Contents

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. Both are in the same class. By adding solely those information in a traditional data base the information that Anna must teaches 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 relation 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. DL 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 formalize 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.
One big research field in DL is the determination of satisfiability of an *knowledge base*, which is formulated in DL. A knowledge base normaly consists of a *TBox*, which contains the axioms (rules), and of an *ABox* which contains assertions of certain elements (objects). 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. In [3] a *Tableau*-algorithm is presented for checking satisfiability for an ABox in the DL $\mathcal{ALCQ}$. A Tableau-algorithm applies *completion rules* to a given *set*(ABox) to decompose complex concepts and try satisfiying violated *statements*(assertions). If the set concludes something unsatisfiable (clash) then the whole set is unsatisfiable. If no more rules are applicable and the set is not unsatisfiable, then it is otherwise. 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 results in 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. 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 $succ(|r.C| \geq 1)$.
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 $Human \sqcap succ(|hasChildren| \geq 1) \sqsubseteq Parent$. 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 give another solution for the satisfiability problem with a Tableau-algorithm. 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. 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 variables according to it to our ABox.

2 Preliminaries

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

Definition 1 ($QFBAPA$)**.** Let T be a 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 are 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 set and σ a mapping which maps

- every symbol a in T to $\sigma(a)$
- $\emptyset$ 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 if it satisfies all cardinality constraints.
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 variables. The set $Var(\mathcal{A})$ is the set of variables 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 variable $x \in Var(\mathcal{A})$ 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 $(x^{\mathcal{I}}, y^{\mathcal{I}}) \in r^{\mathcal{I}}$

3 Tableau for $\mathcal{ALCSCC}$

A Tableau-algorithm consist 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 disjunctions. 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 ends 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 we want to 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 $\mathcal{I}$ of S we have $(C^{\neg})^{\mathcal{I}} = \mathcal{U} \backslash C^{\mathcal{I}}$ and $(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \backslash C^{\mathcal{I}}$. Since $\mathcal{U} \subseteq \Delta$ we can conclude that every element in $(C^{\neg})^{\mathcal{I}}$ 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, $C^{\neg} \to \neg C$ 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 the similarly to $\neg(C \sqcap D) \to \neg C \sqcup \neg D$ and $\neg(C \sqcap D) \to \neg C \sqcup \neg D$ and can also be applied in linear time.
Next we introduce *induced interpretation* with which we can count successors of variables after any rule application.

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

- for each variable $x \in Var(\mathcal{A})$ 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 $r^{\mathcal{I}(\mathcal{A})}$

Since we can now denote the number of successor of a variable x we can determine which assertion of the form $x : succ(c)$ are violated.

Definition 7 (Violated assertion)**.** Let $\mathcal{A}$ be a set of assertion, x be a variable, 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
- $\{x : succ(c)\} \subseteq \mathcal{A}$ 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 a variable 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_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 a variable has. The $QFBAPA$ formula for Example 1 is

$$\mathcal{F}(x) = 1 \leq |X_{|A|\leq|B\cap r|}| \wedge |X_A| \leq |X_B| \wedge \mathcal{U} = X_r$$

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 variables 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 variables 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 solution. However there can be infinite many solution. Actually Example 1 can have infinitetly many solution: We can always increase the amount of successors in B as long as we have fewer successors in A. However not all solutions are desired ones, e.g. we do not need a solution where we have a thousand successors in B if a hundred are also enough. Or we can add just r-successors, which are neither in A nor in B even though it satisfies none of the constraint. Therefore we want to only consider solution inside some *upper bounds*. The problem to bound the solution is transferable to the *Integer Linear Programming* (ILP) problem, for which upper bounds are investigated already (like in [4]). Therefore we want to transform our formula into a *system of equalities*:
We arrange the inequalities into a form of $X_1 \pm \cdots \pm X_n \leq I$, $I \in \mathbb{Z}$, first. That means e.g. $\mathcal{F}(x)$ is arrange into

$$\mathcal{F}(x) = -|X_{|A|\leq|B\cap r|}| \leq -1 \wedge |X_A| - |X_B| \leq 0 \wedge \mathcal{U} - X_r = 0$$

Then we change each inequalities into equalities:

$$\mathcal{F}(x) = -|X_{|A|\leq|B\cap r|}| + I_1 = -1 \wedge |X_A| - |X_B| + I_2 = 0 \wedge \mathcal{U} - X_r = 0$$

Now we can transform the numerical parts into a system $Ax = b$:

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

with the ordering $|X_{|A|\leq|B\cap r|}|, |X_A|, |X_B|, I_1, I_2$.
In [4] the calculated upper bound for each x_i is $n \cdot (m \cdot max_{i,j}\{|a_{ij}|, |b_i|\})^{2 \cdot m+1}$, where A is

a $m \times n$ matrix, b a vector of $\mathbb{Z}^m$ and $1 \leq i \leq m$, $1 \leq j \leq n$. some more text about that paper In our example the upper bound for all x_i is $5 \cdot (2 \cdot max_{i,j}\{|1|, |-1|\})^{2 \cdot 2+1} = 320$. For our example it means that we can have solutions in which we never have more then 320 successors in each $succ(|A| \leq |B \cap r|)$, A and B.

3.3 Algorithm

Finally we can present the Tableau-algorithm for an ABox in $\mathcal{ALCSCC}$. We describe above how we handle the numeric arithmetic of $\mathcal{ALCSCC}$ and that we want to decompose compound concept 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 and a solver returns a possible assignment of elements. 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 a variable 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 $\mathcal{F}(S)$

  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 and for each $e \in E(\sigma)$ we introduce a new variable y and
  - for each $e \in X_C$ we have $\mathcal{A} = \mathcal{A} \cup \{y : C\}$
  - for each $e \in X_c$, c is a cardinality constraint, we have $\mathcal{A} = \mathcal{A} \cup \{y : succ(c)\}$
  - for each $e \in X_r$, $r \in \mathbf{R}$, we have $\mathcal{A} = \mathcal{A} \cup \{(x, y) : r\}$

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 $1 \leq X_{|A|\leq|B\cap r|} \wedge |X_A| \leq |X_B| \wedge \mathcal{U} = \{X_r\}$. Then we let a solver returns an assignment if possible. In this case we see that the formula is satisfiable with $X_{|A|\leq|B\cap r|} = \{f\}, X_A = \{\}, X_B = \{e\}$ and $X_r = \{e, f\}$. Since we have $\mathcal{U} = \{X_r\}$ every element must be in X_r. That means that every successor is connected to its predecessor by a role name and in our example every successor is a r-successor. We then introduce for e ans f two variables 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.

4 Correctness

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

- 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 every input the Tableau-algorithm terminates

In all proves we assume that the $QFBAPA$ solver is correct. First we prove that the algorithm works correctly e.g. we prove the first two points.

Lemma 1. 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 a variable 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 a variable 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 variables 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 $e \in E(\sigma)$ and hence also for each freshly introduced variable 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 $x^{\mathcal{I}(\mathcal{A}')} \in \Delta 1^{\mathcal{I}(\mathcal{A}')}$. Because we introduced e in this step, we also introduced a new variable 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 2. If $\mathcal{A} := \{x : C\}$ is satisfiable then the Tableau-algorithm terminates without a clash.

Proof. 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 $x : succ(c)$: By the induction hypothesis we have $x^{\mathcal{I}} \in succ(c)^{\mathcal{I}}$. 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, formulate a $QFBAPA$ formula $\mathcal{F}(x)$ and let a solver return us all possible solution with in an upper bound. Because $\mathcal{A}_i$ is satisfiable, the *succ*-assertions together are 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 σ with $E(\sigma) = \emptyset$ 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 reduce the solution induced by $\mathcal{I}$. todo. Therefore we are able to pick a solution, such that $\mathcal{I}$ satisfies $\mathcal{A}_{i+1}$.

□

We know that a subset of a satisfiable ABox $\mathcal{A}$ is also satisfiable because the induced interpretation $\mathcal{I}(\mathcal{A})$, which satisfies $\mathcal{A}$, satisfies all subsets of $\mathcal{A}$.
We now prove that the algorithm terminates. For that we define first the depth and size? of a concept C:

$$depth(C) := \begin{cases} 1 + depth(c) & C = succ(c) \\ 0 & \text{otherwise} \end{cases}$$

This function is well-define because by Definition 3 our concepts are inductively defined from concept names, hence the *depth*-function always returns an integer.

Lemma 3. The Tableau-algorithm always terminate for an ABox $\mathcal{A} = \{x : C\}$

Proof. ??? □

References

[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.