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

November 4, 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 signiture

Contents

Chapter 1

Introduction

Traditional data bases where data are stored solely without any connection towards to themselves like many people would imagine are often not enough any more. The reason is that the data are stored without any semantics. However storing data with semantics can provide additional information. For example we have some data about two objects *"Anna"* and *"Beth"*. In a traditional data base, if not explicitly stated, both data are not related to each other. Nethertheless Anna and Beth can have a relation, which also depends on who or what both are. For example both can be human and Anna is a teacher and Beth is a student and are assigned to the same class. By adding solely those information in a traditional data base the information that Anna must teache Beth is not given. One way to apply semantics to data objects is to use *ontologies*. In biological and (bio)medical researches data bases are often based on ontologies [2]. Ontologies (in the computer science field) can be viewed as formal representation of a certain domain of interest. In data base they are collection of relations between the entities in the data base and are formulated as a fragment of first-order logic (FOL). These fragments of FOL are represented as *Description Logic (DL)*, which is a family of knowledge representation system. DLs are mainly built of concepts, which correspond to unary relations in FOL and is often represented by a capital letter, and relation between the concepts, which correspond to binary relations in FOL and is often represented by a lowercase letter. For more complex (compound) concepts operators like $\sqcap$, $\sqcup$,$\sqsubseteq$, $\exists$ and $\forall$, depending on the DL, are used. For example the statement "All Men and Women are Human" is formalized in FOL as $\forall x.Men(x) \vee Women(x) \rightarrow Human(x)$ and in DL as an *axiom* $Men \sqcup Women \sqsubseteq Human$, where Men,$Women$ and $Human$ are concept names. The statement "All Humans, who have children, are parents" can be formalized in FOL as $\forall x \exists y.Human(x) \wedge hasChildren(x, y) \rightarrow Parent(x)$ and in DL as $Human \sqcap \exists hasChildren.\top \sqsubseteq Parent$, where $Human$ and $Parents$ are concept names and $hasChildren$ is a role name. Restriction with the operators $\exists$ and $\forall$ are called *quantified* restrictions. The second statement can also be formalized with a *qualified* restriction: $Human \sqcap \geq 1 hasChildren.\top \sqsubseteq Parent$. Each quantified restriction can be transformed into a qualified restriction.

A knowledge base consists of a *TBox*, which contains the axioms (rules), and of an *ABox* which contains assertions of certain elements (objects). One big research field in DL is the determination of satisfiability of a *knowledge base*, which is formulated in DL. In [3] a *Tableau*-algorithm is presented for checking satisfiability for an ABox in the DL $\mathcal{ALCQ}$. This DL allows conjunctions ($\sqcap$), disjunctions ($\sqcup$), negation $\neg C$ and qualifying number restriction ($\leq n\, r\, C$ and $\geq n\, r\, C$), where n is a number, r a role name, and C a concept name. A Tableau-algorithm applies *completion rules* to a given *set*(ABox) to decompose complex concepts and try satisfiying violated assertions. The satisfiability (of concepts) is stated in [3] as PSPACE-hard problem (without TBox, with TBox it is EXPTime-hard [1]). In [6] a optimized Tableau-algorithm is presented which states that it is a PSPACE-problem. The optimization is that instead of keeping n successors to satisfy a restriction $\geq n\, r.C$ like in [3], the algorithm saves the number of existing successors and by comparing the numbers detects possible *clashes*.
The expressive DL $\mathcal{ALCSCC}$ extends $\mathcal{ALCQ}$ with *set constraint* and *cardinality constrant*, which lays under the logic of *QFBAPA* (quantifier-free fragment of Boolean Algebra with Presburger Arithmetic). As the name says we do not have quantifier. Instead we use set expression (Boolean Algebra part) and numerical constraint (Presburger Arithmetic) which is combined together with cardinality functions. For example $Human \sqcap \geq 1\, hasChildren.\top \sqsubseteq Parent$ is written in $\mathcal{ALCSCC}$ as $Human \sqcap succ(|hasChildren| \geq 1) \sqsubseteq Parent$. This DL is more expressive than $\mathcal{ALCQ}$ because every qualified restriction $\leq n\, r.C$ and $\geq n\, r.C$ can be written in $\mathcal{ALCSCC}$ as $succ(|r.C| \leq 1)$ and $succ(|r.C| \geq 1)$. In [1] a solution for the satisfiability problem (without TBox) is presented which has the complexity PSpace: For an ABox we guess the value (true or false) of the top-level variables which can already lead to a *false*-result. If not then the constraint is formulated into a $QFBAPA$ formula, for which an algorithm determine by guessing a number N of Venn-region to be non-empty whether the formula is satisfied or not.
In this work we present a Tableau-algorithm for $\mathcal{ALCSCC}$. As in previous work for other DLs we define completion rules which can be applied onto assertions in the ABox to determine whether $\bot$ can be concluded from it which states its unsatisfiability. If we can not apply any rules any more and we can not conclude $\bot$, then the ABox is satisfiable. The main difficulty is that unlike $\mathcal{ALCQ}$, where the bond of number of successor is fixed, in $\mathcal{ALCSCC}$ we can compare two cardinalities, which can vary during the algorithm. Hence we need an approach for counting successors and with it calculating the *correct* cardinality, which is necessary to detect satisfied and violated constraint. For this we introduce *induced interpretation* which can determine the cardinalities after each rule application. Further more to deal with the numerical arithmetic of $\mathcal{ALCSCC}$ we use a $QFBAPA$ solver, of which we assume it is capable to return all possible solution. We transform a subset of the ABox into a $QFBAPA$ formula and then let a solver determine whether the formula is satisfiable or not. If not we end with a clash. If it returns a solution, then we add individual names accordingly to our ABox.

Chapter 2

Preliminaries

Before we define the DL $\mathcal{ALCSCC}$ we have to explain first how the language $QFBAPA$ looks like.

2.1 $QFBAPA$

The logic $QFBAPA$ combines boolean algebra (BA) of sets of symbols with Presburger arithmetic (PA):

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 also $s \cap t$, $s \cup t$ 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 also $k + l$ 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 $n\ \neg dvd\ k$

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

A $QFBABA$ formula ϕ is a disjunction ($\vee$) and conjunction ($\wedge$) of (also possible negated) cardinality constraints, where every set symbol is represented as a set variable.

Since $s \subseteq t$ can be expressed as the cardinality constraint $|s \cap t^{\neg}| \leq 0$ we will not consider any set constraints further in this work. In case we want to express $x : succ(s = t)$, with s, t being set terms, we write instead $x : succ(|s \cap t^{\neg}| \leq 0) \sqcap succ(|s^{\neg} \cap t| \leq 0)$. Furthermore instead of $l \geq k$ we write $k \leq l$, instead of $k < l$ we write $k+1 \leq l$ and instead of $k = l$ we write $k \leq l$ and $l \leq k$. Hence for an assertion $x : succ(c)$ the cardinality constraint c is either of the form $k \leq l$ or $n\, dvd\, l$.
The semantic of $QFBAPA$ is define as follows:

Definition 2 (Interpretation of $QFBAPA$)**.** Let $\Delta^{\mathcal{I}}$ be a finite set and σ a mapping which maps

- every symbol a in T to $\sigma(a) \subseteq \mathcal{U}$
- $\emptyset$ to $\sigma(\emptyset)$
- $\mathcal{U}$ to $\sigma(\mathcal{U}) \subseteq \Delta^{\mathcal{I}}$
- $\sigma(s \cap t) := \sigma(s) \cap \sigma(t)$, $\sigma(s \cup t) := \sigma(s) \cup \sigma(t)$
- $\sigma(s^{\neg}) := \sigma(\mathcal{U}) \backslash \sigma(s)$
- $\sigma(|s|) := |\sigma(s)|$
- $\sigma(k + l) := \sigma(k) + \sigma(l)$, $\sigma(n \cdot k) := n \cdot \sigma(k)$

The mappings satisfies for the cardinality terms k, l

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

The mapping σ is a solution of a $QFBAPA$ formula ϕ if it satisfies all cardinality constraints in ϕ.

2.2 $\mathcal{ALCSCC}$

Next we define the description logic $\mathcal{ALCSCC}$. 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 are defined inductively:

- all concept names
- if C, D are concepts then:
  - $\neg C$
  - $C \sqcup D$
  - $C \sqcap D$
- $succ(c)$ if c is a cardinality constraint over $\mathcal{ALCSCC}$ concepts and role names

An ABox $\mathcal{A}$ in $\mathcal{ALCSCC}$ 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 $\mathcal{A}$.

Definition 4 (Interpretation)**.** An *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ over an ABox $\mathcal{A}$ in $\mathcal{ALCSCC}$ 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 $x^{\mathcal{I}} \in \Delta^{\mathcal{I}}$
- every concept names $A \in \mathbf{C}$ to $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
- every 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 $(x, y) \in r^{\mathcal{I}}$ e.g. 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 $\cdot^{\mathcal{I}_x}$ 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 $(x^{\mathcal{I}}, y^{\mathcal{I}}) \in r^{\mathcal{I}}$

Note that in a concept $succ(c)$ c can also be a cardinality constraint over $A \sqcap B$ and $A \sqcup B$. Since the semantic of $A \sqcap B$ and $A \cap B$ (analogously for $A \sqcup B$ and $A \cup B$) are defined the same way ($(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$ and $\sigma(C \cap D) = \sigma(C) \cap \sigma(D)$), we make the assumption that every $\sqcap$ and $\sqcup$ within a $succ(c)$ concept are replaced with $\cap$ and $\cup$.

Chapter 3

Tableau for $\mathcal{ALCSCC}$

A Tableau-algorithm consists of completion rules to decide satisfiability of a set of assertions. The rules are applied exhaustively on the set until none is applicable any more. One major characteristic of this algorithm is that it does not matter in which order the rules are applied. Another characteristic is that it works non-deterministically: In case we have disjunctions we can choose between the concepts in this disjunction. If a choice ends in a *clash* then we track back to the point where we had to chose and take the other choice instead. If all choices end in a clash then the ABox is unsatisfiable, otherwise it is satisfiable. We want to use the Tableau-algorithm to check whether an assertion $x : C$ is satisfiable or not and if it satisfiable then we create a satisfied ABox from $x : C$. To help the algorithm we want to avoid nested negation e.g. $\neg(\neg(\neg(A \cup B)))$. Hence we consider all concepts in *negated normal form (NNF)*.

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

- $\neg\top \to \bot$
- $\neg \bot \to \top$
- $\neg\neg C \to C$
- $\neg(C \sqcap D) \to \neg C \sqcup \neg D$
- $\neg(C \sqcup D) \to \neg C \sqcap \neg D$
- $C^{\neg} \to \neg C$
- $\neg succ(c) \to succ(\neg c)$
- $\neg(k \leq l) \to l \leq k$
- $\neg(n\ dvd\ k) \to n\ \neg dvd\ k$
- $\neg(n\ \neg dvd\ k) \to n\ dvd\ k$
- $(s \cap t)^{\neg} \to s^{\neg} \cup t^{\neg}$
- $(s \cup t)^{\neg} \to s^{\neg} \cap t^{\neg}$
- $(s^{\neg})^{\neg} \to s$

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. It can be transformed into $\neg C$: For every interpretation σ for a concept C based on $QFBAPA$ we have $\sigma(C^\neg) = \sigma(\mathcal{U})\backslash\sigma(C)$ and for every interpretation $\mathcal{I}$ based on $\mathcal{ALCSCC}$ we have $(\neg C)^\mathcal{I} = \Delta^\mathcal{I}\backslash C^\mathcal{I}$. Since $\sigma(\mathcal{U}) \subseteq \Delta^\mathcal{I}$ we can conclude that every element in $\sigma(C^\neg)$ is also in $(\neg C)^\mathcal{I}$.
The first five rules on the left hand side can be applied in linear time [3],[5]. The first four rules on the right hand side and the rules $C^\neg \to \neg C$ and $\neg succ(c) \to succ(\neg c)$ can also be applied in linear time since we only shift the negation sign. The rule $(s^\neg)^\neg \to s$ works similarly to $\neg\neg C \to C$ and the rules $(s \cap t)^\neg \to s^\neg \cup t^\neg$ and $(s \cup t)^\neg \to s^\neg \cap t^\neg$ works similarly to $\neg(C \sqcap D) \to \neg C \sqcup \neg D$ and $\neg(C \sqcap D) \to \neg C \sqcup \neg D$. Therefore all of them can also be applied in linear time.
Next we introduce *induced interpretation* with which we can count 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 $\mathcal{A}$ by the following steps:

- for 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 a set of assertion, x be an individual name, k be a cardinality term and $n \in \mathbb{N}$. An assertion $x : succ(c)$ is *violated* if $x^{\mathcal{I}(\mathcal{A})} \notin succ(c)^{\mathcal{I}(\mathcal{A})}$.

Like already mentioned an ABox is unsatisfiable if all choices ends in a clash.

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

3.1 Transforming an ABox into a formula

Dealing with numerical arithmetic is challenging and hence we use the help of a $QFBAPA$ solver, whenever we want to add successors for an individual name x. For that we collect all *succ*-assertion regarding x first and then transform them into a $QFBAPA$ formula for one *nested level*. We assume that the ABox is already in NNF. As example we look at

Example 1 (Example for transforming ABox into $QFBAPA$ formula)**.**

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

with $\mathbf{C} = \{A, B, C\}$ and $\mathbf{R} = \{r\}$ We first gather all *succ*-assertion regarding x together and transform it into a formula by doing the following steps:

- drop all $x : succ$
- 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 (possible compound) concepts and role names with set variables, for which a solver can assign elements to them. The last bullet point is important because sometimes it is not explicitly stated, what kind of successor an individual name has. That means that every successor is connected to its predecessor and/or to its successors, if available, by at least one role name.
For 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 only replace concepts on one *nested level*. In the example the first assertion tells that x must have at least one successor y which has more successors in $B \cap r$ than in A. The concept $succ(|A| \leq |B \cap r|)$ is on a different *nested level* then $succ(|A| \leq |B|)$. According to one solution of the solver we can introduce individual names and assertion. Also x has more successor in B than in A. By the last conjunction we give the information, that every successor must be an r-successor. Then in the upcoming steps we can gather all *succ*-assertion of the new individual names again and repeat this step.

3.2 Solution of a formula

For our algorithm we assume that we have a solver, which can return all possible solutions. However there can be infinite many solutions. Actually Example 1 can have infinitetly many solutions: We can always increase the amount of successors in B as long as we have fewer successors in A. However having a infinite solution means that the Tableau-algorithm works on a infinite space and/or does not terminate. Therefore we want to only consider solution inside some *upper bounds*. For an *Integer Linear Programming* (ILP) problem, which is describe as a linear system of equalities, possible upper bounds are already investigated (like in [4]). Therefore we want to transform our formula into a linear system of equalities in a form $Ax = b$, where A and b describe our cardinality constraints and x is the solution e.g. denotes the numbers of elements we have to assign to set variables to satisfy the constraints:

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-factor*. Let $c = succ(|A| \leq |B \cap r|)$. We arrange ϕ in (3.1) into

$$\phi = |X_c| \geq 1 \wedge |X_A| - |X_B| \leq 0 \wedge |\mathcal{U}| - |X_r| = 0$$

Then we change each inequalities into equalities by adding two slack variables I_1 and I_2:

$$\phi = |X_c| - I_1 = 1 \wedge |X_A| - |X_B| + I_2 = 0 \wedge |\mathcal{U}| - |X_r| = 0$$

Right now it is not clear whether the set variables are overlapping or not. Therefore we consider *Venn regions*, which is of the form $X_1^i \cap \cdots \cap X_k^i$. The subscript i denotes either 0 or 1. X_1^0 denotes $X_1^\neg$ and X_1^1 denotes X_1. Therefore we construct A and b such that instead of assigning elements to set variables we assign them to the Venn regions. Since we have five set variables, we have $2^5 = 32$ Venn region and therefore 32 vectors denoting the Venn regions. We also have two equation which means we need at most two more vectors for the slack variables. The matrix A therefore has two rows and 34 columns and a_{ij}, $1 \leq i \leq 2$ and $1 \leq j \leq 16$, denotes the sum of the pre-factors of the set variables, in which the j-th Venn region is included, in the i-th equation. We already see that 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 size of ϕ, 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 ϕ a number N, which is polynomial to the size of ϕ, can be computed in polynomial time such that for every solution σ of ϕ there exists a solution σ' of ϕ such that

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

Hence we guess (in non-deterministic polynomial time) a number N of Venn regions, which are non-zero. In Example 1 we can already guess, 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 be all in X_r. Hence we can drop 16 Venn regions and create a system of equation with A being a 2×10 matrix. Let the vectors for the Venn regions be in the following order:

- $v_1 = X_A \cap X_B \cap X_c \cap X_r \cap \mathcal{U}$
- $v_2 = X_A \cap X_B \cap X_c^\neg \cap X_r \cap \mathcal{U}$
- $v_3 = X_A \cap X_B^\neg \cap X_c \cap X_r \cap \mathcal{U}$
- $v_4 = X_A \cap X_B^\neg \cap X_c^\neg \cap X_r \cap \mathcal{U}$
- $v_5 = X_A^\neg \cap X_B \cap X_c \cap X_r \cap \mathcal{U}$
- $v_6 = X_A^\neg \cap X_B \cap X_c^\neg \cap X_r \cap \mathcal{U}$
- $v_7 = X_A^\neg \cap X_B^\neg \cap X_c \cap X_r \cap \mathcal{U}$
- $v_8 = X_A^\neg \cap X_B^\neg \cap X_c^\neg \cap X_r \cap \mathcal{U}$

The last two vectors denote the slack variables: $v_9 = I_1$ and $v_10 = I_2$ We create now the linear system of equation:

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

Note that $a_{2,1}$ and $a_{2,2}$ are 0 because the pre-factor of $|X_A|$ and $|X_B|$ in the second equation are 1 and -1 and the Venn regions v_1 and v_2 are included both in X_A and X_B. If $x_i = 0$ then the Venn region v_i is empty.
Going back to upper bound problem: The following result from [4] can be used to establish upper bound for the solution of the *ILP*:

Theorem 1 (Theorem 1 from [4])**.** 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 exists integers α_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 for which the solution is within our bound:

$$h^T A x' = h^T b$$

Therefore we are able to set an upper bound a priori for the solutions the $QFBAPA$ returns. Furthermore we stay in PSpace while calculating the upper bound by using $\leq N$ Venn regions after the results of [1].
In our example the upper bound for all x_i is $8 \cdot (2 \cdot max_{i,j}\{|1|, |-1|\})^{2 \cdot 2+1} = 512$. Therefore we can bound our number of elements in each Venn regions to 512.

3.3 Algorithm

Finally we can present the Tableau-algorithm for an ABox in $\mathcal{ALCSCC}$. Before we handle the numerical arithmetic of $\mathcal{ALCSCC}$ we want to decompose compound concepts first. Hence we divide the algorithm in two parts: a boolean part, where the decomposing of compound concepts takes place, and a numerical part, where a part of the ABox is transformed into a $QFBAPA$ formula.

Then we calculate an upper bound and let a solver return a possible solution within this bound, in case the ABox is satisfiable. The boolean part has a higher priority than the numerical part.

Definition 9 (Tableau)**.** Let $\mathcal{A}$ be a set of assertions in NNF.
Boolean part:

- $\sqcap$-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 $\mathcal{A} := \mathcal{A} \cup \{x : C_2\}$

Numerical part:

- *successor*-rule: $\mathcal{A}$ contains for an individual name x at least one violated assertion of the form $x : succ(c)$:
  - gather all assertion of the form $x : succ(c)$ into a set $\mathcal{S}$
  - transform $\mathcal{S}$ into a $QFBAPA$ formula ϕ after 3.1
  - calculate the upper bound after Theorem 1

  If a $QFBAPA$ solver returns *unsatisfiable* then $\mathcal{A} = \mathcal{A} \cup \{x :\bot\}$
  If a $QFBAPA$ solver returns *satisfiable* then select one solution σ with in the upper bound. For each $e \in \sigma(\mathcal{U})$ we introduce a new individual name y and
  - for each $e \in X_C$ we have $\mathcal{A} = \mathcal{A} \cup \{y : C\}$
  - for each $e \in X_{succ(c)}$ we have $\mathcal{A} = \mathcal{A} \cup \{y : succ(c)\}$
  - for each $e \in X_r$, $r \in \mathbf{R}$, we have $\mathcal{A} = \mathcal{A} \cup \{(x, y) : r\}$
  - for each $e \notin X_C$ we have $\mathcal{A} = \mathcal{A} \cup \{y : \neg C\}$
  - for each $e \notin X_{succ(c)}$ we have $\mathcal{A} = \mathcal{A} \cup \{y : NNF(\neg succ(c))\}$
  - for each $e \notin X_r$, $r \in \mathbf{R}$, we have $\mathcal{A} = \mathcal{A} \cup \{(x, y) : r^{\neg}\}$

A *complete* ABox is an ABox, on which no more rules of the Tableau-algorithm are applicable.
We pick up Example 1 again but in a form where it is not decomposed yet:

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

We are able to apply the boolean rules and hence we apply them first and decompose into Example 1. Then we apply the *successor*-rule: We collect every *succ*-assertion regarding x to a set $\mathcal{S} := \{x : succ(1 \leq |succ(|A| \leq |B \cap r|)|), x : succ(|A| \leq |B|)\}$ and convert $\mathcal{A}$ to the $QFBAPA$ formula $1 \leq X_{succ(|A| \leq |B \cap r|)} \wedge |X_A| \leq |X_B| \wedge \mathcal{U} = \{X_r\}$. Then we let a solver return an assignment. In this case we see that the formula is satisfiable with $X_{succ(|A| \leq |B \cap r|)} = \{f\}, X_A = \{\}, X_B = \{e\}$ and $X_r = \{e, f\}$. Since we have $\mathcal{U} = \{X_r\}$ every element must be in X_r every successor is a r-successor. We then introduce for e ans f two individual names y and z and the assertion $y : X_B$, $(x, y) : r$, $z : succ(|A| \leq |B \cap r|)$ and $(x, z) : r$ to S. Then for z we have to apply the *successor*-rule again.

Chapter 4

Correctness

For the correctness proof of the Tableau-algorithm we have to show that

- For every input the Tableau-algorithm terminates
- 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

In all proves we assume that the $QFBAPA$ solver is correct. First we prove that the algorithm always terminates.

4.1 Termination of the Tableau-Algorithm

We define a *derived* ABox as an ABox after a finite number of rule application. For the termination proof we map every *derived* ABox to an element $\Psi(\mathcal{A})$ of a set Q. This set Q can be ordered by a well-founded strict partial ordering $\prec$. Since $\prec$ is well-founded we can not have a infinite increasing chain which results into termination of the algorithm. Therefore we show for every ABox $\mathcal{A}'$ which is derivable from a derived ABox $\mathcal{A}$ that $\Psi(\mathcal{A}') \prec \Psi(\mathcal{A})$.
The element of Q are multisets of triples. Each triple consists of a multiset and an integer. A multiset M is smaller than a multiset M' if we can obtain M from M' by replacing at least one element of M' with at least one element, which is smaller. In our definition of Q $\Psi(\mathcal{A}')$ is smaller than $\Psi(\mathcal{A})$ if we can replace at least one triple of $\Psi(\mathcal{A})$ with at least one smaller triple to obtain $\Psi(\mathcal{A}')$. A triple (x_1, y_1) is smaller than a triple (x_2, y_2) if either $x_1 < x_2$ or $x_1 = x_2$ and $y_1 < y_2$ (lexicographical odering from left to right). Otherwise (x_2, y_2) is smaller.
Next we define the size of a concept C inductively:

- $size(A) = 1$ if $A \in \mathbf{C}$
- $size(\neg C) = size(C)$
- $size(C \sqcap D) = size(C \sqcup D) = size(C) + size(D)$

- $size(succ(c)) = \begin{cases} 1 + max\{size(k), size(l)\} & c = k \leq l \\ 1 + size(l) & c = n\, dvd\, l \end{cases}$

The size of a cardinality terms k is also defined inductively:

- $size(n) = 1$ if n is an integer
- $size(C^{\neg}) = size(C)$
- $size(C \cap D) = size(C \cup D) = size(C) + size(D)$

Definition 10. Let $\mathcal{A}$ be a derived ABox. The multiset $\Psi(\mathcal{A})$ consists of triples. Each triple $\psi_{\mathcal{A}}(x)$ represent one individual name x:

- the first component is an integer which denotes $max\{size(C)|x : C \in \mathcal{A}\}$
- the second component is a multiset which contains for each assertion $x : C \sqcap D \in \mathcal{A}$ for which the $\sqcap$-rule is applicable the integer $size(C \sqcap D)$. Respectively for $x : C \sqcup D$
- the third component is an integer which denotes the number of $x : succ(c) \in \mathcal{A}$ for which the *successor*-rule is applicable

We are now able to finally proof 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 obtainable from $\mathcal{A}$ by a rule from Definition 3.3, then we have $\Psi(\mathcal{A}') \prec \Psi(\mathcal{A})$.

- $\mathcal{A}'$ is obtained from $\mathcal{A}$ by applying the $\sqcap$-rule: The first component remains unchanged because $size(C) \leq size(C \sqcap D)$ and $size(D) \leq size(C \sqcap D)$. The second component becomes smaller because the integer for $size(C \sqcap D)$ is removed (because we can not apply this rule any more after one application). In case C and/or D happens to be disjunction or conjunction the multiset still becomes smaller because $size(C) \leq size(C) + size(D) = size(C \sqcap D)$ (respectively for $size(D)$). Hence the triple $\psi_{\mathcal{A}'}(x)$ always becomes smaller.
- $\mathcal{A}'$ is obtained from $\mathcal{A}$ by applying the $\sqcup$-rule: similar to above
- $\mathcal{A}'$ is obtained from $\mathcal{A}$ by applying the *successor*-rule: The first component remains unchanged because we do not add any assertion for x. Because we are able to apply this rule, both $\sqcap$-rule and $\sqcup$-rule are not applicable on $\mathcal{A}$. Hence the second component of $\psi_{\mathcal{A}}(x)$ remains unchanged. In this rule we gather all *succ*-assertion of x and convert them to a $QFBAPA$ formula. If a solver returns unsatisfiable for the formula we have add $x : \bot$ which means we end with a clash. If the solver returns a solution, we add successors according to the solution. Hence all *succ*-assertion of x

becomes satisfied and the third component decreases. Therefore $\psi_{\mathcal{A}'}(x)$ is smaller than $\psi_{\mathcal{A}}(x)$.
For every freshly introduced individual name y we have to add a triple $\psi_{\mathcal{A}'}(y)$ to Q. Since y is a successor the first component of $\psi_{\mathcal{A}'}(y)$ is always smaller than the first component of $\psi_{\mathcal{A}'}(x)$:

- We know that for any successor it has only one predecessor: A new individual name y can only be introduced by the application of the *successor*-rule on one individual name x, which has at least one *succ*-assertion. Furthermore, we can not add an assertion $(a, b) : r, r \in \mathbf{R}$, such that $a, b \in I(\mathcal{A})$ are not in a predecessor-successor relation before. Also we can only add individual names on one nested level. Therefore we can not add new individual names, which are in a predecessor-successor relation to each other.
- Since y has only one successor we know 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 $size(C)$: For each $y : C \in \mathcal{A}'$ we know that C must be part of a cardinality constraint c such that $x : succ(c) \in \mathcal{A}$ and therefore $size(succ(c)) > size(C)$.

For any other individual name y, such that $y \neq x$ and y was not freshly introduced, the triple $\psi_{\mathcal{A}}(y)$ remains unchanged. Hence in all three cases we can obtain $\Psi(\mathcal{A}')$ 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 corresponding triples are always smaller than $\psi_{\mathcal{A}'}(x)$. Therefore it remains $\Psi(\mathcal{A}') \prec \Psi(\mathcal{A})$.
Because we work with integers the strict partial ordering is indeed well-founded.
□

4.2 Soundness and Completeness

After we proved that the algorithm terminates, we continue with the correctness of the algorithm e.g. the algorithm returns a satisfied ABox iff the ABox is satisfiable.

Lemma 3. 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 result after the algorithm terminated. Since we do not remove any assertion during the algorithm we have $\mathcal{A} \subseteq \mathcal{A}'$. 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 concepts:
For each concept name $C \in \mathbf{C}$ such that $x : C \in \mathcal{A}'$, we have $x^{\mathcal{I}(\mathcal{A}')} \in C^{\mathcal{I}(\mathcal{A}')}$ by the definition of induced interpretation. (induction base)
We consider $x : C$ where C is a compound concepts (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 definition of induced interpretation and $x : D \notin A$ we have $x^{\mathcal{I}(\mathcal{A}')} \notin D^{\mathcal{I}(\mathcal{A}')}$ which implies $x^{\mathcal{I}(\mathcal{A}')} \in \Delta^{\mathcal{I}(\mathcal{A}')} \backslash D^{\mathcal{I}(\mathcal{A}')} = C^{\mathcal{I}(\mathcal{A}')}$.

- $C = D \sqcap E$: Since the algorithm terminated, the $\sqcap$-rule is not applicable any more. That means that there is an individual name x, such that $\{x : D, x : E\} \subseteq \mathcal{A}'$. By the induction hypothesis we have $x^{\mathcal{I}(\mathcal{A}')} \in D^{\mathcal{I}(\mathcal{A}')}$ and $x^{\mathcal{I}(\mathcal{A}')} \in E^{\mathcal{I}(\mathcal{A}')}$. Therefore $x^{\mathcal{I}(\mathcal{A}')} \in C^{\mathcal{I}(\mathcal{A}')} \cap D^{\mathcal{I}(\mathcal{A}')} = (C \sqcap D)^{\mathcal{I}(\mathcal{A}')}$.

- $C = D \sqcup E$: Since the algorithm terminated, the $\sqcup$-rule is not applicable any more. That means that there is an individual name x, such that $\{x : D, x : E\} \cap \mathcal{A}' \neq \emptyset$. By the induction hypothesis we have $x^{\mathcal{I}(\mathcal{A}')} \in D^{\mathcal{I}(\mathcal{A}')}$ or $x^{\mathcal{I}(\mathcal{A}')} \in E^{\mathcal{I}(\mathcal{A}')}$. Therefore $x^{\mathcal{I}(\mathcal{A}')} \in C^{\mathcal{I}(\mathcal{A}')} \cup D^{\mathcal{I}(\mathcal{A}')} = (C \sqcup D)^{\mathcal{I}(\mathcal{A}')}$.

- $C = succ(c)$: 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 are needed to be introduced to satisfy $x : C$ and we have $x^{\mathcal{I}(\mathcal{A}')} \in C^{\mathcal{I}(\mathcal{A}')}$. If the solution is not empty then the induced interpretation is updated by introducing a new element $y^{\mathcal{I}(\mathcal{A}')}$ for each $e \in \sigma(\mathcal{U})$ and hence also for each freshly introduced individual name y. For each $e \in X_C$ we have $y : C \in \mathcal{A}'$. By the induction hypothesis $y^{\mathcal{I}(\mathcal{A}')}$ must be in $C^{\mathcal{I}(\mathcal{A}')}$. For each $e \in X_r$ we have $(x, y) : r$. By the induction step we know there must be an element $x^{\mathcal{I}(\mathcal{A}')} \in \Delta^{\mathcal{I}(\mathcal{A}')}$. Because we introduced e in this step, we also introduced a new individual name y which means for the induced interpretation we introduce a new element $y^{\mathcal{I}(\mathcal{A}')}$. Since we also added $(x, y) : r$ to $\mathcal{A}'$, we must have $(x^{\mathcal{I}(\mathcal{A}')}, y^{\mathcal{I}(\mathcal{A}')}) \in r^{\mathcal{I}(\mathcal{A}')}$. Lastly for each $e \in X_c$, c is a cardinality constraint, we have $y : succ(c)$. Again by the induction hypothesis $y^{\mathcal{I}(\mathcal{A}')} \in succ(c)^{\mathcal{I}(\mathcal{A}')}$. Since the solution is correct, we know that $x^{\mathcal{I}(\mathcal{A}')} \in succ(c)^{\mathcal{I}(\mathcal{A}')}$.

Since we know that $\mathcal{I}(\mathcal{A}')$ satisfies $\mathcal{A}'$ and that $\mathcal{A} \subseteq \mathcal{A}'$, $\mathcal{I}(\mathcal{A}')$ also satisfies $\mathcal{A}$. □

Lemma 4. If $\mathcal{A} := \{x : C\}$ is satisfiable then the Tableau-algorithm terminates without a clash.

Proof. By Lemma 1 we know, that the algorithm always terminates. It remains to show that the algorithm terminates with a satisfied 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 while maintaining clash-free and satisfied by $\mathcal{I}$.
We already stated that $\mathcal{I}$ satisfies the clash-free $\mathcal{A} =: \mathcal{A}_0$. (induction base). Let $\mathcal{A}_i$ be a clash-free ABox, which is satisfied by $\mathcal{I}$. (induction hypothesis). We distinguish the cases based on the rules, we apply on $\mathcal{A}_i$ to obtain $\mathcal{A}_{i+1}$ (induction step):

- we apply 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 $\mathcal{I}$

- we apply the $\sqcup$-rule on $x : C \sqcup D$: We have to show 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 $\mathcal{I}$. Again by the induction hypothesis $\mathcal{A}_i$ is satisfied by $\mathcal{I}$ and hence $x^{\mathcal{I}} = (C \sqcup D)^{\mathcal{I}}$. So either we choose $\mathcal{A}_{i+1} = \mathcal{A}_i \cup \{x : C\}$ and hence $x^{\mathcal{I}} \in C^{\mathcal{I}}$ or we choose $\mathcal{A}_{i+1} = \mathcal{A}_i \cup \{x : D\}$ and hence $x^{\mathcal{I}} \in D^{\mathcal{I}}$. In both cases $x^{\mathcal{I}} \in C^{\mathcal{I}} \cup D^{\mathcal{I}} = (C \sqcup D)^{\mathcal{I}}$ and hence $\mathcal{I}$ satisfies $\mathcal{A}_{i+1}$.

- we apply the *succ*-rule on $x : succ(c)$: We have to show that by this step we are able to add successors, such that $\mathcal{I}$ satisfies $\mathcal{A}_{i+1}$. In this step, we gather first all *succ*-assertion together in a set $\mathcal{S}$, formulate a $QFBAPA$ formula $\phi(x)$ and let a solver return us all possible solution with in an upper bound. Because $\mathcal{A}_i$ is satisfiable, $\mathcal{S}$ is also satisfiable (subset of $\mathcal{A}$). Hence there has to be solutions which can be returned by the solver. We need to show, that the solver is capable to return a solution within our 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 from the solver. If $\mathcal{I}$ is finite and within our upper bound, then we can create a solution σ induced by $\mathcal{I}$, which can be returned by our solver. In any other case we have to show, that we can create a (finite) solution 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 σ based on that: Let σ be a empty solution. For every $C \in \mathbf{C}$, $r \in \mathbf{R}$, c such that $x : succ(c) \in \mathcal{A}$, let X_C, X_r, X_c be set variables. For each $y^{\mathcal{I}} \in \bigcup_{r \in \mathbf{R}} r^{\mathcal{I}}(x)$ we introduce an element y_e and add it to $\sigma(\mathcal{U})$:

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

  It is clear that if the solver returns this solution then $\mathcal{I}$ satisfies $\mathcal{A}_{i+1}$ otherwise $\mathcal{I}$ does not satisfy $\mathcal{A}_i$. If $x^{\mathcal{I}}$ has more successors than the calculated upper bound, we can reduce the number of successors after [4]: Convert the $QFBAPA$ formula to a linear system of equations $An = b$. We then create the solution n: If v_k is the k-th Venn region $X_1^i \cap \ldots X_n^i$, $i \in \{0, 1\}$ then let $n_k = |\{e | e \in X_1^i \cap \ldots X_n^i\}|$. We know that n must be a solution to the linear system because otherwise $\mathcal{I}$ does not satisfy $\mathcal{A}_i$. Then we can reduce n after Theorem 1 to n'. With the help of n' we create the new solution σ^* by adding n'_k successors in the k-th Venn region for which it holds that (†):

  - $\sigma^*(\mathcal{U}) \subseteq \sigma(\mathcal{U})$
  - for each $y_e \in \sigma^*(\mathcal{U})$:

$*$ if $y_e \in \sigma^*(X_n)$ then $y_e \in \sigma(X_n)$
$*$ if $y_e \notin \sigma^*(X_n)$ then $y_e \notin \sigma(X_n)$
with $n \in \mathbf{C} \cup \mathbf{R} \cup \{succ(c) | x : succ(c) \in \mathcal{A}_i\}$

The reason for that lays under the fact, that we decreases the number of successors in specific Venn-region to gain n'.
The algorithm then create individual names according to the solution, which leads to the satisfaction of all *succ*-assertions of x. For each created individual name y we know that there is an element $y_e \in \sigma^*(\mathcal{U})$ such that:

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

Because of the fact in (†) we can conclude

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

Since σ is induced by $\mathcal{I}$ we find for each newly introduced individual name y an element $y^{\mathcal{I}}$ such that:

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

Hence $\mathcal{I}$ satisfies $\mathcal{A}_{i+1}$

□

Bibliography

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

[2] 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.

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

[4] C. H. Papadimitriou. On the complexity of integer programming. *J. ACM*, 28(4):765–768, Oct. 1981.

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

[6] 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.