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 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 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 relation between the concepts, which correspond to binary relations in FOL. 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 DL as an *axiom* $Men \sqcup Women \sqsubseteq Human$ and in FOL as $\forall x.Men(x) \vee Women(x) \rightarrow Human(x)$. The statement "All Human, who has children, are parents" in DL can be formalized a $Human \sqcap \exists hasChildren.\top \sqsubseteq Parent$ and in FOL as $\forall x \exists y.Human(x) \wedge hasChildren(x, y) \rightarrow Parent(x)$. 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 *Reasoning* which is the investigation of whether certain information can be concluded from the current data or not.
to be continued...

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 $n \in \mathbb{N}$

For readability we use $\lesseqgtr$ to address the comparison symbols $=, \leq, \geq, <, >$. The negation $\not\lesseqgtr$ address the symbols $\neq, >, <, \geq, \leq$ respectively.
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 $x : succ(|s \cap t^{\neg}| \leq 0) \sqcap succ(|s^{\neg} \cap t| \leq 0)$.

Definition 2 ($\mathcal{ALCSCC}$)**.** $\mathcal{ALCSCC}$ 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 3 (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

- $k \lesseqqgtr l$ iff $k^{\mathcal{I}} \lesseqqgtr l^{\mathcal{I}}$
- $n \, dvd \, l$ 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.

3 Tableau

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 4 (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.

- $\neg\top \rightarrow \bot$
- $\neg\bot \rightarrow \top$
- $\neg\neg C \rightarrow C$
- $\neg(C \sqcap D) \rightarrow \neg C \sqcup \neg D$
- $\neg(C \sqcup D) \rightarrow \neg C \sqcap \neg D$
- $C^{\neg} \rightarrow \neg C$
- $\neg succ(c) \rightarrow succ(\neg c)$
- $\neg(k \lesseqqgtr l) \rightarrow k \not\lesseqqgtr l$
- $\neg(k \not\lesseqqgtr l) \rightarrow k \lesseqqgtr l$
- $\neg(n\ dvd\ k) \rightarrow n\ \neg dvd\ k$
- $\neg(n\ \neg dvd\ k) \rightarrow n\ dvd\ k$
- $(s \cap t)^{\neg} \rightarrow s^{\neg} \cup t^{\neg}$
- $(s \cup t)^{\neg} \rightarrow s^{\neg} \cap t^{\neg}$
- $(s^{\neg})^{\neg} \rightarrow s$

The rule $C^{\neg} \rightarrow \neg C$ 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,$C^{\neg} \rightarrow \neg C$ and $\neg succ(c) \rightarrow succ(\neg c)$ can also be applied in linear time since we only shift the negation sign. the rule $(s^{\neg})^{\neg} \rightarrow s$ works similarly to $\neg\neg C \rightarrow C$ and can be applied in linear time. The rules $(s \cap t)^{\neg} \rightarrow s^{\neg} \cup t^{\neg}$ and $(s \cup t)^{\neg} \rightarrow s^{\neg} \cap t^{\neg}$ works the similarly to $\neg(C \sqcap D) \rightarrow \neg C \sqcup \neg D$ and $\neg(C \sqcap D) \rightarrow \neg C \sqcup \neg D$ 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 5 (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 6 (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

- $x : succ(k \lesseqgtr n)$ and $k^{\mathcal{I}(S)_x} \not\lesseqgtr n$
- $x : succ(k \lesseqgtr l)$ and $k^{\mathcal{I}(S)_x} \not\lesseqgtr l^{\mathcal{I}(S)_x}$
- $x : succ(n\, dvd\, k)$ 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. A clash in a ABox S implies that $\bot$ can be derived from S.

Definition 7 (Clash)**.** An ABox S contains a *clash* if

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

Also important for the algorithm is to consider the *signs* of concept names and role names.

Definition 8 (Positive and Negative Sign)**.** Let $(x, y) : s$ be an arbitrary assertion with $x, y \in Var(S)$ and s being a set term in NNF. A concept name C has a *positive sign* in s if no negation sign appears immediately in front of C. It has a *negative sign* otherwise. A role name r has a *positive sign* if no negation sign appears above it. It has a *negative sign* otherwise.

All concept names in **C** and role names in **R** have a positive sign.

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. We already define that all concepts are in NNF. To enhance the readability in this work we write $k \leq l$ instead of $l \geq k$ and $k < l$ instead of $l > k$. Therefore $k \lesseqgtr l$ can only represent $k \leq l$, $k = l$ or $k < l$ from now on. Also if we have $k \leq l$ and $l \geq k$ then we replace it with $k = l$.
Therefore c in an assertion $x : succ(c)$ can have only one of the following form:

- $k < l$
- $k \leq l$
- $k = l$
- $n\, dvd\, l$

where k, l are in NNF.
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 $c = s_1 \subseteq s_2$ **then**
    $c := (|s_1 \cap s_2^{\neg}| \leq 0)$; return;
  end if
  if $c = s_1 \supseteq s_2$ **then**
    $c := (|s_1^{\neg} \cap s_2| \leq 0)$; return;
  end if
  if $c = l > k$ **then**
    $c := k < l$; return;
  end if
  if $c = l \geq k$ **then**
    for each assertion $x : succ(d)$ **do**
      if $d = l \leq k$ or $d = k \geq l$ **then**
        remove $x : succ(c)$ and $x : succ(d)$ from S;
        add $x : (k = l)$ to S; return;
      else
        $c := k \leq l$; return;
      end if
    end for
  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 three conditions holds, we only replace c which runs in constant time. If the fourth conditions holds, we go through the ABox again, which leads to a polynomial time in total. In cases of an ABox where the fourth conditions never holds the first part runs in linear time. The second part always runs in polynomial time because each transformation into NNF runs in linear time.

3.2 Restrictions

The idea of a Tableau-algorithm is rather simple: We apply rules in a arbitrary order whenever they are applicable. However often some caution has to be made. In [3] the considered DL is $\mathcal{ALCQ}$ which allows qualified number restrictions. The algorithm uses a rule for replacing variables but to prevent endless loops of rule application, which can occur due to the replication, a safeness condition ia added. In [4] and[5] the considered DL allows inverse roles which is handled with blocking techniques. A similar blocking technique is used in [1] also to prevent non-termination due to considered cyclic TBoxes. The DL $\mathcal{ALCSCC}$ is a very expressive concept language and hence there are some difficulty to handle for the Tableau-algorithm. We also restrict ourself to solely ABoxes.
We loose a bit the property of a Tableau-algorithm that rules can be applied in any order: In case we add $(x, y) : s$ to our ABox and s is a (always finite) chain of disjunction and conjunction we want to add all assertions of y first before applying any other rules. This way y has all assertion we want to assign to it and hence avoid adding unnecessary variables which can lead into a violation of other assertions.
Similar to [3] and [5], where a variable can be replaced by another variable, we can merge two variables during the Tableau-algorithm like in [4].

Definition 9 (Merge)**.** *Merging* y_1 *and* y_2 *results in one variable* y*: replace all occurrence of* y_1 *and* y_2 *with* y*.*

By merging variables assertion may become violated. To ensure the termination of the algorithm we intuitively want to avoid violating any assertions especially when they are satisfied. However there are cases where a violation is unavoidable. We look at the following example:

Example 1.

$$\begin{aligned} S = \{&x : succ(|t| + |A \cup B| \geq 4),\, x : succ(|A \cup B| \leq 1) \\ &y_1 : A,\, y_2 : B,\, (x, y_1) : t,\, (x, y_2) : t\} \end{aligned}$$

We see that the assertion $x : succ(|A \cup B| \leq 1)$ is violated and a solution is to merge y_1 and y_2. This leads to $x : succ(|t| + |A \cup B| \geq 4)$ being violated. We can fix it by adding new successors. We can easily detect that adding a t-successor leads to a satisfied ABox. On the other hand a successor in $A \cup B$ leads to the initial state where $x : succ(|A \cup B| \leq 1)$ is violated. Hence the algorithm can run into a endless loop of adding and merging variables. Therefore we introduce a notion of *blocking*. If we merged two variable y_1 and y_2 into y because of a violated assertion $x : succ(k \lesseqgtr l)$,

$k = n_0 + n_1 \cdot |s_1| + \cdots + n_j \cdot |s_j|$, then we want to *block* any introduction of an assertion of the form $(x, z) : s_1 \cap \cdots \cap s_i$, $1 \leq i \leq j$. This way we want to avoid a possible re-violation of $x : succ(k \lesseqgtr l)$ and a possible unless loop of merging and introducing variables. For that we introduce for a variable x a blocking set $b(S, x)$ in which set terms of k are listed.

Definition 10 (The blocking set $b(S, x)$)**.** Let y_1 and y_2 be successors of a variable x. If y_1 and y_2 are merged into y because of an assertion $x : succ(k \lesseqgtr l)$ then $b(S, x) := b(S, x) \cup \{s_i | k = n_0 + n_1 \cdot |s_1| + \cdots + n_j \cdot |n_j|, \forall i : 1 \leq i \leq j\}$. A set term $u = t_1 \cap \cdots \cap t_j$ is *blocked* by x in the ABox S if $\forall i : 1 \leq i \leq j$, $t_i \cup u \in b(S, x)$, where u is a arbitrary set term. At the beginning of the Tableau-algorithm all sets $b(S, x)$, $x \in Var(S)$, are empty.

We say $t_i \cup u$ because in case there is a set term with disjunction $s_1 \cup s_2$ in $b(S, x)$ then it is enough to add either $(x, y) : s_1$ or $(x, y) : s_2$ with y being a new variable to possible violate a satisfied assertion.
If it is clear of which ABox we talk about, we only say "u is blocked by x".
In our example after the merging the set term $A \cup B$ is added to $b(S, x)$ and hence this set term is blocked by x, which means that to satisfy $x : succ(|t| + |A \cup B| \geq 4)$ we can only add a t-successor. This blocking set is then used to determine whether a variable can be introduced or not. Beside this set we also need to consider one more thing: Intuitively we for a assertion $x : succ(k \lesseqgtr l$ we want to add variables, such that l increases. However if both k and l can increase by a new variable, we have to avoid cases where k increases faster then l. The increment depends on the non-blocked set term u e.g. can not be formed from set terms in $b(S, x)$, for which we want to introduce a new variable y and add $(x, y) : u$ to S. To determine whether u is *safe* we count how often u "appears" in l and k. If it appears more often in l than in k then it is safe.

Definition 11 (Safe)**.** Let $x : succ(k \lesseqgtr l)$ be an assertion in S. Let $u = t_1 \cap \cdots \cap t_j$ with $1 \leq i \leq j$. If $n_k(u) < n_l(u)$ and u is not blocked by x then u is called *safe regarding* $k \leq l$. The number $n_k(u)$ (and $n_l(u)$ respectively) is computed as followed:

Algorithm 2 Compute $n_k(u)$

$n_k(u) := 0$
$k = n_0 + n_1 \cdot |s_1| + \cdots + n_j \cdot |s_j|$
$u = t_1 \cap \cdots \cap t_o$
for each $1 \leq i \leq j : n_i \cdot |s_i|$, $s_i = s'_1 \cup \cdots \cup s'_p$, $p \in \mathbb{N}$ **do**
  if $\exists q, 1 \leq q \leq p : u = s'_q \cap t'$ **then**
    $n_k(u) := n_k(u) + n_i$
  end if
end for
return $n_k(u)$

This says that it is only safe to add a variable if l increases faster then k. As example we look at

Example 2.

$$S = \{x : succ(|r \cup s| < |r| + |s|)\}$$

The set terms r and s are not blocked but still not safe because $n_{|r \cup s|}(r) = n_{|r|+|s|}(r) = 1$ and $n_{|r \cup s|}(s) = n_{|r|+|s|}(s) = 1$. However the set term $r \cap t$ is safe because $r \cap t \notin b(S, x) = \emptyset$ and $1 = n_{|r \cup s|}(r \cap s) < n_{|r|+|s|}(r \cap s) = 2$.

3.3 Algorithm

Finally we can present the Tableau-algorithm to determine the satisfiability of an ABox in $\mathcal{ALCSCC}$.

Definition 12 (Tableau)**.** Let S be a set of assertions in simplified NNF.

1. $\sqcap$-rule: S contains $x : C_1 \sqcap C_2$ but not both $x : C_1$ and $x : C_2$
   $\rightarrow S := S \cup \{x : C_1, x : C_2\}$
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 $S := S \cup \{x : C_2\}$
3. *choose*-rule: S contains
   - $x : succ(k \lesseqgtr l)$
   - $(x, y) : k'$, $k = n \cdot |k' \cup u_1| + m \cdot |k' \cap u_2| + u_3$, $n, m \in \mathbb{N}_0$, u_1, u_2, u_3 are set terms
   - but not $(x, y) : k$

   $\rightarrow$ either $S := S \cup \{(x, y) : k\}$ or $S := S \cup \{(x, y) : k^\neg\}$. Then jump to rule 9
4. *choose-a-role*-rule: S contains $(x, y) : s$ but for any $r \in \mathbf{R}$: $(x, y) : r \notin S$
   $\rightarrow$ choose $r \in \mathbf{R}$, such that $(x, y) : r^\neg \notin S$. $S := S \cup \{(x, y) : r\}$. Then jump to rule 9
5. *divide*-rule: S contains $x : succ(n\, dvd\, l)$, $l = n_1 \cdot |s_1| + \cdots + n_i \cdot |s_i| + \cdots + n_j \cdot |s_j|$, which is violated
   $\rightarrow$ introduce a new variable y, choose $s = s_1 \cap \cdots \cap s_i$, $1 \leq i \leq j$ and $S := S \cup \{(x, y) : s\}$. Then jump to rule 9
6. $\leq$-rule: S contains
   - $x : succ(k \lesseqgtr l)$, which is violated
   - there is a set term $s := |s_1 \cap \cdots \cap s_i|$, $l = n_1 \cdot |s_1| + \cdots + n_i \cdot |s_i| + \cdots + n_j \cdot |s_j|$, which is safe regarding $k \lesseqgtr l$

   $\rightarrow S := S \cup \{(x, y) : s\}$. Then jump to rule 9
7. *merge*-rule: S contains
   - $x : succ(k \lesseqgtr l)$, which is violated

- $(x, y_1) : s_1$ and $(x, y_2) : s_2$, such that $y_1 \neq y_2$ and $k = n \cdot |s_1 \cup s_2| + u$, where u is a cardinality term

$\rightarrow$ merge y_1 and y_2

8. ≤ 0-rule: S contains
   - $x : succ(|s_1 \cap \cdots \cap s_j| \leq 0)$
   - $(x, y) : s_1, \ldots, (x : y) : s_i$, $1 \leq i < j$
   - but not $(x, y) : s_{i+1}, \ldots, (x, y) : s_j$

   $\rightarrow$ choose $n \in \{i+1, \ldots, j\}$, extend $S := S \cup \{(x, y) : s_n^{\neg}\}$ and then jump to rule 9

9. *set.term*-rule (Repeat until inapplicable): In S is $(x, y) : s$ and
   a) $s = s_1 \cap s_2$ but $\{(x, y) : s_1, (x, y) : s_2\} \not\subseteq S$
   $\rightarrow S := S \cup \{(x, y) : s_1, (x, y) : s_2\}$
   b) $s = s_1 \cup s_2$ and neither $\{(x, y) : s_1\} \subseteq S$ nor $S\{(x, y) : s_2\} \subset S$
   $\rightarrow$ either $S := S \cup \{(x, y) : s_1\}$ or $S := S \cup \{(x, y) : s_2\}$
   c) $s = C$ and $y : C \notin S$, where C is an $\mathcal{ALCSCC}$ concepts
   $\rightarrow S := S \cup \{y : C\}$

Definition 13 (Derived Set)**.** A *derived set* is an ABox S' where rule 9 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 first.
We now explain the rules of the Tableau-algorithm and their intention, if not already mention in Section 3.2.
The first rule decompose the conjunction and the second rule adds non-deterministically the right assertion.
The *choose*-rule is important because we need to know of every successor what kind of role successors they are and in which concepts they are. For an assertion $x : succ(k \lesseqgtr l$ it is important that $k^{\mathcal{I}(S)_x}$ and $l^{\mathcal{I}(S)_x}$ counts the successors correctly. In the following case the successor y is not counted in $l^{\mathcal{I}(S)_x}$ while $x : succ(k \leq l)$ is violated.

Example 3.

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

There might be an model $\mathcal{I}'$ where y is also a s-successor of x and hence $l^{\mathcal{I}(S)_x} < l^{\mathcal{I}'(S)_x}$. However the Tableau-algorithm should be able to construct every model of S if S is consistent. Therefore this rule adds non-deterministically either $(x, y) : s$ or $(x, y) : s^{\neg}$ which are the only two possibilities. This way we are also able to construct $\mathcal{I}'$.
The *choose-a-role*-rule is necessary because for a assertion $x : succ(c)$ we might have no role name with a positive sign in c. Which means we know x must have some successors but we can not decide which role-successor it is. As example we have

Example 4.

$$\mathbf{R} = \{r, s\}$$
$$S = \{x : succ(|r^{\neg}| \geq 1)\}$$

It states that x have at least one successor which is not a r-successor. Since $\mathbf{R}$ only contains r and s we know that the successors must be s-successors. First we apply rule 6 to actually add a successor. Therefore y is introduced and $(x, y) : r^{\neg}$ is added to S. Now no more rules are applicable except for the *choose-a-role*-rule. With that rule we can pick either r or s. We can not pick r because $r^{\neg}$ occurs in the assertion. Therefore we have to pick s. Another more simple but not so significant example is

Example 5.

$$\mathbf{R} = \{r, s\}$$
$$S = \{x : succ(|A| \geq 1)\}$$

We know that x must have a successor in A but we still need to assign a role. In this case we can choose between r and s.

The *divide*-rule is straightforward: We choose one set term $s = s_1 \cap \cdots \cap s_i$ such that $l = n_1 \cdot |s_1| + \cdots + n_i \cdot |s_i| + \cdots + n_j \cdot |s_j|$ and introduce a new variable y and add $(x, y) : s$ to S. For any $x : succ(n\, dvd\, l)$ we know that the chain of this rule application is finite because in worst case we have to introduce n new variables with the same set term.

The main idea of the $\leq$-rule is written in Section 3.2.

The same goes for the *merge*-rule. We restrict the merging to the left hand side of cardinality constraints $k \lesseqqgtr l$: It can be reasonable for the right hand side if by merging l increases, for example $x : succ(1 \leq |r \cap t|)$ with $(x, y_1) : r$ and $(x, y_2) : t$. However the easiest solution is just to add an $r \cap t$-successor. In case it is important to also restrict the number of $r \cap t$-successor e.g. $x : succ(|r \cap t| < 2)$ we can define with the *choose*-rule whether y_1 and y_2 are also $r \cap t$-successor. And then with the assertion $x : succ(|r \cap t| < 2)$ we can merge y_1 and y_2.

The ≤ 0-rule deal with an assertion with a set constraint $s_1 \subseteq s_2$, which is written here as cardinality constraint $|s_1 \cap s_2^{\neg}| \leq 0$. Those cardinality constraint can not be dealt with the other rules. In case the left side has at least three set term e.g. $|s_1 \cap s_2 \cap s_3|$ we have can have multiple possible solutions e.g. $(x, y) : s_1 \cap s_2 \cap s_3^{\neg}$, $(x, y) : s_1 \cap s_2^{\neg} \cap s_3$ and $(x, y) : s_1 \cap s_2^{\neg} \cap s_3^{\neg}$. Hence we let the algorithm choose and backtrack if needed.

The *set.term*-rules are applied immediately after a new assertions $(x, y) : s$ is added to S. The reason for that is, that we want to add all needed assertions for y and hence update all $k^{\mathcal{I}(S)_x}$ correctly. We know that the number of this application is finite because an ABox is finite and hence the number of concept names and role names occurring in this ABox is also finite. Since the constraints are in NNF set terms can never be infinite and hence this rule applies only a finite times.

References

[1] F. Baader, I. Horrocks, C. Lutz, and U. Sattler. *An Introduction to Description Logic*. Cambridge University Press, 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] I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for very expressive description logics. *CoRR*, cs.LO/0005014, 05 2000.

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