^{|\mathcal{T}|+|\mathcal{K}|} \cdot 2^{2\cdot(|\mathcal{T}|+|\mathcal{K}|)}.$$

□

It is easily seen that similarly to the tableau algorithm for $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$-concept satisfiability, the modified completion algorithm runs in time double exponential in $|\mathcal{T}|+|\mathcal{K}|$. Fortunately, as established in Lemma 46, the constructed model is of bounded size exponential in $|\mathcal{T}|+|\mathcal{K}|$. Similarly to Theorem 34, that fact suggests a NExpTime upper complexity bound for $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$-concept satisfiability w.r.t. acyclic TBoxes and safe key boxes, which coincides with the NExpTime lower complexity bound established in Theorem 17.

Theorem 47. *For a key-admissible concrete domain $\mathcal{D}$ such that extended $\mathcal{D}$-satisfiability is in NP, $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$-concept satisfiability w.r.t. acyclic TBoxes and safe key boxes is in* NExpTime.

Proof. We consider satisfiability of a concept name A w.r.t. a simple pair $\langle \mathcal{T}, \mathcal{K} \rangle$. Let $C = \mathsf{unfold}(A, \mathcal{T})$ and $\mathcal{K}' = \mathsf{unfold}(\mathcal{K}, \mathcal{T})$.

Let

$$\mathsf{B}(\mathcal{T}, \mathcal{K}) = ((|\mathcal{T}| + |\mathcal{K}|)^{|\mathcal{T}|} + 1)^{|\mathcal{T}|+|\mathcal{K}|} \cdot 2^{2\cdot|\mathcal{T}|+|\mathcal{K}|)};$$

and $V = \{v_1, \ldots, v_n\}$ where $n \leq \mathsf{B}(\mathcal{K}, \mathcal{T}) \cdot (|\mathcal{T}| + |\mathcal{K}|)$.

A quasi-interpretation for A, $\mathcal{T}$, and $\mathcal{K}$ is defined similarly to that in the proof of Theorem 34, except that

- $\Delta_{\mathcal{M}}$ is a non-empty set such that $|\Delta_{\mathcal{M}}| \leq \mathsf{B}(\mathcal{T}, \mathcal{K})$;
- beside conditions (C1), (C2), and (C3), to be a *quasi-model* for A, $\mathcal{T}$, and $\mathcal{K}$ a quasi-interpretation must satisfy the following additional condition:

  (C4) $\mathcal{M}$ satisfies all concept definitions in $\mathcal{T}$.

It is not difficult to see that there is a quasi-model for the concept name A, the TBox $\mathcal{T}$, and the key box $\mathcal{K}$ iff A is satisfiable w.r.t. $\mathcal{T}$ and $\mathcal{K}$.

Based on quasi-models, we present an alternative decision procedure for deciding satisfiability of concept names w.r.t. simple pairs of TBoxes key boxes. First, we "guess" a quasi-interpretation for A, $\mathcal{T}$, and $\mathcal{K}$, then we check whether it is a quasi-model for A, $\mathcal{T}$, and $\mathcal{K}$.

Checking the condition (C2), i.e., if there exists an $a \in \Delta_{\mathcal{M}}$ such that $a \in A^{\mathcal{M}}$, is easy because A is a concept name. Checking the condition (C4), i.e., if $\mathcal{M}$ is a model for $\mathcal{T}$ is done via the following algorithm. Let $\mathcal{T} = \{A_1 \doteq C_1, \ldots, A_n \doteq C_n\}$, the algorithm is given in pseudo code as follows:

for $i := 1$ to n **do**
  if $A_i^{\mathcal{M}} \neq C_i^{\mathcal{M}}$ **then**
    return false
return true

It is easy to see that every step of the algorithm can be carried out in time $\mathcal{O}(|\Delta_{\mathcal{M}}|)$, which is at most exponential in $|\mathcal{T}| + |\mathcal{K}|$. Furthermore, the number of steps is bounded the number of concept definitions in $\mathcal{T}$, which in turn is bounded by $|\mathcal{T}|$. Therefore, the algorithm can be carried out in time exponential in $|\mathcal{T}| + |\mathcal{K}|$.

Finally, conditions (C1) and (C3) are checked similarly as in the proof of Theorem 34.

Since all described parts of the non-deterministic algorithm runs in exponential time, we conclude that $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$-concept satisfiability w.r.t. acyclic TBoxes and safe key boxes can be decided in NExpTime if extended $\mathcal{D}$-satisfiability is in NP. □

4.2 $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$ with General TBoxes

Generalizing the notion of acyclic TBoxes, we obtain general TBoxes defined as follows.

Definition 48 (General TBox)**.** *An expression of the form $C \sqsubseteq D$, where C and D are concepts, is called a* general concept inclusion axiom *(or GCI for short). A finite set $\mathcal{T}$ of GCIs is called a* general TBox*.*

An interpretation $\mathcal{I}$ is a model *of a TBox $\mathcal{T}$ iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for all $C \sqsubseteq D \in \mathcal{T}$. A concept C is* satisfiable w.r.t. a TBox $\mathcal{T}$ and a key box $\mathcal{K}$ *iff there exists a model of $C, \mathcal{T}$, and $\mathcal{K}$. A concept D* subsumes *a concept C* w.r.t. a TBox $\mathcal{T}$ and a key box $\mathcal{K}$*, written $C \sqsubseteq_{\mathcal{T},\mathcal{K}} D$, iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for all models $\mathcal{I}$ of $\mathcal{T}$ and $\mathcal{K}$.*

Note that a quite common form of TBoxes is obtained by replacing the subset relation "$\sqsubseteq$" in GCIs by the equality "$\doteq$", and in that case each expression $C \doteq D$ is called a *concept equation* (see, e.g., [6, 14]). It is easily seen that the two forms of TBoxes are equivalent since each GCI $C \sqsubseteq D$ can be expressed

by a concept equation $C \doteq C \sqcap D$ and each concept equation $C \doteq D$ can be represented by two GCIs: $C \sqsubseteq D$ and $D \sqsubseteq C$.

General TBoxes are a powerful tool for describing background knowledge about application domains and have been considered in the literature (e.g. [2, 10, 14]). However, in most description logics, the presence of general TBoxes increases the complexity of reasoning significantly. For example, while reasoning in $\mathcal{ALC}$ is PSPACE-complete, referring to general TBoxes makes reasoning in $\mathcal{ALC}$ become EXPTIME-complete (see, e.g., [2]).

In [18], it is shown that for a particular concrete domain W, which was introduced in [15] and is given in the following definition, even reasoning w.r.t. sets of concept equations in $\mathcal{ALC}(\mathsf{W})$, a fragment of $\mathcal{ALCOK}(\mathsf{W})^{\mathcal{FD}}$, becomes undecidable.

Definition 49 (Concrete domain W)**.** *Let Σ be an alphabet. The concrete domain W is defined by setting $\Delta_{\mathsf{W}} := \Sigma^*$ and defining Φ_{W} as the smallest set containing the following predicates:*

- *unary predicate* word *and* nword *with* $\mathsf{word}^{\mathsf{W}} = \Delta_{\mathsf{W}}$ *and* $\mathsf{nword}^{\mathsf{W}} = \emptyset$*,*
- *unary predicate $=_\epsilon$ 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*

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

As shown in [15], W-satisfiability is in PTIME, and thus W is admissible. This is important since it guarantees that the undecidablity of reasoning in $\mathcal{ALC}(\mathsf{W})$ is not due to undecidability of W-satisfiability.

The undecidability result is shown by a reduction of the well-known undecidable *Post Correspondence Problem (PCP)* [23], which is to be defined explicitly in Section 4.3, to satisfiability of $\mathcal{ALC}(\mathsf{W})$-concepts. Since GCIs and concept equations have the same expressive power, the undecidability result also holds for reasoning w.r.t. general TBoxes.

Lemma 50 (Theorem 5.24 in [18])**.** *$\mathcal{ALC}(\mathsf{W})$-concept satisfiability w.r.t. general TBoxes is undecidable.*

From the fact that $\mathcal{ALC}(\mathsf{W})$ is a fragment of $\mathcal{ALCOK}(\mathsf{W})^{\mathcal{FD}}$ and Lemma 50, we have the following undecidability result.

Corollary 51. *$\mathcal{ALCOK}(\mathsf{W})^{\mathcal{FD}}$-concept satisfiability w.r.t. general TBoxes is undecidable.*

Moreover, to emphasize that this undecidability result is obtained using a simple concrete domain, we combine Corollary 51 and the fact that W-satisfiability is in PTIME as follows.

Theorem 52. *There exists a concrete domain $\mathcal{D}$ such that $\mathcal{D}$-satisfiability is in* PTIME *and reasoning in $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$ is undecidable.*

Note that the concrete domain W seems unnatural and thus the obtained undecidability result may be not relevant. However, in [14] it is shown that words over the alphabet Σ can be encoded as numbers in base $\#\Sigma+1$ (without "0" digit). Therefore, the corresponding natural numbers in base 10 can be used to represent non-empty words, and 0 to represent the empty word. The concatenation of two words v and w can be represented as $vw = v \cdot (\#\Sigma + 1)^{|w|} + w$, where $|w|$ denoted the length of the word w. Hence, the concrete domain W can be replaced by an arithmetic one. Moreover, with the help of general TBoxes, similarly to [18], we can express exponentiation as multiple multiplications, multiplications as additions, and additions as incrementations with one. Therefore, arithmetic concrete domains can be replaced by weaker concrete domains.

Theorem 53. *Let $\mathcal{D}$ be a concrete domain. If $\mathbb{N} \subseteq \Delta_{\mathcal{D}}$ and $\Phi_{\mathcal{D}}$ provides a unary predicate for equality with 0, a binary equality predicate, and a binary predicate for incrementation, then $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$-concept satisfiability w.r.t. general TBoxes is undecidable.*

Because concept subsumption can be reduced to concept satisfiability, unsatisfiability of concept subsumption w.r.t. general TBoxes in $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$ with concrete domains specified in Theorem 53 obviously follows.

4.3 Adding Inverse Roles

In this section, we consider the description logic $\mathcal{ALCOIK}(\mathcal{D})^{\mathcal{FD}}$, which is obtained by extending $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$ with inverse roles. More precisely, the logic $\mathcal{ALCOIK}(\mathcal{D})^{\mathcal{FD}}$ allows the use of inverse roles inside subconcepts of the form $\exists R.C$ or $\forall R.C$. The reason for not admitting inverse roles of abstract features inside the concrete domain constructor is that inverse roles of abstract features are not necessarily functional.

The semantics of inverse roles is defined in the obvious way. Given an interpretation $\mathcal{I}$ and a role name S, $\mathcal{I}$ is extended to inverse roles as follows

$$(S^-)^{\mathcal{I}} := \{(b, a) \mid (a, b) \in S^{\mathcal{I}}\}.$$

In [14, 18], the description logic $\mathcal{ALCI}(\mathcal{D})$ (denoted as $\mathcal{ALC}^-(\mathcal{D})$ in [14, 18]), which is $\mathcal{ALC}(\mathcal{D})$ extended with inverse roles, is proposed. It is proved that for an arithmetic concrete domain $\mathcal{D}$, $\mathcal{ALCI}(\mathcal{D})$-concept satisfiability is NExpTime-hard. Interestingly, it can be shown that adding nominals to $\mathcal{ALCI}(\mathcal{D})$, which yields the description logic $\mathcal{ALCOI}(\mathcal{D})$, makes a dramatic jump in complexity of reasoning from NExpTime-hard to undecidable for an arithmetic concrete domain $\mathcal{D}$. The idea is based on the undecidability of reasoning w.r.t. general TBoxes in $\mathcal{ALC}(\mathcal{D})$ with an arithmetic concrete domain $\mathcal{D}$ and the fact that using the "spy-point" technique as known from [1], we can "simulate" general TBoxes by inverse roles and nominals.

Let W be the concrete domain introduced in Definition 49, which is used in [18] to show that PCP can be reduced to satisfiability of $\mathcal{ALC}(\mathsf{W})$-concepts w.r.t. general TBoxes. We adapt the reduction and the "spy-point" technique in order to obtain undecidability result for reasoning in $\mathcal{ALCOI}(\mathsf{W})$. First, let us explicitly define the well-known undecidable Post Correspondence Problem.

Definition 54 (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*

$$l_{i_1} \ldots l_{i_m} = r_{i_1} \ldots r_{i_m}.$$

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

We can now discuss the reduction of the PCP to $\mathcal{ALCOI}(\mathsf{W})$-concept satisfiability. Given an instance $P = (l_1, r_1), \ldots, (l_k, r_k)$ of the PCP, we construct the following concepts:

$$C_R := \exists l.=_\epsilon \sqcap \exists r.=_\epsilon$$

$$C_\mathcal{T} := \Big(\bigsqcap_{(l_i, r_i) \in P} \exists l, f_i l.\mathsf{conc}_{l_i} \sqcap \exists r, f_i r.\mathsf{conc}_{r_i} \Big) \sqcap \big(\exists l.=_\epsilon \sqcup \neg \exists l, r.= \big)$$

$$C_P := N_\mathsf{spy} \sqcap \exists u.C_R \sqcap \forall u.\Big(\bigsqcap_{1 \leq i \leq k} \forall f_i.\exists u^-.N_\mathsf{spy} \Big) \sqcap \forall u.C_\mathcal{T}$$

in which N_spy is a nominal, $f_1, \ldots, f_k$ are abstract features, u is a role name, and l, r are concrete features. The idea behind the reduction is that a model of C_P encodes all potential solutions for the PCP P, and moreover, the existence of such a model guarantees that no potential solution actually is a solution. Hence, P has a solution iff C_P is unsatisfiable. An example model is depicted in Figure 4.1. The following lemma formulates the reduction formally.

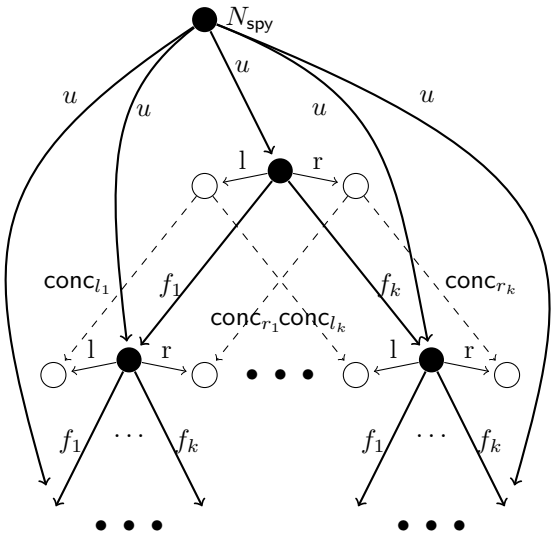

Figure 4.1: An example model of C_P

Lemma 55. *Let* $P = (l_1, r_1), \ldots, (l_k, r_k)$ *be a PCP. Then,* P *has a solution iff* C_P *is unsatisfiable.*

Proof. For both directions, we show the contrapositive.

($\Leftarrow$) Assuming that P has no solution, we show that C_P is satisfiable by constructing an interpretation $\mathcal{I}$ that is a model for C_P. If $w = i_1, \ldots, i_n$ is a sequence of indices, we use $\mathsf{leftconc}(w)$ to denote the concatenation of the words $l_{i_1}, \ldots, l_{i_n}$ and $\mathsf{rightconc}(w)$ to denote the concatenation of the words $r_{i_1}, \ldots, r_{i_n}$. We define

$$\begin{aligned}
\Delta_{\mathcal{I}} &:= \{i_1 \ldots i_n \mid n \geq 0 \text{ and } 1 \leq i_j \leq k \text{ for } 1 \leq j \leq n\} \cup \{\mathsf{spy}\},\\
N^{\mathcal{I}}_{\mathsf{spy}} &:= \{\mathsf{spy}\},\\
f^{\mathcal{I}}_i(w) &:= wi \text{ for } w \in \Delta_{\mathcal{I}} \text{ and } 1 \leq i \leq k,\\
l^{\mathcal{I}}(w) &:= \mathsf{leftconc}(w) \text{ for } w \in \Delta_{\mathcal{I}},\\
r^{\mathcal{I}}(w) &:= \mathsf{rightconc}(w) \text{ for } w \in \Delta_{\mathcal{I}},\\
u^{\mathcal{I}} &:= \{(\mathsf{spy}, a) \mid a \in \Delta_{\mathcal{I}} \backslash \{\mathsf{spy}\}\}.
\end{aligned}$$

Note that $\Delta_{\mathcal{I}}$ also contains the empty sequence of indices. Since P has no solution, it is readily checked that $\mathcal{I}$ is a model for C_P.

($\Rightarrow$) Suppose that C_P is satisfied by a model $\mathcal{I}$. Let $\mathsf{spy}, a_0 \in \Delta_{\mathcal{I}}$ be such that $N^{\mathcal{I}}_{\mathsf{spy}} = \{\mathsf{spy}\}$, $(\mathsf{spy}, a_0) \in u^{\mathcal{I}}$, and $a_0 \in C^{\mathcal{I}}_R$.

We must show that P has no solution. Assume to the contrary that $w = i_1, \ldots, i_n$ is a solution for P. By induction on j, it is easy to show that $(f_{i_1} \ldots f_{i_j} l)^{\mathcal{I}}(a_0) = \mathsf{leftconc}(i_1, \ldots, i_j)$ and $(f_{i_1} \ldots f_{i_j} r)^{\mathcal{I}}(a_0) = \mathsf{rightconc}(i_1, \ldots, i_j)$ for $1 \leq j \leq n$.

Since $\mathsf{spy} \in C^{\mathcal{I}}_P$ and $(\mathsf{spy}, a_0) \in u^{\mathcal{I}}$, we have that $a_0 \in C^{\mathcal{I}}_{\mathcal{T}}$. Let $b = (f_{i_1} \ldots f_{i_n})^{\mathcal{I}}(a_0)$. By induction on n, it is easy to show that $(\mathsf{spy}, b) \in u^{\mathcal{I}}$, and thus $b \in (C_{\mathcal{T}})^{\mathcal{I}}$, which yields that $b \in (\exists l. =_\epsilon \sqcup \neg\exists l, r. =)^{\mathcal{I}}$. Since $n \geq 1$, it is easily seen that $b \notin (\exists l. =_\epsilon)^{\mathcal{I}}$, and thus $b \in (\neg\exists l, r. =)^{\mathcal{I}}$. Therefore, $l^{\mathcal{I}}(b) \neq r^{\mathcal{I}}(b)$.

Therefore, since $f_{i_1}, \ldots, f_{i_n}$ are abstract features, we clearly have that $(f_{i_1} \ldots f_{i_n} l)^{\mathcal{I}}(a_0) \neq (f_{i_1} \ldots f_{i_n} r)^{\mathcal{I}}(a_0)$, and thus $\mathsf{leftconc}(w) \neq \mathsf{rightconc}(w)$. Hence, w is no solution to P, which is a contradiction to what has been assumed.

□

Clearly, the size of the concept C_P is polynomial in k. Therefore, we obtain the following.

Corollary 56. *$\mathcal{ALCOI}(\mathsf{W})$-concept satisfiability is undecidable.*

From the fact that $\mathcal{ALCOI}(\mathsf{W})$ is a fragment of $\mathcal{ALCIOK}(\mathsf{W})^{\mathcal{FD}}$, we have the following.

Corollary 57. *Reasoning in $\mathcal{ALCOIK}(\mathsf{W})^{\mathcal{FD}}$ is undecidable.*

Moreover, to emphasize that this undecidability result is obtained using a simple concrete domain, we combine Corollary 57 and the fact that W-satisfiability is in PTime as follows.

Theorem 58. *There exists a concrete domain $\mathcal{D}$ such that $\mathcal{D}$-satisfiability is in* PTime *and reasoning in $\mathcal{ALCOIK}(\mathcal{D})^{\mathcal{FD}}$ is undecidable.*

As dicussed earlier, the concrete domain W can be replaced by an arithmetic one. Moreover, it is easily noticed that the sub-concept $C_{\mathcal{T}}$ in C_P is satisfied in every element of the logical domain that is "reachable" from N_{spy} via the role name u, and thus it can be used to simulate general TBoxes. Hence, we can express exponentiation as multiple multiplications, multiplications as additions, and additions as incrementations with one. Therefore, arithmetic concrete domains can be replaced by weaker concrete domains.

Theorem 59. *Let $\mathcal{D}$ be a concrete domain. If $\mathbb{N} \subseteq \Delta_{\mathcal{D}}$ and $\Phi_{\mathcal{D}}$ provides a unary predicate for equality with 0, a binary equality predicate, and a binary predicate for incrementation, then reasoning in $\mathcal{ALCOIK}(\mathcal{D})^{\mathcal{FD}}$ is undecidable.*

Chapter 5

Conclusion

In this paper, we have investigated the first description logic with concrete domains and both types of key constraints, namely uniqueness constraints and functional dependencies, thus completing previous work where each of them was treated separately. More precisely, we have combined both uniqueness constraints and functional dependencies in the description logic $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$. Moreover, we have also given a discussion on the impact on complexity of reasoning of adding acyclic TBoxes, general TBoxes, or inverse roles to $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$.

Following immediately from the undecidability results of reasoning in DL with uniqueness constraints [19] and DL with functional dependencies [20], we have discussed about that in the general case, reasoning in $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$ is undecidable and in order to preserve decidability, a restricted class of key boxes is needed. The identified key boxes are safe key boxes, which additionally allow for sub-concepts of the forms $\exists R.C, \forall R.C$, and $g\uparrow$ besides Boolean connectives in concepts occurring in key constraints. Moreover, being a super-logic of $\mathcal{ALCOK}(\mathcal{D})$ and $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$, $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$ inherits the NExpTime lower complexity bound for reasoning w.r.t. safe key boxes from these logics.

The main contribution of this paper is the tableau algorithm deciding satisfiability of $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$-concepts w.r.t. safe key boxes. It turns out that a naive combination of completion rules for DL with uniqueness constraints in [19] and those for DL with functional dependencies in [20] does not work. The reason is that we need to take into account two different sources influencing the structure of logical models, namely uniqueness constraints and strong functional dependencies. Due to the same reason, the blocking mechanism for ensuring termination of the algorithm in this paper is more complicated than that in [20] whereas a blocking mechanism is unnecessary in the case of uniqueness constraints only. Moreover, based on the bounded model property induced by the proofs of termination, soundness, and completeness of the tableau algorithm, we have shown the NExpTime upper complexity bound for reasoning w.r.t. safe key boxes in $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$, which coincides with the NExpTime lower complexity bound. Therefore, implicitly we have shown that we can treat a larger class of uniqueness constraints than ones from [19], which only allow for Boolean combinations of concept names in uniqueness constraints.

Extending $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$, we have considered impact of acyclic TBoxes, general TBoxes, and inverse roles on complexity of reasoning in $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$. Unfortunately, most result that we have come up with is discouraging, i.e.,

providing $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$ with general TBoxes and inverse roles leads to undecidability for a large class of concrete domains $\mathcal{D}$. Besides the discouraging results, we have shown that reasoning w.r.t. acyclic TBoxes is still decidable and adding acyclic TBoxes to $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$ does not make reasoning harder in complexity.

For future work, it would be useful to integrate both uniqueness constraints and functional dependencies into more expressive DLs with concrete domains, e.g. $\mathcal{SHOQ}(\mathcal{D})$. In fact, an extension of $\mathcal{SHOQ}(\mathcal{D})$ with uniqueness constraints, called $\mathcal{SHOQK}(\mathcal{D})$, is analyzed in [19]. As it is well-known that combining general TBoxes, which is a very important feature of $\mathcal{SHOQ}(\mathcal{D})$, and the concrete domain constructor easily leads to undecidability (see, e.g., [18]), only a path-free variant of the concrete domain constructor, i.e., only concrete features are admitted inside the constructor, is allowed in order to regain decidability [12]. Furthermore, in $\mathcal{SHOQK}(\mathcal{D})$ only path-free uniqueness constraints are considered. As proved in [19], reasoning in the restricted $\mathcal{SHOQK}(\mathcal{D})$ is NExpTime-complete. For an extension of $\mathcal{SHOQ}(\mathcal{D})$ with both types of key constraints, since we restricted ourselves to the path-free variant of the concrete domain constructor and path-free key boxes, functional dependencies cannot influence the structure of logical models. Thus, we believe that upper complexity bounds can be preserved and the algorithm can be easily extended in order to treat path-free key constraints.

Besides, it would be interesting to know if we can combine both types of key constraints with number restrictions, transitive roles, or role hierarchies in order to preserve decidability in the case that path-freeness of key constraints is not strictly required. We are also interested in integrating DLs with uniqueness constraints and functional dependencies and action formalisms [5].

It remains to do the implementation of the algorithm for $\mathcal{ALCOK}(\mathcal{D})^{\mathcal{FD}}$ presented in this paper, and moreover, we find it challenging to do an investigation on optimization techniques and implement the algorithm efficiently.

Bibliography

[1] Areces, C., Blackburn, P., and Marx, M. A road-map on complexity for hybrid logics. In *Computer Science Logic* (Madrid, Spain, 1999), J. Flum and M. Rodríguez-Artalejo, Eds., no. 1683 in LNCS, Springer, pp. 307–321.

[2] Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D., and Patel-Schneider, P. F., Eds. *The Description Logic Handbook: Theory, Implementation, and Applications* (2003), Cambridge University Press.

[3] Baader, F., and Hanschke, P. A scheme for integrating concrete domains into concept languages. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence, IJCAI-91* (Sydney (Australia), 1991), pp. 452–457.

[4] Baader, F., Horrocks, I., and Sattler, U. Description logics as ontology languages for the semantic web. In *Mechanizing Mathematical Reasoning: Essays in Honor of Jörg H. Siekmann on the Occasion of His 60th Birthday*, D. Hutter and W. Stephan, Eds., vol. 2605 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 2005, pp. 228–248.

[5] Baader, F., Lutz, C., Milicic, M., Sattler, U., and Wolter, F. Integrating description logics and action formalisms: First results. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05)* (Pittsburgh, PA, USA, 2005).

[6] Baader, F., and Sattler, U. An overview of tableau algorithms for description logics. *Studia Logica 69* (2001), 5–40.

[7] Borgida, A., and Weddell, G. E. Adding uniqueness constraints to description logics (preliminary report). In *Deductive and Object-Oriented Databases* (1997), pp. 85–102.

[8] Calvanese, D., De Giacomo, G., and Lenzerini, M. Identification constraints and functional dependencies in description logics. In *Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001)* (2001), pp. 155–160.

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

[10] Donini, F. M., and Massacci, F. ExpTime tableaux for $\mathcal{ALC}$. *Artificial Intelligence 124*, 1 (2000), 87–138.

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

[12] Horrocks, I., and Sattler, U. Ontology reasoning in the $\mathcal{SHOQ}$(D) description logic. In *Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001)* (2001), Morgan Kaufmann, Los Altos, pp. 199–204.

[13] Khizder, V. L., Toman, D., and Weddell, G. On decidability and complexity of description logics with uniqueness constraints. *Lecture Notes in Computer Science 1973* (2001), 54–67.

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

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

[16] Lutz, C. Reasoning about entity relationship diagrams with complex attribute dependencies. In *Proceedings of the 2002 International Workshop on Description Logics* (2002), vol. 53 of *CEUR-WS*, pp. 185–194.

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

[18] Lutz, C. NExpTime-complete description logics with concrete domains. *ACM Transactions on Computational Logic 5*, 4 (2004), 669–705.

[19] Lutz, C., Areces, C., Horrocks, I., and Sattler, U. Keys, nominals, and concrete domains. *Journal of Artificial Intelligence Research 23* (2005), 667–726.

[20] Lutz, C., and Milicic, M. Description logics with concrete domains and functional dependencies. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI-2004)* (2004), pp. 378–382.

[21] Lutz, C., and Milicic, M. Description logics with concrete domains and functional dependencies. LTCS-Report LTCS-04-06, Chair for Automata Theory, Institute for Theoretical Computer Science, Dresden University of Technology, Germany, 2004. See http://lat.inf.tu-dresden.de/research/reports.html.

[22] Nebel, N. Terminological reasoning is inherently intractable. *Artificial Intelligence 43* (1990), 235–249.

[23] Post, E. M. A variant of a recursively unsolvable problem. *Bullentin of the American Mathematical Society 52* (1946), 264–268.

[24] Schmidt-Schauss, M., and Smolka, G. Attributive concept descriptions with complements,. *Artificial Intelligence 48* (1991), 1–26.

[25] Toman, D., and Weddell, G. E. On path-functional dependencies as first-class citizens in description logics. In *Proceedings of the 2005 International Workshop on Description Logics (DL2005)* (2005), no. 147 in CEUR-WS.

[26] Toman, D., and Weddell, G. E. On the interaction between inverse features and path-functional dependencies in description logics. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI-05)* (2005), pp. 603–608.