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

In this work $\mathbf{C}$ denotes a set of concept names and $\mathbf{R}$ a set of role names, which are disjoint. 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 $n \in \mathbb{N}$

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 $n \, dvd \, l$.

Definition 2 ($QFBAPA$ formula)**.** A $QFBAPA$ formula with the restriction from above has the following grammar (see [4])

$$
\begin{aligned}
F \rightarrow &\quad C \mid F \wedge F \mid F \vee F \mid \neg F \\
C \rightarrow &\quad K \leq K \mid N \, dvd \, K \\
K \rightarrow &\quad N \mid K + K \mid N \cdot K \mid |S| \\
S \rightarrow &\quad \emptyset \mid \mathcal{U} \mid S \cap S \mid S \cup S \mid S^{\neg} \mid x \\
N \rightarrow &\quad 0 \mid 1 \mid \ldots
\end{aligned}
$$

Definition 3 ($\mathcal{ALCSCC}$)**.** $\mathcal{ALCSCC}$ concepts are defined inductively:

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

– $C \sqcap D$

An ABox S in $\mathcal{ALCSCC}$ is a finite set of assertions of the form $x : C$ and $(x, y) : s$, where C is a $\mathcal{ALSCSS}$ concept, s a set term and x, y variables. The set $Var(S)$ is the set of variables occurring in S.

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

- $\emptyset$ to $\emptyset^{\mathcal{I}}$
- $\mathcal{U}$ to $\mathcal{U}^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
- each variable $x \in Var(S)$ 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 $\bot^{\mathcal{I}} = \emptyset^{\mathcal{I}}$
- $(C \sqcap D)^{\mathcal{I}} := C^{\mathcal{I}} \cap D^{\mathcal{I}}$, $(C \sqcup D)^{\mathcal{I}} := C^{\mathcal{I}} \cup D^{\mathcal{I}}$
- $(\neg C)^{\mathcal{I}} := \Delta^{\mathcal{I}} \backslash C^{\mathcal{I}}$
- $(s \cap t)^{\mathcal{I}} := s^{\mathcal{I}} \cap t^{\mathcal{I}}$, $(s \cup t)^{\mathcal{I}} := s^{\mathcal{I}} \cup t^{\mathcal{I}}$
- $(s^{\neg})^{\mathcal{I}} := \mathcal{U}^{\mathcal{I}} \backslash s^{\mathcal{I}}$
- $|s|^{\mathcal{I}} := |s^{\mathcal{I}}|$
- $(k + l)^{\mathcal{I}} := (k^{\mathcal{I}} + l^{\mathcal{I}})$, $(n \cdot k)^{\mathcal{I}} := n \cdot k^{\mathcal{I}}$
- $succ(c)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} | \text{the mapping } \cdot^{\mathcal{I}_x} \text{ satisfies } c\}$

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 $r^{\mathcal{I}_x} := r^{\mathcal{I}}(x)$.
The mappings satisfies for the cardinality terms k, l

- $k \leq l$ iff $k^{\mathcal{I}} \leq l^{\mathcal{I}}$
- $n \, dvd \, l$ iff $\exists m \in \mathbb{N} : n \cdot m = l^{\mathcal{I}}$

The *assignment* $\pi_{\mathcal{I}} : Var(S) \to \Delta^{\mathcal{I}}$ satisfies

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

$\pi_{\mathcal{I}}$ satisfies an ABox S if $\pi_{\mathcal{I}}$ satisfies every assertion in S. If $\pi_{\mathcal{I}}$ satisfies S then $\mathcal{I}$ is a model of S.

We say a set term t occurs semantically in a cardinality term $k = n_0 + n_1 \cdot |s_1| + \cdots + n_j \cdot |s_j|$ if for a $i \in \{1, \ldots, j\}$ we have $s_i = t \cup t_{rest}$, where t_{rest} is a set term. Note that a set term t does not occurs semantically in the cardinality term$|t^{\neg}|$.
For the Tableau-algorithm we can apply rules for decomposing concepts, introducing variables, adding assertion and merging variables.

Definition 5 (Merge)**.** *Merging* two variables y_1 and y_2 in an ABox S results in one variable y: replace all occurrence of y_1 and y_2 with y in S.

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.
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 6 (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 7 (Induced Interpretation)**.** An interpretation $\mathcal{I}(S)$ can be induced from an ABox S by the following steps:

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

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 8 (Violated assertion)**.** Let S be a set of assertion, x be a variable, k be a cardinality term and $n \in \mathbb{N}$. An assertion is *violated* if

- $x : succ(n \leq l)$ and $n \not\leq l^{\mathcal{I}(S)_x}$
- $x : succ(k \leq n)$ and $k^{\mathcal{I}(S)_x} \not\leq n$
- $x : succ(k \leq l)$ and $k^{\mathcal{I}(S)_x} \not\leq l^{\mathcal{I}(S)_x}$
- $x : succ(n \, dvd \, k)$ and $mod(k^{\mathcal{I}(S)_x}, n) \neq 0$

where $n \in \mathbb{N}$.

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

Definition 9 (Clash)**.** An ABox S contains a *clash* if

- $\{x : \bot\} \subseteq S$ or
- $\{x : A, \, x : \neg A\} \subseteq S$ or
- $\{x : succ(n \, \neg dvd \, l), x : succ(m \, dvd \, l)\} \subseteq S$, with $n \, dvd \, m$ or $m \, dvd \, n$, or
- $\{(x, y) : s, \, (x, y) : s^\neg\} \subseteq S$ or
- $\{x : succ(c)\} \subseteq S$ violated and no more rules are applicable

3.1 The form of the ABox

Like already mention a ABox S in $\mathcal{ALCSCC}$ contains assertions of the form $x : C$ and $(x, y) : C$ where C is a $\mathcal{ALCSCC}$ concept and $x, y \in Var(S)$ are variables. Like already discussed we want constraints to be in NNF and for an assertion $x : succ(c)$ that c is either $k \leq l$ or $n\, dvd\, l$. Our desired form for the ABox is gained by the following algorithm.

Algorithm 1 Transforming ABox

ABox S
for each assertion $x : succ(c) \in S$ **do**
  if c is $s_1 \subseteq s_2$ **then**
    $c := |s_1 \cap s_2^{\neg}| \leq 0$; return;
  end if
  if c is $s_1 \supseteq s_2$ and a set term constraint **then**
    $c := |s_1^{\neg} \cap s_2| \leq 0$; return;
  end if
  if c is $s_1 = s_2$ **then**
    remove $x : succ(c)$;
    add $x : succ(|s_1 \cap s_2^{\neg}| \leq 0)$ and $x : succ(|s_1^{\neg} \cap s_2| \leq 0)$; return;
  end if
  if c is $l > k$ **then**
    $c := k + 1 \leq l$; return;
  end if
  if c is $k < l$ **then**
    $c := k + 1 \leq l$; return;
  end if
  if c is $l \geq k$ **then**
    $c := k \leq l$; return;
  end if
  if c is $l = k$ and a cardinality constraint **then**
    remove $x : succ(c)$;
    add $x : succ(l \leq k)$ and $x : succ(k \leq l)$; return;
  end if
end for

for each assertion $x : C$ or $(x, y) : s$ **do**
  $C := NNF(C)$ or $s := NNF(s)$;
end for

The first part of the transformation runs in worst case in polynomial time: If any of the first four conditions holds, we only replace c which runs in constant time. The complexity of a removing function depends on the implementation: If we have direct access to the elements in the ABox the remove function has a constant (runtime) complexity. If we

have some kind of sorting then the (runtime) complexity is logarithm. In worst case if we the ABox is stored as a simple list, then the (runtime) complexity is linear. Hence in worse cast the first part runs in polynomial time. The second part always runs in polynomial time because each transformation into NNF runs in linear time. Hence our desired form of the ABox can be obtained in a polynomial time.

3.2 Transforming an ABox into a formula

Dealing with numerical arithmetic is challenging and hence we use the help of a $QFBAPA$ s. We want to use it 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*. As example we look at

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

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

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. 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 $succ(|A| \leq |B|)$. The next step would be to introduce variables and the assignments according to the solution the solver gives us. Hence we also introduced variables for the nested *succ*-assertion. Then we can do the procedure from above again for the new variables.
If x has already successors in S we have to apply the following steps additionally.
For each successor y introduce a new element e:

- for each $(x, y) : r$, $r \in \mathbf{R}$: add e to X_r
- for each $y : C$, $C \in \mathbf{C}$ add e to X_C
- for each $y : succ(c)$ add e to X_c

The $QFBAPA$ formula for Example 1 is

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

with $e \in X_B \cap X_r$.

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 10 (Tableau)**.** Let S be a set of assertions in simplified NNF.
Boolean part:

- $\sqcap$-rule: S contains $x : C_1 \sqcap C_2$ but not both $x : C_1$ and $x : C_2$
  $\rightarrow S := S \cup \{x : C_1, x : C_2\}$
- $\sqcup$-rule: S contains $x : C_1 \sqcup C_2$ but neither $x : C_1$ nor $x : C_2$
  $\rightarrow S := S \cup \{x : C_1\}$ or $S := S \cup \{x : C_2\}$
- *set*-rule: S contains $(x, y) : s$ and
  - $s = s_1 \cup s_2$ and either $(x, y) : s_1 \notin S$ or $(x, y) : s_2 \notin S \rightarrow S := S \cup \{(x, y) : s_1, (x, y) : s_2\}$
  - $s = s_1 \cup s_2$ and neither $(x, y) : s_1 \in S$ nor $(x, y) : s_2 \in S \rightarrow S := S \cup \{(x, y) : s_1\}$ or $S := S \cup \{(x, y) : s_2\}$
  - $s = C$, C is a $\mathcal{ALCSCC}$ concept $\rightarrow S := S\{y : C\}$

Numerical part:

- *successor*-rule: S contains for an unmarked variable x at least one assertion of the form $x : succ(c)$:
  - gather all assertion of the form $x : succ(c)$ into a set $\mathcal{A}$
  - transform $\mathcal{A}$ into a $QFBAPA$ formula

  If a $QFBAPA$ solver returns *unsatisfiable* then $S = S \cup \{x : \bot\}$
  If a $QFBAPA$ solver returns *satisfiable* then for each new element e in the solution we introduce a new variable y and
  - for each $e \in X_C$ we have $S = S \cup \{y : C\}$
  - for each $e \in X_c$, c is a cardinality constraint, we have $S = S \cup \{y : succ(c)\}$
  - for each $e \in X_r$, $r \in \mathbf{R}$, we have $S = S \cup \{(x, y) : r\}$
  - mark x

and for each old element e we add the remaining assertion according to above to S

We pick up Example 1 again but in a form where it is not decomposed yet:

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

We are able to apply the boolean rules and hence we apply them first and decompose into Example 1. It is also possible to start with the *set*rule and decompose $(x, y) : B \cap r$. When no more rules from the boolean part are applicable we have

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

Then we apply the *successor*-rule: We collect every *succ*-assertion regarding x to a set $\mathcal{A} := \{x : succ(1 \leq |succ(|A| \leq |B \cap r|)|), x : succ(|A| \leq |B|)\}$ and convert $\mathcal{A}$ to the $QFBAPA$ formula $1 \leq X_{|A|\leq|B\cap r|} \wedge |X_A| \leq |X_B| \wedge \mathcal{U} = \{X_r\}$. Since x also have the successor y we have to introduce an element e with $e \in X_B \cap X_r$. Then we let a solver gives as a possible assignment if possible. In this case we see that the formula is satisfiable 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 add the assertion $y : X_B$ and $(x, y) : r$ and introduce for f a new variable z and add $z : succ(|A| \leq |B \cap r|)$ and $(x, z) : r$ to S. Then for z we apply the *successor*-rule again.

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 S then S is satisfiable
- If S is satisfiable then the Tableau-algorithm terminates without a clash

First we prove that the tableau algorithm terminates.

Proposition 1. Let C be a concept in simplified $CNNF$. Then there is no infinite chain of applications of any tableau rules issuing from $\{x : C\}$.

Definition 11 (Derived Set)**.** A *derived set* is an ABox S' where rule **??** is not applicable.

In order words a derived set is an ABox on which we applied a rule completely e.g. every time we add a new assertion $(x, y) : s$ we add all assertion concluded by it to S (rule **??**) first.
To prove this we map any derived set S to an element $\Psi(S)$ from a set Q. We then show that the elements in Q can be ordered by a well-founded relation $\prec$. A well-founded

relation says that there is no infinite decreasing chain. If we can show that by obtaining a derived set S' from another set S we have $\Psi(S') \prec \Psi(S)$ then the algorithm terminates. The elements in Q are finite multisets of septuples and the elements of the septuples are either integers or mutlisets of integers. For two septuples $q = (q_1, \ldots, q_7)$ and $q' = (q'_1, \ldots, q'_7)$ it holds $q \prec q'$ if for the first i, $1 \leq i \leq 7$, for which q_i and q'_i differs it holds that $q_i \prec q'_i$ (also called lexicographical ordering). For two mutlisets of integers q_i and q'_i it holds $q'_i \prec q_i$ if q'_i can be obtained from q_i by replacing an integer c in q'_i by a finite number of integers which are all smaller than c. The relation $\prec$ for those multisets is well-founded because we work with integers. That means from a multiset $\{0, \ldots, 0\}$ the smallest multiset which can be obtain is $\{\}$ by removing all 0s.
For a concept C its size $size(C)$ is inductively defined as

- 0, if C is $\bot$
- 1, if C is a concept name of $\mathbf{C}$
- $size(\neg C) = 1 + size(C)$
- $size(succ(c)) = 1 + \sum_{C \text{ in c}} size(C)$
- $size(C \sqcap D) = size(C \sqcup D) = size(C) + size(D)$

The asymmetrical difference of two numbers n, m is denoted by

$$n \trianglelefteq m \begin{cases} n - m & \text{if } n > m \\ 0 & \text{if } n \leq m \end{cases}$$

The septuples in Q are defined as follows

Definition 12. Let S be an ABox. The multiset $\Psi(S)$ consist of septuples $\psi_S(x)$ for each variable x. The component of the septuples are structured as follows

- the first component is a non-negative integer $max\{size(C) \mid x : C \in S\}$
- the second component is a multiset of integers containing for each $x : C \sqcap D$, on which the $\sqcap$-rule is applicable, the non-negative integer $size(C \sqcap D)$ (respectively for $C \sqcup D$)
- the third component is the number of x's successors on which rule **??** is applicable
- the fourth component is a multiset of integers containing for each $x : succ(k \leq n) \in S$ the number of all successors y of x on which rule **??** is applicable
- the fifth component is a multiset which denotes for every $x : succ(k \leq l)$ and $x : succ(n \, dvd \, l)$ the integer $k^{\mathcal{I}(S)_x} \trianglelefteq l^{\mathcal{I}(S)_x}$ and $n \trianglelefteq l^{\mathcal{I}(S)_x}$ respectively
- the sixth component denotes the number of all successors of x.
- the seventh component is a multiset which denotes for each $x : succ(k \leq 0)$ the number of successors on which the rule **??** is applicable

Lemma 1. The following properties hold

1. For any concept C we have $size(C) \geq size(NNF(\neg C))$
2. Any variable y in a derived set S has at most one predecessor x in S
3. If $(x, y) : r \in S$ for a $r \in \mathbf{R}$ (and y is a introduced variable) then

$$max\{size(C) \mid x : C \in S\} > max\{size(D) \mid y : D \in S\}$$

Proof.

1. By induction over the number of applications to compute the negation normal form we have $size(C) = size(NNF(\neg C))$. Because $\neg succ(0 \leq k)$ can be replace by $\bot$ which is *smaller*, we have $size(C) \geq size(NNF(\neg C))$. This can be done because $\neg succ(0 \leq k) = succ(k < 0)$ which is impossible to satisfy and therefore $\neg succ(0 \leq k) = \bot$.
2. If y is a newly introduced variable, then it can only be introduced by exactly one variable x which is y's only predecessor. If two variables are merged together by rule **??** then both variables must have the same predecessor x by the condition of that rule.
3. By the second fact we know that x is the only predecessor of y. When y is introduced by applying **??** on a assertion $x : succ(k \leq l)$ then we have $y : C$ for every concept C occurring in l (for $\neg C$ we have $y : \neg C$). We know that $size(succ(k \leq l))$ is greater then $size(C)$ therefore Lemma 1.3 holds. A new assertion $y : D$ can occur either because rule 10 or 10 are applicable on $y : C$ with $C = D \sqcap D'$ or $C = D \sqcup D'$, which neither raise $max\{size(D) \mid y : D \in S\}$, or because rule **??** is applicable but that also does not raise $max\{size(D) \mid y : D \in S\}$: If rule **??** is applicable on $x : succ(k \leq l)$ then for every added assertion $y : D$ the concept D must occur in k and therefore $size(succ(k \leq l)) > size(D)$. If y gets merged together with another variable z, then y and z must have the same predecessor which means that all concept sizes regarding z are also smaller then $max\{size(C) \mid x : C \in S\}$.

□

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] V. Kuncak and M. Rinard. Towards efficient satisfiability checking for boolean algebra with presburger arithmetic. 07 2007.

[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.