
Faculty of Computer Science

Institute of Theoretical Computer Science

Chair of Automata Theory

Prof. Dr.-Ing. Franz Baader

Master's Thesis

# A Tableau Algorithm for the Numerical Description Logic $\mathcal{ALCSCC}$

Ryny Khy

**Matriculation Number:** 4751049
born on 30. November 1994 in Landshut

supervised by
Dr.-Ing. Stefan Borgwardt
Filippo De Bortoli M.Sc.

reviewed by
Dr.-Ing. Stefan Borgwardt
Prof. Sebastian Rudolph

December 15, 2020

# Declaration of Authorship

**Author:** Ryny Khy
**Matriculation Number:** 4751049
**Title:** A Tableau Algorithm for the Numerical Description Logic $\mathcal{ALCSCC}$

I hereby declare that I have written this final thesis independently and have listed all used sources and aids. I am submitting this thesis for the first time as a piece of assessed academic work. I understand that attempted deceit will result in the failing grade "not sufficient" (5.0).

| | |
|---|---|
| Place and Date | Author's signature |

**Abstract**

In the research field of Description Logics (DLs) checking satisfiability of $\mathcal{ALCQ}$ has been investigated thoroughly and is therefore well-known. The DL $\mathcal{ALCSCC}$ extend $\mathcal{ALCQ}$ with constraints over role successor using quantifier-free fragment (QF) of Boolean Algebra (BA) and Presburger Arithmetic (PA). Checking satisfiability of this DL has been proven to be decidable and PSpace-complete. We provide in this work a tableau algorithm for checking satisfiability of $\mathcal{ALCSCC}$ and its correctness proof.

# Contents

# Chapter 1

# Introduction

## 1.1 Motivation

In traditional databases stored data objects do not have any relation with each other unless explicitly stated. However we can extract additional information about these objects if we use database systems which employ *semantics*. Imagine we want to add data entries for two people (Anna and Beth) to a traditional database. Anna is a teacher at the local school. Beth is a student of her class. In our database we save their names, the class that Beth attends as well as the class that Anna teaches. As we do not explicitly encode their student-teacher relationship the traditional database does not know about it. If we use an ontology-based system we can deduce this information by making use of semantics. These semantics are described by a set of rules (axioms). In our example one axiom would be that if a teacher teaches a class and students attend the same class, then there is a student-teacher relationship between the teacher and these students. By applying this axiom the ontology-based database can automatically deduce that Anna is the teacher of Beth. Popular use-cases for ontology-based systems are databases for biological and medical research [4]. As an example ontologies can be used to automatically fill in missing information about patients which are helpful in diagnostics. Another major use-case for ontologies is the *Semantic Web*, which is an extension of the World Wide Web with standards given by the World Wild Web Consortium (W3C)[1]. These standards allow a more effective way of combining information from different sources.
An Ontology (in the field of computer science) can be viewed as formal representation of a certain domain of interest. The relationships between entities in an ontology-based database are formulated by a fragment of first-order logic (FOL). This fragment of FOL is called *Description Logic (DL)* and is a family of knowledge representation systems. DLs mainly consist of concepts, which correspond to unary relations in FOL, and relations between the concepts, which

[1] `https://www.w3.org/standards/semanticweb/`, last accessed on December 15, 2020

correspond to binary relations in FOL. To create more complex (compound) concepts we can combine concepts by using operators like $\sqcap$, $\sqcup$, $\sqsubseteq$, $\exists$ and $\forall$. For example the statement "All Humans who have children are parents" can be formalized in DL as $Human \sqcap \exists hasChild.\top \sqsubseteq Parent$, where $Human$ and $Parent$ are concept names and $hasChild$ is a role name. This statement can also be formalized with a a numerical restriction: $Human \sqcap \geq 1 hasChild.\top \sqsubseteq Parent$. A knowledge base consists of a *TBox*, which contains the axioms, and an *ABox*, which contains assertions about certain individual names (objects).
The process of determining whether some statement can be concluded from a set of information is called *reasoning*. Reasoning can be done by adding this statement in negated form (as an axiom or assertion) to the set of information (TBox or ABox) and then checking whether the updated knowledge base is now *unsatisfiable*. If it is unsatisfiable the statement can be concluded from the information set. Being able to check the satisfiability of DL statements is therefore a valuable tool to conduct reasoning in ontology systems. The DL $\mathcal{ALCQ}$ [5][9] has been investigated thoroughly and therefore we know a lot about its satisfiability. This DL allows conjunctions ($\sqcap$), disjunctions ($\sqcup$), negations ($\neg$) and number restrictions ($\leq\ n\, r\, C$ and $\geq\ n\, r\, C$, where $n$ is a number, $r$ a role name and $C$ a concept name). In [5] Hollunder and Baader proved that checking satisfiability of a $\mathcal{ALCQ}$ concept without a TBox is in PSpace and otherwise in ExpTime.
The DL $\mathcal{ALCSCC}$ [1] extends $\mathcal{ALCQ}$ with *set constraints* and *cardinality constraints* over role successors, which use the logic of *QFBAPA* (quantifier-free fragment of Boolean Algebra with Presburger Arithmetic)[7]. Instead of the quantifiers $\exists$ and $\forall$ we use set expressions (Boolean Algebra) and numerical constraints (Presburger Arithmetic). For example $Human \sqcap \geq 1\, hasChild.\top \sqsubseteq Parent$ is written in $\mathcal{ALCSCC}$ as $Human \sqcap succ(|hasChild| \geq 1) \sqsubseteq Parent$. This DL is more expressive than $\mathcal{ALCQ}$ because every quantified restriction of the form $\leq\ n\, r.C$ or $\geq\ n\, r.C$ can be written in $\mathcal{ALCSCC}$ as $succ(|r \cap C| \leq 1)$ or $succ(|r \cap C| \geq 1)$ respectively. However a constraint like $succ(|r| = |s|)$ can not be formulated in $\mathcal{ALCQ}$ [1]. Because of this extension checking satisfiability over $\mathcal{ALCSCC}$ becomes more complicated. Nevertheless in [1] Baader has shown that the satisfiability problem for $\mathcal{ALCSCC}$ is still PSpace-complete.
In this work we present a tableau algorithm for $\mathcal{ALCSCC}$. While a tableau algorithm leads to a runtime complexity worse than PSpace-complete the benefit of a tableau algorithm is that we gain a satisfied interpretation (a correct assignment without contradiction) of the concepts, which is also called *witness*. A tableau algorithm consists of completion rules which are applied to the assertions of the ABox. By applying these rules new assertions that can be derived from the original assertions are added to the ABox. If we can no longer apply any rules and the ABox contains a contradiction, then the ABox is unsatisfiable. Otherwise it is satisfiable. The main difficulty in creating the completion rules for $\mathcal{ALCSCC}$ is that unlike in $\mathcal{ALCQ}$, the number of successor is not bounded. By adding role successors the cardinalities in a constraint can vary. For example if we have a constraint $succ(|r| = |s|)$ then the bound for the number of $s$-successors is equal to the number of $r$-successors we already have. During a

tableau algorithm we can add, merge or replace $r$-successors which changes the bound for the number of $s$-successors. To deal with the changing cardinalities of $\mathcal{ALCSCC}$ we transform the assertions with cardinalities to a QFBAPA formula and use a QFBAPA solver to determine whether the formula is satisfiable or not. If the formula is not satisfiable, there must be a contradiction. If the solver returns a solution, we add new assertions accordingly. Since we use a solver that is capable of returning every possible solution, there can be an infinite number of solutions. We can show that we can shorten each of these solutions to a bound number of role successors without losing any information.

## 1.2 Related Works

The tableau algorithm is a popular tool to solve the satisfiability problem for description logics. Hence a lot of research has been done trying to formulate tableau algorithms for different description logic languages. In [5] Hollunder and Baader present a tableau algorithm for checking the satisfiability of an ABox in the DL $\mathcal{ALCQ}$. This satisfiability problem is PSpace-hard. The presented algorithm consists of five rules which can be applied to the ABox in non-deterministic order. Two of these rules are decomposing rules for $\sqcap$ and $\sqcup$. Furthermore there is a rule that decides whether a successor of an individual name also contributes in any other numerical assertion, which helps to determine the exact number of role successors. The last two rules add and replace individual names to the ABox according to the numerical restrictions. This is can lead to an endless loop of adding and replacing invidual names. To avoid endless loops the author introduces a concept of *safeness*, which has a similar purpose as blocking in other tableau algorithms: Individual names can only be replaced if they fullfill the safeness criteria. In [10] Tobies presented an optimized tableau algorithm for $\mathcal{ALCQ}$, which runs in PSpace. This optimization is achieved by saving an integer which denotes the number of successors already introduced to satisfy a restriction $\geq n\, r.C$, instead of keeping all $n$ possible successors.
In [6] Horrocks et al. published tableau algorithms for $\mathcal{SHIF}$ concepts and $\mathcal{SI}$ concepts. The DL $\mathcal{SI}$ extends the DL $\mathcal{ALC}$ with transitive and inverse role names. The DL $\mathcal{SHIF}$ further extends $\mathcal{SI}$ with role hierarchy and functional restriction. For both DLs the tableau algorithm has to have a blocking technique to avoid infinite chains of introducing elements with the same properties which can be caused by transitive or inverse roles. For $\mathcal{SI}$ the tableau algorithm does not only look at the successors but also the predecessors of the considered individual names when dealing with a $\forall$-assertion. In case of $\exists$-assertions the algorithm first determines whether the considered individual name $x$ is blocked or not. It is blocked if an ancestor (a non-direct predecessor) is blocked or if an ancestor has "similar" assertions as $x$. This algorithm runs in PSpace. For $\mathcal{SHIF}$ the tableau algorithm has to ensure that the considered individual name $x$ is not pair-wise blocked for any rule. Being pair-wise blocked means that for a predecessor $y$ of $x$ there are two ancestors of $x$, such that they behave "similar" to $y$ and $x$. In [3] a DL called $\mathcal{SHQ}$ (also known $\mathcal{ALCQH}_{R^+}$ from the $SI$ family

is presented. This DL does not have inverse roles, but a role hierarchy and numerical restrictions. The difficulty of creating a tableau algorithm for this DL is that with numerical restrictions an infinite chain of adding individual names can prevent the termination of the algorithm. Hence a blocking technique is also needed to ensure termination: An individual name $x$ blocks another individual name $y$ if $x$ was introduced before $y$ and if any assertion about $x$ also holds for $y$. To deal with the number restrictions a reasoner about sets of linear inequations is used.
Regarding the DL $\mathcal{ALCSCC}$, which is considered in this work, Baader provides a solution for the satisfiability problem without a TBox in [1], which is PSpace-complete: For a part of the Abox we guess the values (true or false) of the top-level atoms (concepts). This can already lead to a *false* result, which would mean that the ABox is unsatisfiable. If not, then the ABox is formulated into a QFBAPA formula. Then the formula is extended with constraints over the *Venn regions* of the concepts. For the new formula we test whether it returns true or false with a satisfiability algorithm for QFBAPA. This satisfiability algorithm runs in NP. If the algorithm returns true we are done. If it returns false, we create a concept for every guessed Venn region. Then the algorithm is applied on these new concepts recursively. If it return false, the ABox is unsatisfiable, otherwise satisfiable.

# Chapter 2

# Preliminaries

In order to be able to follow the construction of the tableau algorithm for $\mathcal{ALCSCC}$ this section provides a number of important definitions on QFBAPA and $\mathcal{ALCSCC}$.

## 2.1 QFBAPA

The logic QFBAPA [7] combines boolean algebra (BA) over a sets of symbols with Presburger arithmetic (PA). *Terms* in boolean algebra over a symbol set $T$ are comprised of conjunctions ($\cap$) and/or disjunctions ($\cup$) of symbols. These symbols can also be used in negated form ($s^\neg$, $s \in T$). *Terms* in Presburger arithmetic are additions of natural numbers. In QFBAPA we construct *set terms* using boolean algebra and create *cardinality terms* over the cardinalities of those set terms with the help of Presburger arithmetic. As multiplications can be constructed as chained additions, we also allow multiplication in cardinality terms. QFBAPA allows the construction of inclusion and comparison constrains over set and cardinality terms. This is defined as:

**Definition 1** (QFBAPA)**.** Let $T$ be a finite set of symbols

- set terms over $T$ are:
  - empty set $\emptyset$ and universal set $\mathcal{U}$
  - every set symbol in $T$
  - if $s, t$ are set terms then so are $s \cap t$, $s \cup t$ and $s^\neg$
- set constraints over $T$ are
  - $s \subseteq t$ and $s \not\subseteq t$
  - $s = t$ and $s \neq t$

  where $s, t$ are set terms

- cardinality terms over $T$ are:
  - every number $n \in \mathbb{N}$
  - $|s|$ if $s$ is a set term
  - if $k, l$ are cardinality terms then so are $k + l$ and $n \cdot k$, $n \in \mathbb{N}$
- cardinality constraints over $T$ are:
  - $k = l$ and $k \neq l$
  - $k < l$ and $k \geq l$
  - $k \leq l$ and $k > l$
  - $n\ dvd\ k$ and $n\ \neg dvd\ k$ ($n\ dvd\ k$: $n$ divides $k$)

  where $k, l$ are cardinality terms and $n \in \mathbb{N}$

A QFBAPA formula $\phi$ consists of disjunctions ($\vee$) and/or conjunctions ($\wedge$) of (possible negated) cardinality constraints, where every set symbol is represented as a set variable.
Set constraints of the form $s \subseteq t$ can be expressed as cardinality constraints like $|s \cap t^{\neg}| \leq 0$. Analogously $s \not\subseteq t$ can be expressed as $|s \cap t^{\neg}| > 0$. Set constraints of the form $s = t$ can be written as $|s \cap t^{\neg}| \leq 0$ and $|s^{\neg} \cap t| \leq 0$. Analogously for $s \neq t$. As all set constraints can be written as cardinality constraints we will not use them any further. For better readability we write $k \leq l$ instead of $l \geq k$, $k + 1 \leq l$ instead of $k < l$ and $k \leq l$ and $l \leq k$ instead of $k = l$.
As an example for a QFBAPA formula consider the symbols $T = \{l, a, n, e, f\}$ and the constraints $|l| = 2$, $|l| = |a|$, $|e \cap f^{\neg}| = 0$, $|n \cap f^{\neg}| = 0$. A formula can be written as:

$$|l| = 2 \wedge |l| = |a| \wedge |e \cap f^{\neg}| = 0 \wedge |n \cap f^{\neg}| = 0 \tag{2.1}$$

To satisfy this formula we have to create a semantic that satisfies all cardinality constraints (since all constraints are connected with $\wedge$). In this case we need two elements which are in the semantic of $l$ and therefore also two elements in the semantic of $a$.
The semantics of QFBAPA, called substitutions, are defined as follows:

**Definition 2** (Substitutions of QFBAPA)**.** A substitution $\sigma$ over a symbol set $T$ is a mapping that assigns

- $\mathcal{U}$ to a finite set $\sigma(\mathcal{U})$
- every symbol $a$ in $T$ to $\sigma(a) \subseteq \sigma(\mathcal{U})$
- $\emptyset$ to $\sigma(\emptyset) = \emptyset$
- $\sigma(s \cap t) := \sigma(s) \cap \sigma(t)$, $\sigma(s \cup t) := \sigma(s) \cup \sigma(t)$
- $\sigma(s^{\neg}) := \sigma(\mathcal{U}) \backslash \sigma(s)$

- $\sigma(|s|) := |\sigma(s)|$
- $\sigma(k + l) := \sigma(k) + \sigma(l)$, $\sigma(n \cdot k) := n \cdot \sigma(k)$

Given cardinality terms $k, l$ we say that $\sigma$ satisfies

- $k \leq l$ iff $\sigma(k) \leq \sigma(l)$
- $n \, dvd \, l$ iff $\exists m \in \mathbb{N} : n \cdot m = \sigma(l)$

The substitution $\sigma$ is a solution of a QFBAPA formula $\phi$ if it evaluates the formula to $\top$. A QFBAPA formula is satisfiable if it has a solution $\sigma$ and is unsatisfiable otherwise.
For Equation (2.1) a possible solution $\sigma$ is: Let $\sigma(\mathcal{U}) = \{$leg1, leg 2, arm1, arm 2, nose, ear1, ear2$\}$ and:

- $\sigma(l) = \{$leg1, leg2$\}$
- $\sigma(a) = \{$arm1, arm2$\}$
- $\sigma(n) = \{$nose$\}$
- $\sigma(e) = \{$ear1, ear 2$\}$
- $\sigma(f) = \{$nose, ear1, ear2$\}$

This interpretation satisfies the formula because $\sigma(|l|) = 2 = \sigma(|k|)$, $\sigma(|n \cap f^{\neg}|) = 0$ and $\sigma(|e \cap f^{\neg}|) = 0$.
Now we can interpret our formula as:

- we have 2 legs
- we have as many legs as arms
- nose and two ears are both in the same set hence they belong to a common body part (face)

In [7] Kuncak and Rincard show that checking satisfiability of QFBAPA formulas is a NP-complete problem.

## 2.2 $\mathcal{ALCSCC}$

Next we define the parts and semantics of the description logic $\mathcal{ALCSCC}$ [1]. Let $\mathbf{C}$ be a set of concept names and $\mathbf{R}$ a set of role names, such that $\mathbf{C}$ and $\mathbf{R}$ are disjoint.

**Definition 3** ($\mathcal{ALCSCC}$)**.** $\mathcal{ALCSCC}$ concepts over $\mathbf{C}$ and $\mathbf{R}$ are defined inductively as:

- all concept names in $\mathbf{C}$
- if $C, D$ are $\mathcal{ALCSCC}$ concepts over $\mathbf{C}$ and $\mathbf{R}$ then so are:
  - $\neg C$

  - $C \sqcup D$
  - $C \sqcap D$

- if $c$ is a QFBAPA cardinality constraint over a set $T$ of role names in $\mathbf{R}$ and $\mathcal{ALCSCC}$ concepts over $\mathbf{C}$ and $\mathbf{R}$ then $succ(c)$ is an $\mathcal{ALCSCC}$ concept over $\mathbf{C}$ and $\mathbf{R}$

An $\mathcal{ALCSCC}$ ABox $\mathcal{A}$ is a finite set of assertions of the form $x : C$ and $(x, y) : r$, where $C$ is a $\mathcal{ALCSCC}$ concept, $r \in \mathbf{R}$ and $x, y$ are individual names. The set $I(\mathcal{A})$ is the set of individual names occurring in $\mathcal{A}$.
Regarding Equation (2.1) we can now construct an ABox of specific individual names. Let $\mathbf{C} = \{Legs, Arms, Female\}$ and $\mathbf{R} = \{bodyParts\}$. A possible ABox, which states that an individual name $Anna$ has two legs and two arms and is female, is:

$$\{Anna : succ(|Legs \cap bodyParts| = 2) \sqcap succ(|Legs| = |Arms|) \sqcap Female\} \quad (2.2)$$

Similar to the substitutions for QFBAPA we now define the semantics for $\mathcal{ALCSCC}$ which are called interpretations.

**Definition 4** (Interpretations of $\mathcal{ALCSCC}$)**.** An *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ over an $\mathcal{ALCSCC}$ ABox $\mathcal{A}$ consists of a non-empty set $\Delta^{\mathcal{I}}$ and a mapping $\cdot^{\mathcal{I}}$ which maps:

- each individual name $x \in I(\mathcal{A})$ to $x^{\mathcal{I}} \in \Delta^{\mathcal{I}}$
- each concept name $A \in \mathbf{C}$ to $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
- each role name $r \in \mathbf{R}$ to $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, such that every element in $\Delta^{\mathcal{I}}$ has a finite number of successors.

The set $r^{\mathcal{I}}(x)$ contains all elements $y$ such that $(x, y) \in r^{\mathcal{I}}$ i.e. it contains all $r$-successors of $x$.
For compound concepts the mapping $\cdot^{\mathcal{I}}$ is extended inductively as follows

- $\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$ and $\bot^{\mathcal{I}} = \emptyset^{\mathcal{I}}$
- $(C \sqcap D)^{\mathcal{I}} := C^{\mathcal{I}} \cap D^{\mathcal{I}}$, $(C \sqcup D)^{\mathcal{I}} := C^{\mathcal{I}} \cup D^{\mathcal{I}}$
- $(\neg C)^{\mathcal{I}} := \Delta^{\mathcal{I}} \backslash C^{\mathcal{I}}$
- $succ(c)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} | \text{the mapping } \cdot^{\mathcal{I}_x} \text{ satisfies } c\}$

The mapping $\cdot^{\mathcal{I}_x}$ is a QFBAPA substitution that maps $\emptyset$ to $\emptyset^{\mathcal{I}}$, $\mathcal{U}$ to $\mathcal{U}^{\mathcal{I}_x} := \{\bigcup_{r \in \mathbf{R}} r^{\mathcal{I}}(x)\}$, every concept $C$ occurring in $c$ to $C^{\mathcal{I}_x} := C^{\mathcal{I}} \cap \mathcal{U}^{\mathcal{I}_x}$ and every role name $r$ occurring in $c$ to $r^{\mathcal{I}_x} := r^{\mathcal{I}}(x)$.
$\mathcal{I}$ is a model of $\mathcal{A}$ iff

- $x : C$ iff $x^{\mathcal{I}} \in C^{\mathcal{I}}$
- $(x, y) : r$ iff $(x^{\mathcal{I}}, y^{\mathcal{I}}) \in r^{\mathcal{I}}$

We can define a model $\mathcal{I}$ of the ABox defined in (2.2) by setting $\Delta^{\mathcal{I}} = \{Anna^{\mathcal{I}}, Leg1^{\mathcal{I}}, Leg2^{\mathcal{I}}, Arm1^{\mathcal{I}}, Arm2^{\mathcal{I}}\}$ and

- $Female^{\mathcal{I}} = \{Anna^{\mathcal{I}}\}$
- $Leg^{\mathcal{I}} = \{Leg1^{\mathcal{I}}, Leg2^{\mathcal{I}}\}$
- $Arm^{\mathcal{I}} = \{Arm1^{\mathcal{I}}, Arm2^{\mathcal{I}}\}$
- $bodyPart^{\mathcal{I}} = \{(Anna, Leg1), (Anna, Leg2), (Anna, Arm1), (Anna, Arm2)\}$

By mapping $Anna$ to $Anna^{\mathcal{I}}$, $Leg1$ to $Leg1^{\mathcal{I}}$ and so on we see that this interpretation satisfies the ABox: $Anna^{\mathcal{I}}$ is indeed in $succ(|Legs \cap bodyParts| = 2)^{\mathcal{I}}$ because $Leg1^{\mathcal{I}}, Leg2^{\mathcal{I}} \in bodyPart^{\mathcal{I}_{Anna}} \cap Leg^{\mathcal{I}_{Anna}}$. Analogously for the second $succ$-assertion.
Next we define the *negated normal form* (NNF) for $\mathcal{ALCSCC}$. By transforming all concepts into $NNF$ we avoid nested negations e.g. $\neg(\neg(\neg(A \cup B)))$ which helps to formulate the rules for the tableau algorithm.

**Definition 5** (Negation Normal Form)**.** A $\mathcal{ALCSCC}$ concept is in *negation normal form* ($NNF$) if the negation sign $\neg$ only appears in front of a concept name or above a role name. Let $C$ be an arbitrary $\mathcal{ALCSCC}$ concept. With $NNF(C)$ we denote the concept which is obtained by applying the the following rules on $C$ until none of them are applicable anymore.

| | |
|---|---|
| 1 $\neg\top \to \bot$ | 8 $\neg(k \leq l) \to l \leq k$ |
| 2 $\neg \bot \to \top$ | 9 $\neg(n\ dvd\ k) \to n\ \neg dvd\ k$ |
| 3 $\neg\neg C \to C$ | 10 $\neg(n\ \neg dvd\ k) \to n\ dvd\ k$ |
| 4 $\neg(C \sqcap D) \to \neg C \sqcup \neg D$ | 11 $(s \cap t)^{\neg} \to s^{\neg} \cup t^{\neg}$ |
| 5 $\neg(C \sqcup D) \to \neg C \sqcap \neg D$ | 12 $(s \cup t)^{\neg} \to s^{\neg} \cap t^{\neg}$ |
| 6 $C^{\neg} \to \neg C$ | 13 $(s^{\neg})^{\neg} \to s$ |
| 7 $\neg succ(c) \to succ(\neg c)$ | |

The rule $C^{\neg} \to \neg C$ is necessary because $C^{\neg}$ can be a result of $s^{\neg}$, where $s$ is a set term. $C^{\neg}$ can be transformed into $\neg C$: For every substitution $\sigma$ for a concept $C$ based on QFBAPA it holds that $\sigma(C^{\neg}) = \sigma(\mathcal{U}) \backslash \sigma(C)$ and for every interpretation $\mathcal{I}$ based on $\mathcal{ALCSCC}$ it holds that $(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \backslash C^{\mathcal{I}}$. Since $\sigma(\mathcal{U}) \subseteq \Delta^{\mathcal{I}}$ we can conclude that every element in $\sigma(C^{\neg})$ is also in $(\neg C)^{\mathcal{I}}$.
All rules used to obtain the NNF can be applied in linear time. For rules 1-5 this is shown in [5],[9]. For rules 6-11 this can be shown analogously because we only shift the negation signs. Rules 11, 12 and 13 work similarly to rules 4, 5 and 3 respectively.
Regarding the normal form we additionally replace every disjunction and conjunction in form as $\sqcap$ and $\sqcup$ in every $succ(c)$ concept with $\cap$ and $\cup$. We can do

this because for an arbitrary interpretation $\mathcal{I}$ for each $x, y \in \Delta^{\mathcal{I}}$ it holds that $y \in (C \sqcap D)^{\mathcal{I}_x}$ iff $y \in (C \cap D)^{\mathcal{I}_x}$:

$$\begin{aligned}
& y \in (C \sqcap D)^{\mathcal{I}_x} && \leftrightarrow \\
& y \in (C \sqcap D)^{\mathcal{I}} \cap \mathcal{U}^{\mathcal{I}_x} && \leftrightarrow \\
& y \in C^{\mathcal{I}} \cap D^{\mathcal{I}} \cap \mathcal{U}^{\mathcal{I}_x} && \leftrightarrow \\
& y \in (C^{\mathcal{I}} \cap \mathcal{U}^{\mathcal{I}_x}) \cap (D^{\mathcal{I}} \cap \mathcal{U}^{\mathcal{I}_x}) && \leftrightarrow \\
& y \in C^{\mathcal{I}_x} \cap D^{\mathcal{I}_x} && \leftrightarrow \\
& y \in (C \cap D)^{\mathcal{I}_x}
\end{aligned}$$

Next we define the size of an $\mathcal{ALCSCC}$ concept $C$ inductively over concepts, set terms and cardinality constraints. This definition is necessary for the termination proof.

- $size(r) = size(C) = 1$ if $r \in \mathbf{R}$, $C \in \mathbf{C}$
- $size(n) = size(|k|) = 1$ if $n \in \mathbb{N}$, $k$ cardinality term
- $size(\neg C) = size(C) + 1$
- $size(k^{\neg}) = size(k) + 1$
- $size(C \sqcap D) = size(C \sqcup D) = size(C) + size(D) + 1$
- $size(k \cap l) = size(k \cup l) = size(k) + size(l) + 1$
- $size(|k|) = size(k)$
- $size(succ(c)) = \begin{cases} 1 + size(k) + size(l) & c = k \leq l \\ 1 + size(l) & c = n \, dvd \, l \end{cases}$

# Chapter 3

# Tableau for $\mathcal{ALCSCC}$

The tableau algorithm is a popular tool to check satisfiability. Even though the complexity of tableau algorithms can grow exponentially one advantage of them is that they do not only check if an Abox is satisfiable but also return an interpretation that satisfies this ABox (*witness*) if one exists. A tableau algorithm consists of completion rules that are used to iteratively add new assertions to the ABox that are derived from pre-existing assertions. These rules are exhaustively applied to the Abox until there are no more applicable rules. For some completion rules like the rule for disjunctions ($x : C \sqcap D$), the algorithm can decide which assertion is added to the ABox (either $x : C$ or $x : D$). If such a choice results in a *clash* the algorithm back tracks to the point of the decision and tries an alternative choice instead. If all choices end in a clash, then the ABox is unsatisfiable.
Before *clashes* can be defined we first need to introduce the concept of *induced interpretations*. Induced interpretations can be used to count the number of successors of any individual name after any rule application and hereby detect *violated assertions*.

**Definition 6** (Induced Interpretation)**.** An interpretation $\mathcal{I}(\mathcal{A})$ can be induced from an ABox $\mathcal{A}$ through the following steps:

- for each individual name $x \in I(\mathcal{A})$ we introduce $x^{\mathcal{I}(\mathcal{A})}$ and add it to $\Delta^{\mathcal{I}(\mathcal{A})}$
- for each $x : C$ such that $C$ is a concept name we add $x^{\mathcal{I}(\mathcal{A})}$ to $C^{\mathcal{I}(\mathcal{A})}$
- for each $(x, y) : r$ such that $r$ is a role name we add $(x^{\mathcal{I}(\mathcal{A})}, y^{\mathcal{I}(\mathcal{A})})$ to $r^{\mathcal{I}(\mathcal{A})}$

**Definition 7** (Violated assertion)**.** Let $\mathcal{A}$ be an ABox and $x$ be an individual name in $I(\mathcal{A})$. An assertion $x : succ(c)$ is *violated* if $x^{\mathcal{I}(\mathcal{A})} \notin succ(c)^{\mathcal{I}(\mathcal{A})}$.

Violated assertions can sometimes be resolved by applying further completion rules. However if we find a violated assertion and can no longer apply any rules there is a clash. Aside from unresolvable violated assertions there are other kinds of situations that are also labelled as clashes.

**Definition 8** (Clash)**.** An ABox $\mathcal{A}$ contains a *clash* if

- $\{x : \bot\} \subseteq \mathcal{A}$ or
- $\{x : C, x : \neg C\} \subseteq \mathcal{A}$ or
- $\{(x, y) : r, (x, y) : \neg r\} \subseteq \mathcal{A}$ or
- $x : succ(c) \in \mathcal{A}$ violated and no more rules are applicable

## 3.1 Transforming an ABox into a formula

One major difficulty of creating a tableau algorithm for $\mathcal{ALCSCC}$ is its numerical arithmetic in the form of successor-assertions. The application of rules can change the number of successors which can in turn influence the number of successors that are demanded by certain constraints. Furthermore it introduces the problem of *nested* successor assertions.

**Definition 9** (Nested Level)**.** Let $\mathcal{A} = \{x : C\}$ be an ABox. An individual name lays in the $i$-th nested level if it is the $i$-th individual name in a role chain beginning from $x$, where the individual name $x$ is in the 0th nested level. A direct successor of $x$ is in the 1st nested level.

In some DLs we are able to describe the successor of a successor like $\exists r.(\exists r.C)$, such that the number of needed successors is fixed. However in $\mathcal{ALCSCC}$ such boundaries can vary like in $x : succ(|succ(|A| < |B|)| > |A|)$. The number of successors needed to satisfy $succ(|A| < |B|)$ depends on how many successors $x$ already has in $A$. By applying rules the number of successors for $x$ in $A$ can change. Hence we use a QFBAPA solver whenever we want to add successors for an individual name $x$. To do this we first collect all *succ*-assertions regarding $x$ and then transform them into a QFBAPA formula for the next nested level, which means we only consider the direct successors of $x$. We assume that the ABox is already in $NNF$. To create an example for a transformation we consider the following ABox:

**Example 1.**

$$\mathcal{A} = \{x : succ(1 \leq |succ(|A| \leq |B \cap r|)|), x : succ(|A| \leq |B|), x : C\}$$

with $\mathbf{C} = \{A, B, C\}$ and $\mathbf{R} = \{r\}$

In this example the first assertion states that $x$ must have at least one successor $y$ which in turn has at least as many successors in $B \cap r$ as in $A$. The individual names for $succ(|A| \leq |B \cap r|)$ are on a different nested level than the ones for $succ(|A| \leq |B|)$. The second assertion states that $x$ has at least as many successors in $B$ as in $A$.
We start by gathering all *succ*-assertion regarding $x$ and then transform the cardinality constraints into a formula by carrying out the following steps:

- replace all role names $r$ with $X_r$
- replace all concepts names $C$ with $X_C$
- replace all $succ(c)$ with $X_{succ(c)}$
- connect all formulas with $\wedge$
- include the conjunct $\mathcal{U} = X_{r_1} \cup \cdots \cup X_{r_n}$, $r_1, \ldots, r_n \in \mathbf{R}$

We replace (compound) concepts and role names with set variables, so a solver can assign elements to them. The last step is important because sometimes it is not explicitly stated what kind of successor an individual name has. However a successor must always be "connected" to its predecessor by at least one role name. Through the last step we ensure that every element (successor) is assigned to a set variable, which represents a role name.
In our example we have five set variables: $X_A$, $X_B$, $X_r$, $X_{succ(|A| \leq |B \cap r|)}$ and $\mathcal{U}$. The QFBAPA formula for Example 1 is:

$$\phi = 1 \leq |X_{succ(|A| \leq |B \cap r|)}| \wedge |X_A| \leq |X_B| \wedge \mathcal{U} = X_r \tag{3.1}$$

We can use the solver to get a possible solution, if there is one. We make two assumptions about the QFBAPA solver so we can use it in the tableau algorithm.

**Assumption 1.** We assume that every considered QFABAPA solver is correct which means

- it terminates for all QFBAPA formulas
- a formula is satisfiable iff it returns a solution

**Assumption 2.** Let $\phi$ be an arbitrary QFBAPA formula. We assume that every considered QFABAPA solver is able to return all possible solutions of $\phi$.

## 3.2 Solution of a formula

By Assumption 2 the QFBAPA solver can return infinitely many solutions for some formulas. For instance the solver can find infinitely many solutions for Example 1: Increasing the number of successors in $B$ will always yield new valid solutions as long as the number of successors in $A$ is kept lower. However this means that the tableau algorithm is sometimes working on an infinite solution space and hence might not terminate. Therefore we limit the considered solutions to some pre-computed *upper bound*. For *Integer Linear Programming* (ILP) problems, that can be described as systems of linear equalities, there are already known ways to compute upper bounds as is shown in [8]. Thus we want to transform our formulas into a linear system of equalities of the form $Mx = b$, where $M$ and $b$ describe our cardinality constraints and $x$ is the solution i.e. denotes the numbers of elements we have to assign to set variables to satisfy the

formula.

First, we notice that every inequality in a QFBAPA formula can be rewritten as $n_1 \cdot |X_1| \pm \cdots \pm n_i \cdot |X_i| \lesseqgtr I$, $\lesseqgtr \in \{\leq, \geq, =\}$, where $n_1, \ldots, n_i, I \in \mathbb{Z}$ are a constants. The numbers $n_1, \ldots, n_i$ are called *pre-factors*. Let $c = succ(|A| \leq |B \cap r|)$. We can rearrange $\phi$ of Example 1 (3.1) as:

$$\phi' = |X_c| \geq 1 \wedge |X_A| - |X_B| \leq 0 \wedge |\mathcal{U} \cap X_r^{\neg}| = 0 \wedge |\mathcal{U}^{\neg} \cap X_r| = 0 \tag{3.2}$$

Next we transform the two inequalities into equalities by adding slack variables $I_1$ and $I_2$:

$$\begin{aligned} \phi'' = &|X_c| - I_1 = 1 \wedge |X_A| - |X_B| + I_2 = 0 \wedge \\ &|\mathcal{U} \cap X_r^{\neg}| = 0 \wedge |\mathcal{U}^{\neg} \cap X_r| = 0 \end{aligned} \tag{3.3}$$

At the moment it is not clear whether the set variables are overlapping which would be problematic for a system of linear equations because its variables must be disjunct. To ensure disjunct variables we consider *Venn regions*, which are of the form $X_1^i \cap \cdots \cap X_k^i$. The subscript $i$ is either 0 or 1. $X_1^0$ denotes $X_1^{\neg}$ and $X_1^1$ denotes $X_1$. As we have 5 set variables, there are $2^5 = 32$ Venn regions. The number of Venn regions grows exponentially with the number of set variables. In [1] it is stated that there exists a number $N$, which is polynomial in the size of $\phi$, such that at most $N$ Venn regions are not empty if there exists a solution.

**Lemma 1** (Lemma 3 from [1])**.** For every QFBAPA formula $\phi$ there is a number $N$, which is polynomial in the size of $\phi$ and can be computed in polynomial time such that for every solution $\sigma$ of $\phi$ there exists a solution $\sigma'$ of $\phi$ such that:

- $|\{v | v \text{ is a Venn region and } \sigma'(v) \neq \emptyset\}| \leq N$
- $\{v | v \text{ is a Venn region and } \sigma'(v) \neq \emptyset\} \subseteq \{v | v \text{ is a Venn region and } \sigma(v) \neq \emptyset\}$

We can guess a number $N$ of Venn regions, which are non-empty (in nondeterministic polynomial time). For Example 1 we know that any Venn region within $X_r^{\neg}$ or $\mathcal{U}^{\neg}$ must be empty, because every element must be in $\mathcal{U}$ and since $\mathcal{U} = X_r$ they must all be in $X_r$. Therefore we can drop 24 Venn regions. We construct $M$ and $b$ such that instead of assigning elements to set variables we assign them to the remaining 8 Venn regions. That means that for the vector $x$ the entry $x_k$, $1 \leq k \leq 8$, denotes the number of elements in the $k$-th Venn region. As there are four equations and two slack variables the matrix $M$ has four rows and ten columns with $m_{ij}$, $1 \leq i \leq 4$ and $1 \leq j \leq 10$, denoting the sum of pre-factors of the set variables, in which the $j$-th Venn region is included, occurring in the $i$-th equation. Let the column vectors describing the Venn regions be in the following order:

- $v_1 = X_A \cap X_B \cap X_c \cap X_r \cap \mathcal{U} \cap I_1^{\neg} \cap I_2^{\neg}$
- $v_2 = X_A \cap X_B \cap X_c^{\neg} \cap X_r \cap \mathcal{U} \cap I_1^{\neg} \cap I_2^{\neg}$

- $v_3 = X_A \cap X_B^{\neg} \cap X_c \cap X_r \cap \mathcal{U} \cap I_1^{\neg} \cap I_2^{\neg}$
- $v_4 = X_A \cap X_B^{\neg} \cap X_c^{\neg} \cap X_r \cap \mathcal{U} \cap I_1^{\neg} \cap I_2^{\neg}$
- $v_5 = X_A^{\neg} \cap X_B \cap X_c \cap X_r \cap \mathcal{U} \cap I_1^{\neg} \cap I_2^{\neg}$
- $v_6 = X_A^{\neg} \cap X_B \cap X_c^{\neg} \cap X_r \cap \mathcal{U} \cap I_1^{\neg} \cap I_2^{\neg}$
- $v_7 = X_A^{\neg} \cap X_B^{\neg} \cap X_c \cap X_r \cap \mathcal{U} \cap I_1^{\neg} \cap I_2^{\neg}$
- $v_8 = X_A^{\neg} \cap X_B^{\neg} \cap X_c^{\neg} \cap X_r \cap \mathcal{U} \cap I_1^{\neg} \cap I_2^{\neg}$

The last two vectors describe the slack variables:

- $v_9 = X_A^{\neg} \cap X_B^{\neg} \cap X_c^{\neg} \cap X_r^{\neg} \cap \mathcal{U}^{\neg} \cap I_1 \cap I_2^{\neg}$
- $v_{10} = X_A^{\neg} \cap X_B^{\neg} \cap X_c^{\neg} \cap X_r^{\neg} \cap \mathcal{U}^{\neg} \cap I_1^{\neg} \cap I_2$

We now create the linear system of equations:

$$\begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & -1 & -1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \end{pmatrix}$$

Note that $a_{2,1}$ and $a_{2,2}$ are 0 because the pre-factors of $|X_A|$ and $|X_B|$ in the second equation are 1 and $-1$ and the Venn regions $v_1$ and $v_2$ are both included in $X_A$ and $X_B$. If $x_i = 0$, then the $i$-th Venn region is empty. The last two rows of $M$, which represent the equations $|\mathcal{U} \cap X_r^{\neg}| = 0$ and $|\mathcal{U}^{\neg} \cap X_r| = 0$, are lines of zeros because we omit the Venn regions, in which $\mathcal{U} \cap X_r^{\neg}$ and $\mathcal{U}^{\neg} \cap X_r$ are included.
Now that we have created a linear system of equations we are able to calculate an upper bound for the number of elements within the Venn regions. The following theorem from [8] can be used to establish an upper bound for the solution of the *ILP* problem in NP:

**Theorem 1** (Theorem 1 from [8])**.** Let $A \in \mathbb{Z}^m \times \mathbb{Z}^n$ be a matrix and $b \in \mathbb{Z}^m$ a vector. If $x \in \mathbb{N}^n$ is a solution of $Ax = b$, then there exists a solution $x'$ such that all entries are integers between 0 and $n \cdot (m \cdot max_{i,j}\{|a_{ij}|, |b_i|\})^{2 \cdot m+1}$.

We take a look now in the proof of this theorem to understand how the solution is decreased. We distinguish between two cases. Let $M = m \cdot max_{i,j}\{|a_{ij}|\}^m$, $F = \{i | x_i > M\}$ and $v_i$ be the $i$-th column of $A$

- If there exist integers $\alpha_i$, for all $i \in F$, such that $\sum_{i \in F} \alpha_i \cdot v_i = 0$ and $\exists i : \alpha_i > 0$ then $x' = x - d$, $d_j = \alpha_j$ if $j \in M$ else $d_j = 0$, $1 \leq j \leq n$.
- Else: There must be a vector $h \in \{0, \pm 1, \pm 2, \cdots \pm M\}^m$ such that $h^T v_i \geq 1, i \in F$. We premultiply $A$ and $b$ with $h^T$ and show that $x$ is already in the bound:

$$h^T Ax = h^T b$$

Therefore we are able to calculate an upper bound for the number of elements in each Venn region of the solutions returned by the QFBAPA solver a priori, which is important for the termination proof.
In Example 1 the upper bound for all $x_i$ is $10 \cdot (4 \cdot max\{|1|, |-1|\})^{2 \cdot 4+1} = 2621440$, which means that we can discard any solution in which a Venn region has more than 2621440 elements.

## 3.3 The Tableau Algorithm

Now that we have described how to deal with the numerical challenges of $\mathcal{ALCSCC}$ we construct a tableau algorithm for an ABox in $\mathcal{ALCSCC}$. The algorithm can be divided into two repeating steps. The first step decomposes disjunctions and conjunctions of concepts into basic concepts. This step considers disjunctions before conjunctions. The second step transforms successor assertions of a selected individual name into a QFBAPA formula. In the second step an upper bound for the number of elements in each Venn region is calculated. Then a solver is used to find a possible solution within this bound, if the ABox is satisfiable. The second step can only be applied if all conjunctions and disjunctions have already been decomposed by the first step.

**Definition 10** (Tableau for $\mathcal{ALCSCC}$)**.** The completion rules for an $\mathcal{ALCSCC}$ ABox $\mathcal{A}$ in $NNF$ are as follows.
First step:

- $\sqcap$-rule: $\mathcal{A}$ contains $x : C_1 \sqcap C_2$ but not both $x : C_1$ and $x : C_2$
  $\rightarrow \mathcal{A} := \mathcal{A} \cup \{x : C_1, x : C_2\}$
- $\sqcup$-rule: $\mathcal{A}$ contains $x : C_1 \sqcup C_2$ but neither $x : C_1$ nor $x : C_2$
  $\rightarrow \mathcal{A} := \mathcal{A} \cup \{x : C_1\}$ or $\mathcal{A} := \mathcal{A} \cup \{x : C_2\}$

Second step:

- *successor*-rule: $\mathcal{A}$ contains an individual name $x$ which has at least one violated assertion of the form $x : succ(c)$, this rule has not been applied for $x$ yet and no rules from step 1 are applicable:
  - gather all assertions of the form $x : succ(c)$ into a set $\mathcal{S}$
  - transform $\mathcal{S}$ into a QFBAPA formula $\phi$ as in Section 3.1

– calculate the upper bound as in Theorem 1

If the QFBAPA solver returns *unsatisfiable*, then $\mathcal{A} := \mathcal{A} \cup \{x : \bot\}$
If the QFBAPA solver returns *satisfiable*, then select one solution $\sigma$ within the upper bound. For each $e \in \sigma(\mathcal{U})$, we introduce a new individual name $y_e$ and

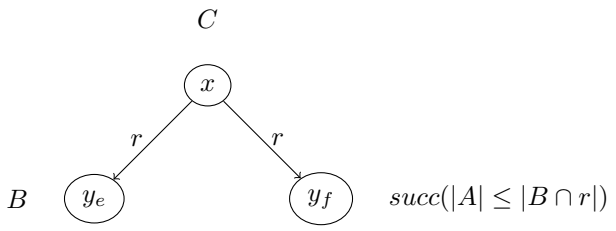
– if $e \in X_C$ we set $\mathcal{A} := \mathcal{A} \cup \{y_e : C\}$
– if $e \in X_{succ(c)}$ we set $\mathcal{A} = \mathcal{A} \cup \{y_e : succ(c)\}$
– if $e \in X_r$, $r \in \mathbf{R}$, we set $\mathcal{A} := \mathcal{A} \cup \{(x, y_e) : r\}$
– if $e \notin X_C$ we set $\mathcal{A} := \mathcal{A} \cup \{y_e : \neg C\}$
– if $e \notin X_{succ(c)}$ we set $\mathcal{A} := \mathcal{A} \cup \{y_e : NNF(\neg succ(c))\}$
– if $e \notin X_r$, $r \in \mathbf{R}$, we have $\mathcal{A} := \mathcal{A} \cup \{(x, y_e) : \neg r\}$

A *complete* ABox is an ABox for which no more rules of the tableau algorithm are applicable.
Next an example run of this algorithm over the following ABox is given:

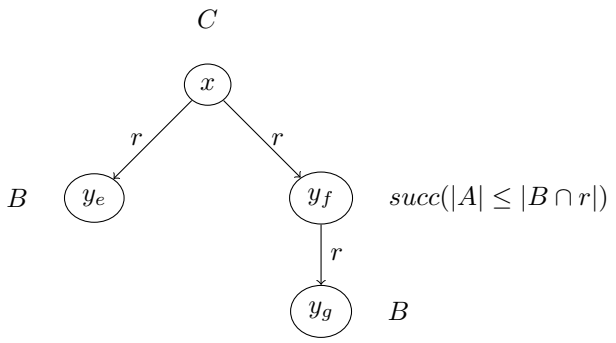
**Example 2.**

$$\mathcal{A} = \{x : succ(1 \leq |succ(|A| \leq |B \cap r|)|) \sqcap succ(|A| \leq |B|) \sqcap C\}$$

Initially the $\sqcap$-rule is applicable hence it is applied first which results in an ABox identical to the one given in Example 1. Now neither the $\sqcap$- nor the $\sqcup$-rule is applicable. Therefore the *successor*-rule as applied: Every *succ*-assertion regarding $x$ is gathered in a set $\mathcal{S} := \{x : succ(1 \leq |succ(|A| \leq |B \cap r|)|), x : succ(|A| \leq |B|)\}$ and then converted to a QFBAPA formula which is identical to the one presented in Equation (3.1). The upper bound for the number of elements in this formula is 2621440 (see Section 3.2). If the formula had been unsatisfiable, $x : \bot$ would have been added to the ABox. This would result in the termination of the tableau algorithm as there would be no more applicable rules and no point to backtrack to. However as the formula is satisfiable the solver must return a solution, because of Assumption 1. We see that the formula is satisfiable with $X_{succ(|A| \leq |B \cap r|)} = \{f\}, X_A = \{\}, X_B = \{e\}$ and $X_r = \{e, f\}$. Because of the constraint $\mathcal{U} = \{X_r\}$ contained in the formula every element must be in $X_r$ i.e. every successor is an $r$-successor. The QFBAPA solver is capable of returning this solution because of Assumption 2. Obviously in this solution every Venn-region has less than 2621440 elements. Next two individual names $y_e$ and $y_f$ are introduced for the elements $e$ and $f$ and the assertions $y_e : X_B$, $(x, y_e) : r$, $y_f : succ(|A| \leq |B \cap r|)$ and $(x, y_f) : r$ are added to $\mathcal{A}$. For the next two graphics we omitted the information, that $x$ is of the concepts $succ(1 \leq |succ(|A| \leq |B \cap r|)|) \sqcap succ(|A| \leq |B|) \sqcap C, succ(1 \leq |succ(|A| \leq |B \cap r|)|)$ and $succ(|A| \leq |B|)$ due to readability:




Then the *successor*-rule is applied to $y_f$ because no rules of step 1 are applicable. Since $y_f : succ(|A| \leq |B \cap r|)$ is the only *succ*-assertion it is sufficient to introduce an $r$-successor of the concept $B$. We assume that the QFBAPA solver returns the solution $X_B = X_r = \{g\}$. Therefore the algorithm introduces an individual name $y_g$ and adds $y_g : B$ and $(y_f, y_g) : r$ to $\mathcal{A}$.




The algorithm only adds assertions for individual names, which are currently considered in the first step or which are introduced in the second step. Hence if all possible rules are applied to the individual name $x$ and there is no clash, then all assertions of $x$ must remain satisfied until the tableau algorithm terminates. In the next chapter we formally prove the correctness of this algorithm.

# Chapter 4

# Correctness

To prove the correctness of the tableau algorithm we have to show the following statements:

- The tableau algorithm terminates for every input.
- If no more rules are applicable on a clash-free ABox $\mathcal{A}$, then $\mathcal{A}$ is satisfiable.
- If $\mathcal{A}$ is satisfiable, then the tableau algorithm terminates without a clash.

For all proofs we make use of Assumptions 1 and 2. First we prove that the algorithm always terminates.

## 4.1 Termination

An ABox is called a *derived* ABox if it is the result of a finite number of rule applications on an ABox.

We can show that each of the completion rules must terminate:

- $\sqcap$- and $\sqcup$-rule: Both rules terminate because they decompose finite compound concepts.
- *successor*-rule: Since ABoxes are finite they can only have a finite subset $\mathcal{S}$ of *successor*-assertions and therefore the algorithm can always create a (finite) QFBAPA formula. By Assumption 1 the QFBAPA solver always terminates and returns a finite solution. Therefore only a finite number of successors are added during the application of this rule. Hence this rule application always terminates.

To prove termination we map every ABox $\mathcal{A}$ to an element $\Psi(\mathcal{A})$ of a set $Q$. Each $\Psi(\mathcal{A})$ consists of triples $\psi_{\mathcal{A}}(x)$ for each individual name $x$. Let $\prec$ be an strict partial ordering, which is an irreflexive (if $a \prec b$ then $b \not\prec a$) and transitive (if $a \prec b$ and $b \prec c$ then $a \prec c$) relation (with $a, b, c$ being comparable elements).

If we can show that $\prec$ is well-founded (e.g. there is no infinite decreasing chain) and that for every ABox $\mathcal{A}'$, which is derivable from an ABox $\mathcal{A}$, it must hold that $\Psi(\mathcal{A}') \prec \Psi(\mathcal{A})$ and $\Psi(\mathcal{A}) \not\prec \Psi(\mathcal{A}')$, then the termination of the algorithm can be concluded.
Each triple in $Q$ consists of a multiset of natural numbers and two natural numbers. A multiset of natural numbers $M'$ is smaller than another multiset of natural numbers $M$ if we can obtain $M'$ from $M$ by replacing at least one number $n$ of $M$ with natural numbers, which are all smaller than $n$, or by deleting at least one number of $M$. For example $M' = \{2, 2, 2, 1, 5\}$ is smaller than $M = \{2, 3, 5\}$ because it can be obtained by replacing the second entry 3 of $M$ with $2, 2, 1$. We say that the empty multiset $\{\}$ is always smaller than any multiset of natural numbers. Since the multisets consist of natural numbers, which can be ordered by the strict partial order $<$, they can also be ordered by $\prec$. A triple $T_1 = (x_1, x_2, x_3)$ is smaller than a triple $T_2 = (y_1, y_2, y_3)$ if $T_1$ is lexicographically smaller (from left to right) than $T_2$ which means that for the first $i \in \{1, 2, 3\}$ where $x_i \neq y_i$ it holds that $x_i$ is smaller than $y_i$. Because we have triples of numbers they can also be ordered by $\prec$. Therefore $\Psi(\mathcal{A}') \prec \Psi(\mathcal{A})$ if we can replace at least one triple $\psi_{\mathcal{A}}(x)$ in $\Psi(\mathcal{A})$ with at least one triple $\psi_{\mathcal{A}'}(x')$, such that $\psi_{\mathcal{A}'}(x') \prec \psi_{\mathcal{A}}(x)$ or if we can remove at least one triple from $\Psi(\mathcal{A})$ in order to obtain $\Psi(\mathcal{A}')$.
Note that in Section 2.2 we require that each instance of $C \sqcap D$ and $C \sqcup D$ in a cardinality term is replaced by $C \cap D$ and $C \cup D$ respectively.
Now we describe what the triples in $Q$ look like.

**Definition 11.** Let $\mathcal{A}$ be a derived ABox. The multiset $\Psi(\mathcal{A})$ consists of triples. Each triple $\psi_{\mathcal{A}}(x)$ represents one individual name $x$:

- The first component $\psi_{\mathcal{A},1}(x)$ is the natural number $max\{size(C) | x : C \in \mathcal{A}\}$.
- The second component $\psi_{\mathcal{A},2}(x)$ is a multiset which contains the natural numbers $size(C \sqcap D)$ for each assertion $x : C \sqcap D \in \mathcal{A}$ for which the $\sqcap$-rule is applicable. Respectively for $x : C \sqcup D$.
- The third component $\psi_{\mathcal{A},3}(x)$ is the number 1 if the *successor*-rule is applicable and 0 otherwise.

For the ABox $\mathcal{A}$ in Example 2 we have the following multiset:

$$\Psi(\mathcal{A}) = \{\psi_{\mathcal{A}}(x)\} = \{(7, \{7, 7\}, 0)\} \tag{4.1}$$

After decomposing we get the ABox $\mathcal{A}^1$, which is used in Example 1.

$$\Psi(\mathcal{A}^1) = \{\psi_{\mathcal{A}^1}(x)\} = \{(7, \{\}, 2)\} \tag{4.2}$$

We can see that $\psi_{\mathcal{A}',2}(x) \prec \psi_{\mathcal{A},2}(x)$ because $\mathcal{A}^1$ does not contain any conjunctions on which the $\sqcap$-rule is applicable anymore. Therefore it does not matter that the third entry of the triple $\psi_{\mathcal{A}}(x)$ has grown. Hence $\psi_{\mathcal{A}'}(x) \prec \psi_{\mathcal{A}}(x)$

which means $\Psi(\mathcal{A}') \prec \Psi(\mathcal{A})$.
Next the *successor*-rule is applied and we add two new individual names $y_e$ and $y_f$ to obtain $\mathcal{A}^2$. Therefore two new triples have to be added:

$$\Psi(\mathcal{A}^2) = \{\psi_{\mathcal{A}^2}(x), \psi_{\mathcal{A}^2}(y_e), \psi_{\mathcal{A}^2}(y_f)\} = \{(7, \{\}, 0), (1, \{\}, 0), (3, \{\}, 1)\} \quad (4.3)$$

We see that $\psi_{\mathcal{A}^2}(x)$, $\psi_{\mathcal{A}^2}(y_e)$ and $\psi_{\mathcal{A}^2}(y_f)$ are smaller than $\psi_{\mathcal{A}^1}(x)$ so it holds that $\Psi(\mathcal{A}^2) \prec \Psi(\mathcal{A}^1)$.
We then apply the *successor*-rule on last time to $y_f$ which results in the multiset:

$$\begin{aligned}\Psi(\mathcal{A}^3) = \{&\psi_{\mathcal{A}^3}(x), \psi_{\mathcal{A}^3}(y_e), \psi_{\mathcal{A}^3}(y_f), \psi_{\mathcal{A}^3}(y_g)\} = \\ &\{(7, \{\}, 0), (1, \{\}, 0), (3, \{\}, 0), (2, \{\}, 0)\}\end{aligned} \quad (4.4)$$

The newly introduced triple $\psi_{\mathcal{A}^3}(y_g)$ is smaller then $\psi_{\mathcal{A}^2}(y_f)$, because $\psi_{\mathcal{A}^3,1}(y_g) < \psi_{\mathcal{A}^2,1}(y_f)$, and therefore $\Psi(\mathcal{A}^3) \prec \Psi(\mathcal{A}^2) \prec \Psi(\mathcal{A}^1) \prec \Psi(\mathcal{A})$.
We finally prove the termination of the algorithm.

**Lemma 2.** For any ABox $\mathcal{A} = \{x : C\}$ the tableau algorithm terminates.

*Proof.* We show that if $\mathcal{A}'$ is derivable from $\mathcal{A}$ by applying a rule from Definition 3.3, then it holds that: $\Psi(\mathcal{A}') \prec \Psi(\mathcal{A})$.

- $\mathcal{A}'$ is obtained from $\mathcal{A}$ by applying the $\sqcap$-rule on $x : C \sqcap D$: The first component remains unchanged because $size(C) \leq size(C \sqcap D)$ and $size(D) \leq size(C \sqcap D)$. It holds that $\psi_{\mathcal{A}',2}(x) \prec \psi_{\mathcal{A},2}(x)$ because the number $size(C \sqcap D)$ is removed from $\psi_{\mathcal{A},2}(x)$ (as we can only apply the $\sqcap$-rule once on $x : C \sqcap D$). In case $C$ and/or $D$ happen to be disjunctions or conjunctions $\psi_{\mathcal{A}',2}(x)$ still becomes smaller because $size(C) < size(C) + size(D) + 1 = size(C \sqcap D)$ (respectively for $size(D)$). Hence $\psi_{\mathcal{A}'}(x) \prec \psi_{\mathcal{A}}(x)$.
  For any other individual name $y$, where $y \neq x$, the triple $\psi_{\mathcal{A}}(y)$ remains unchanged.

- $\mathcal{A}'$ is obtained from $\mathcal{A}$ by applying the $\sqcup$-rule on $x : C \sqcup D$: analogously as for the $\sqcap$-rule.

- $\mathcal{A}'$ is obtained from $\mathcal{A}$ by applying the *successor*-rule on $x : succ(c)$: $\psi_{\mathcal{A},1}(x)$ remains unchanged because no assertions are added for $x$. This rule is applicable, as both the $\sqcap$-rule and $\sqcup$-rule are not applicable on $\mathcal{A}$ and we have not applied this rule for $x$ yet. Because neither the $\sqcap$-rule nor the $\sqcup$-rule is applied, $\psi_{\mathcal{A},2}(x)$ remains unchanged. It also holds that $\psi_{\mathcal{A},3}(x) = 1$ because the *successor*-rule can be applied on an assertion of $x$. After applying the *successor*-rule it holds that $\psi_{\mathcal{A}',3}(x) = 0$. Therefore $\psi_{\mathcal{A}'}(x) \prec \psi_{\mathcal{A}}(x)$.
  For every freshly introduced individual name $y$ we have to add a triple $\psi_{\mathcal{A}'}(y)$ to $\Psi(\mathcal{A}')$. For every $y : C \in \mathcal{A}'$ we know that $C$ must be part of a cardinality constraint $c$ such that $x : succ(c) \in \mathcal{A}$ and therefore $size(succ(c)) > size(C)$. That means that $max\{size(C) | y : C \in \mathcal{A}'\}$ is

always smaller then $max\{size(C)|x : C \in \mathcal{A}'\}$ by the definition of $size(C)$. Hence $\psi_{\mathcal{A}',1}(y) < \psi_{\mathcal{A}',1}(x)$ and therefore $\psi_{\mathcal{A}'}(y) \prec \psi_{\mathcal{A}'}(x)$.
For any other individual name $z$, where $z \neq x$ and $z = y$, the triple $\psi_{\mathcal{A}}(y)$ remains unchanged.

Hence in all three cases $\Psi(\mathcal{A}')$ can be obtained from $\Psi(\mathcal{A})$ by replacing $\psi_{\mathcal{A}}(x)$ with the smaller triple $\psi_{\mathcal{A}'}(x)$. For any newly introduced individual names we showed that the new triples must be smaller than $\psi_{\mathcal{A}'}(x)$. Therefore $\Psi(\mathcal{A}') \prec \Psi(\mathcal{A})$.
Because we work with natural numbers the ordering $<$ over them is well-founded. Therefore we also know that $\prec$ for multisets over natural numbers is also well-founded [2](Theorem 2.5.5). Since the natural numbers and multisets of natural numbers can be ordered by a well-founded ordering, the triples which are constructed after Definition 11, which are "lexicographical products of two terminating relations"[2](Theorem 2.4.2), can be ordered by a well-founded ordering as well. Therefore $\prec$ over the multisets of these triples is well-founded, too [2](Theorem 2.5.5). □

## 4.2 Soundness and Completeness

Next we prove the correctness of the algorithm e.g. that the algorithm terminates with a clash-free ABox iff the ABox is satisfiable.

**Lemma 3** (Soundness)**.** If the tableau algorithm is applied on an ABox $\mathcal{A} = \{x : C\}$ and terminates without a clash, then $\mathcal{A}$ is satisfiable.

*Proof.* Let $\mathcal{A}'$ be the returned ABox after the algorithm terminated. By Lemma 2 the tableau algorithm must always terminate. Since no assertions are removed during the algorithm it must hold that $\mathcal{A} \subseteq \mathcal{A}'$. Hence if an interpretation $\mathcal{I}$ satisfies $\mathcal{A}'$ then it also satisfies $\mathcal{A}$. Let $\mathcal{I}(\mathcal{A}')$ be the induced interpretation of $\mathcal{A}'$. We show that $\mathcal{I}(\mathcal{A}')$ indeed satisfies $\mathcal{A}'$ by induction over the concepts:
For each concept name $C \in \mathbf{C}$ where $x : C \in \mathcal{A}'$, it holds that $x^{\mathcal{I}(\mathcal{A}')} \in C^{\mathcal{I}(\mathcal{A}')}$ by the definition of induced interpretations (induction base).
We consider $x : C$ where $C$ is a compound concept (induction step):

- $C = \neg D$: Since $\mathcal{A}'$ does not contain a clash, $x : C \in A$ implies $x : D \notin A$. $D$ must be a concept name, because $\mathcal{A}'$ is in $NNF$. Therefore by the definition of induced interpretations and $x : D \notin A$ it holds that $x^{\mathcal{I}(\mathcal{A}')} \notin D^{\mathcal{I}(\mathcal{A}')}$ which implies $x^{\mathcal{I}(\mathcal{A}')} \in \Delta^{\mathcal{I}(\mathcal{A}')} \backslash D^{\mathcal{I}(\mathcal{A}')} = C^{\mathcal{I}(\mathcal{A}')}$.

- $C = D \sqcap E$: After the algorithm terminates, the $\sqcap$-rule is no longer applicable. That means that there is an individual name $x$, such that $\{x : D, x : E\} \subseteq \mathcal{A}'$. By the induction hypothesis it holds that $x^{\mathcal{I}(\mathcal{A}')} \in D^{\mathcal{I}(\mathcal{A}')}$ and $x^{\mathcal{I}(\mathcal{A}')} \in E^{\mathcal{I}(\mathcal{A}')}$. Therefore $x^{\mathcal{I}(\mathcal{A}')} \in D^{\mathcal{I}(\mathcal{A}')} \cap E^{\mathcal{I}(\mathcal{A}')} = C^{\mathcal{I}(\mathcal{A}')}$.

- $C = D \sqcup E$: After the algorithm terminates, the $\sqcup$-rule is not applicable anymore. That means that there is an individual name $x$, such that $\{x :$

$D, x : E\} \cap \mathcal{A}' \neq \emptyset$. By the induction hypothesis it holds that $x^{\mathcal{I}(\mathcal{A}')} \in D^{\mathcal{I}(\mathcal{A}')}$ or $x^{\mathcal{I}(\mathcal{A}')} \in E^{\mathcal{I}(\mathcal{A}')}$. Therefore $x^{\mathcal{I}(\mathcal{A}')} \in D^{\mathcal{I}(\mathcal{A}')} \cup E^{\mathcal{I}(\mathcal{A}')} = C^{\mathcal{I}(\mathcal{A}')}$.

- $C = succ(c)$: Since $\mathcal{A}'$ does not contain a clash, the QFBAPA solver must have returned a solution. If the solution is empty, then no individual names need to be introduced to satisfy $x : C$ and therefore $x^{\mathcal{I}(\mathcal{A}')} \in C^{\mathcal{I}(\mathcal{A}')}$. If the solution is not empty, then a new individual name $y_e$ is introduced for the ABox for each $e \in \mathcal{U}$. Therefore the induced interpretation is also updated with a new element $y_e^{\mathcal{I}(\mathcal{A}')}$. For each $e \in X_C$ we have $y_e : C \in \mathcal{A}'$. Therefore $y_e^{\mathcal{I}(\mathcal{A}')}$ must be in $C^{\mathcal{I}(\mathcal{A}')}$. For each $e \in X_r$ we have $(x, y_e) : r \in \mathcal{A}'$ and therefore $(x^{\mathcal{I}(\mathcal{A}')}, y_e^{\mathcal{I}(\mathcal{A}')}) \in r^{\mathcal{I}(\mathcal{A}')}$. Lastly for each $e \in X_{succ(d)}$ we have $y_e : succ(d) \in \mathcal{A}'$. By the induction hypothesis it must hold that $y_e^{\mathcal{I}(\mathcal{A}')} \in succ(d)^{\mathcal{I}(\mathcal{A}')}$. Since the solver returns a valid solution, we know that $x^{\mathcal{I}(\mathcal{A}')} \in succ(c)^{\mathcal{I}(\mathcal{A}')}$.

As $\mathcal{I}(\mathcal{A}')$ satisfies $\mathcal{A}'$ and it holds that $\mathcal{A} \subseteq \mathcal{A}'$, $\mathcal{I}(\mathcal{A}')$ must also satisfy $\mathcal{A}$. □

Next we prove that the algorithm can construct ABoxes with which every possible model within the pre-computed upper bound can be induced.

**Lemma 4** (Completeness)**.** If $\mathcal{A} := \{x : C\}$ is satisfiable then the tableau algorithm terminates without a clash.

*Proof.* By Lemma 2 we know that the tableau algorithm always terminates. It remains to show that the algorithm terminates returning a clash-free ABox. Since $\mathcal{A}$ is satisfiable it does not contain a clash. Let $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ be an interpretation which satisfies $\mathcal{A}$. We show that if $\mathcal{A}_i$ does not contain a clash and $\mathcal{I}$ satisfies $\mathcal{A}_i$, then $\mathcal{A}_{i+1}$ can be obtained from $\mathcal{A}_i$ by applying a rule without introducing a clash and while maintaining satisfiability by $\mathcal{I}$.
As stated $\mathcal{I}$ satisfies the clash-free ABox $\mathcal{A} =: \mathcal{A}_0$ (induction base). Let $\mathcal{A}_i$ be a clash-free ABox which is satisfied by $\mathcal{I}$ (induction hypothesis). We distinguish cases based on the rules applied on $\mathcal{A}_i$ to obtain $\mathcal{A}_{i+1}$ (induction step):

- the algorithm applies the $\sqcap$-rule on $x : C \sqcap D$: The interpretation $\mathcal{I}$ satisfies $\mathcal{A}_{i+1} = \mathcal{A}_i \cup \{x : C, x : D\}$ because by the hypothesis $\mathcal{I}$ already satisfies $\mathcal{A}_i$ and hence also $x : C \sqcap D$. That means that $x^{\mathcal{I}} \in (C \sqcap D)^{\mathcal{I}}$ and therefore $\{x, C, x : D\} \cup \mathcal{A}_i$ is satisfied by $\mathcal{I}$.

- the algorithm applies the $\sqcup$-rule on $x : C \sqcup D$: It has to be shown that either $\mathcal{A}_{i+1} = \mathcal{A}_i \cup \{x : C\}$ or $\mathcal{A}_{i+1} = \mathcal{A}_i \cup \{x : D\}$ is satisfied by $\mathcal{I}$. By the induction hypothesis $\mathcal{A}_i$ is satisfied by $\mathcal{I}$ and hence $x^{\mathcal{I}} = (C \sqcup D)^{\mathcal{I}}$. So either $x^{\mathcal{I}} \in C^{\mathcal{I}}$ and hence the algorithm chooses $\mathcal{A}_{i+1} = \mathcal{A}_i \cup \{x : C\}$ or $x^{\mathcal{I}} \in D^{\mathcal{I}}$ and hence the algorithm chooses $\mathcal{A}_{i+1} = \mathcal{A}_i \cup \{x : D\}$. In both cases $\mathcal{I}$ satisfies $\mathcal{A}_{i+1}$.

- the algorithm applies the *succ*-rule on $x : succ(c)$: It has to be shown that during that step successors are added such that $\mathcal{I}$ satisfies $\mathcal{A}_{i+1}$. First all *succ*-assertions are gathered in a set $\mathcal{S}$, transformed to a QFBAPA formula

$\phi(x)$ and given to a solver that returns all possible solutions within an upper bound. Because $\mathcal{A}_i$ is satisfiable, $\mathcal{S}$ is also satisfiable (subset of $\mathcal{A}_i$). Hence there have to be solutions which are returned by the solver. We need to show that the solver is capable of returning a solution within the upper bound, such that $\mathcal{A}_{i+1}$ is satisfied by $\mathcal{I}$. In case $x^{\mathcal{I}}$ has no successors, the empty solution must be a valid solution, which can be returned the solver. If $\mathcal{I}$ is finite and the number of $x$'s successors within each Venn region is within our upper bound, then we can create a solution $\sigma$ induced by $\mathcal{I}$, which can be returned by our solver. In any other case it has to be shown that a (finite) solution can be created from $\mathcal{I}$, which the solver is able to return. We know that $x^{\mathcal{I}}$ must have a finite number of successors in $\mathcal{I}$. Therefore we can create a solution $\sigma$ based on that: Let $\sigma$ be an empty solution. For each $e \in \bigcup_{r \in \mathbf{R}} r^{\mathcal{I}}(x)$ we add $e$ to $\sigma(\mathcal{U})$:

- for each $(x^{\mathcal{I}}, e) \in r^{\mathcal{I}}$ add $e$ to $\sigma(X_r)$
- for each $e \in C^{\mathcal{I}}$ add $e$ to $\sigma(X_C)$
- for each $e \in succ(c)^{\mathcal{I}}$ add $e$ to $\sigma(X_c)$

It is clear that if the solver returns this solution, then $\mathcal{I}$ satisfies $\mathcal{A}_{i+1}$. If each Venn region of this solution has more elements than the calculated upper bound, we can reduce the number of successors by applying the following steps [8]: Convert the QFBAPA formula to a system of linear equations $An = b$ like in Section 3.2. There has to be a solution, because $\mathcal{S}$ is satisfiable. Let $n$ be a solution to this system such that $n_k = |\{e | e \in \sigma(X_1^i) \cap \cdots \cap \sigma(X_m^i)\}|$, where $X_1^i \cap \cdots \cap X_m^i$ is the $k$-th Venn region. Then $n$ can be reduced to $n'$ like shown in Section 3.2. With the help of $n'$ a new solution $\sigma'$ is created by adding $n'_k$ successors to the $k$-th Venn region. It holds that (†):

- $\sigma^*(\mathcal{U}) \subseteq \sigma(\mathcal{U})$
- for each $e \in \sigma^*(\mathcal{U})$: $e \in \sigma^*(X_v)$ iff $e \in \sigma(X_m)$ with $m \in \mathbf{C} \cup \mathbf{R} \cup \{succ(c) | x : succ(c) \in \mathcal{A}_i\}$

This holds true because no successors are added to any Venn-region in order to obtain $n'$.
The algorithm then creates individual names according to the solution $\sigma'$, which leads to the satisfaction of all *succ*-assertions of $x$. For each element $e \in \sigma'(\mathcal{U})$ there is an individual name $y_e$ such that:

- $y_e : C \in \mathcal{A}_{i+1}$ iff $e \in \sigma'(X_C)$, $C \in \mathbf{C}$
- $y_e : succ(c) \in \mathcal{A}_{i+1}$ iff $e \in \sigma'(X_{succ(c)})$
- $(x, y_e) : r \in \mathcal{A}_{i+1}$ iff $e \in \sigma'(X_r)$

Because of (†) it can be concluded that:

- $y_e : C \in \mathcal{A}_{i+1}$ iff $e \in \sigma(X_C)$, $C \in \mathbf{C}$

- $y_e : succ(c) \in \mathcal{A}_{i+1}$ iff $e \in \sigma(X_{succ(c)})$
- $(x, y_e) : r \in \mathcal{A}_{i+1}$ iff $e \in \sigma(X_r)$

Since $\sigma$ is induced by $\mathcal{I}$:

- $y_e : C \in \mathcal{A}_{i+1}$ iff $e \in C^{\mathcal{I}}$, $C \in \mathbf{C}$
- $y_e : succ(c) \in \mathcal{A}_{i+1}$ iff $e \in succ(c)^{\mathcal{I}}$
- $(x, y_e) : r \in \mathcal{A}_{i+1}$ iff $(x^{\mathcal{I}}, e) \in r^{\mathcal{I}}$

Hence we can extend $\mathcal{I}$ by $y_e^{\mathcal{I}} = e$ which satisfies $\mathcal{A}_{i+1}$.

□

# Chapter 5

# Conclusion

The description logic $\mathcal{ALCSCC}$ is an extension to the well-known description logic $\mathcal{ALCQ}$ which adds set constraints and cardinality constraints over role successors. These roles successors are hard to deal with when checking satisfiability of $\mathcal{ALCSCC}$ concepts. This work presents a way of checking satisfiability of $\mathcal{ALCSCC}$ concepts, by constructing a tableau algorithm which uses a QFBAPA solver to deal with role successors. In [10] Tobies and Ganzinger show that checking satisfiability for $\mathcal{ALCSCC}$ is in PSpace, however the tableau algorithm we present runs in 2ExpSpace: We know that for each *successor*-rule application there can be an exponential number of successors introduced in the worst case which is shown in Section 3.2. Each of the newly added successors is also capable of introducing exponentially many successors. The advantage of using a tableau algorithm is that it does not only check satisfiability but also returns a satisfying interpretation (model) for the concept.
For future works the presented algorithm can be extended for $\mathcal{ALCSCC}$ concepts w.r.t. a TBox. In [1] Baader proves that the satisfiability problem w.r.t. a TBox is in ExpTime. One approach to extend the algorithm would be to add all information that can be concluded from the TBox first i.e. if it holds that $C \sqsubseteq D$ (every individual name in $C$ is also in $D$) and $x : C$ then $x : D$ has to be added to the ABox. This has to be done for every newly introduced individual name.
One drawback of the presented tableau algorithm is that it relies on a QFBAPA solver which is used as a black-box solver. Therefore an interesting extension to this work would be to modify the tableau algorithm such that it works without using a black-box solver. Doing this would most likely require the introduction of blocking techniques to prevent endless loops of adding and merging (or replacing) individual names like it has been done for $\mathcal{ALCQ}$ and the $\mathcal{SI}$ families.
Another interesting area to do research on is the pre-computed upper bound. In the presented work this bound grows exponentially with the number of successor assertions, which causes the complexity to be in 2ExpSpace. Finding a smaller upper bound would prove that the algorithm does have a lower complexity.

# Bibliography


[1] F. Baader. A New Description Logic with Set Constraints and Cardinality Constraints on Role Successors, 2017.

[2] F. Baader and T. Nipkow. *Abstract Reduction Systems*, page 7–33. Cambridge University Press, 1998.

[3] V. Haarslev. Combining Tableau and Algebraic Methods for Reasoning with Qualified Number Restrictions in Description Logics. 05 2002.

[4] R. Hoehndorf, P. N. Schofield, and G. V. Gkoutos. The Role of Ontologies in Biological and Biomedical Research: a Functional Perspective. *Brief. Bioinform*, page 1069–1080, 2015.

[5] B. Hollunder and F. Baader. Qualifying Number Restrictions Concept Languages. Technical report, Research Report RR-91-03, 1991.

[6] I. Horrocks, U. Sattler, and S. Tobies. Practical Reasoning for Very Expressive Description Logics. *CoRR*, cs.LO/0005014, 05 2000.

[7] V. Kuncak and M. Rinard. Towards Efficient Satisfiability Checking for Boolean Algebra with Presburger Arithmetic. In *Conference on Automateded Deduction (CADE-21)*, volume 4603 of *LNCS*, 2007.

[8] C. H. Papadimitriou. On the Complexity of Integer Programming. *J. ACM*, 28(4):765–768, Oct. 1981.

[9] S. Tobies. *Complexity Results and Practical Algorithms for Logics in Knowledge Representation*. PhD thesis, Rheinisch-Westfälischen Technischen Hochschule Aachen, 2001.

[10] S. Tobies and H. Ganzinger. A PSpace Algorithm for Graded Modal Logic. In *In Proc. CADE 1999, Lecture Notes in Artificial Intelligence*, volume 1632, pages 674–674, 01 1999.