1 Preliminaries

Let **C** be a set of concept names and **R** a set of role names such that they are disjoint.

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.

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:
  - $\neg C$
  - $C \sqcup D$

– $C \sqcap D$

In case we want to express $x : succ(s = t)$, with s, t being set terms, we write instead $x : succ(|s \cap t^{\neg}| \leq 0) \sqcap succ(|s^{\neg} \cap t| \leq 0)$.
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

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

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.

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

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 $(\neg C)^{\mathcal{I}}$.

2 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.

Definition 5 (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
- $\{x : succ(c)\} \subseteq S$ and c is violated regarding x and no more rules are applicable

To maintain readability 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.

Definition 6 (Induced Interpretation $\mathcal{I}(S)$)**.** 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 $r^{\mathcal{I}(S)}$

With $\mathcal{I}(S)$ we can count how many successors a variable has during the Tableau-algorithm.

Definition 7 (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 $n \in \mathbb{N}$.

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.
Concept names in $\mathbf{C}$ have a positive sign and role names in $\mathbf{R}$ have a positive sign.

For the Tableau-algorithm we define the properties of the following notations:

- Conjunction binds stronger than disjunction: $s \cup t \cap u = s \cup (t \cap u)$
- if k, l are cardinality terms then $k = l$ replaces $k \leq l$ and $k \geq l$

2.1 Restrictions

Similar in [1] and [2], where a variable can be replaced by another variable, we can merge two variables during the Tableau-algorithm.

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.

Note that by merging two successors of x into one variable y other assertions might become violated. In regards of the Tableau-algorithm we distinguish between two types:

- assertions regarding the predecessor x
- assertions regarding the successor y

We want to avoid the first cases because we can otherwise end with an endless loop of adding and merging variables:

Example 1.

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

We can clearly see that the ABox is unsatisfiable but the algorithm will try to fix the second assertion by adding a two new successors. After the second successors been added however the first assertions, which was satisfied, becomes violated. The algorithm tries then to merge the two variable leading the second assertions being violated. Then the cycle begins and the algorithm tries to satisfies the second assertion and so on. Hence we want to prevent the algorithm from violating already satisfied assertion regarding the predecessor.
However we allow the second case because it can be unavoidable to violate assertions regarding the successor first before fixing it later.

Example 2.

$$\begin{aligned} S = \{&x : succ(|r| \leq 1),\ (x, y_1) : r,\ (x, y_2) : r \\ &y_1 : succ(|s| \leq 1),\ y_2 : succ(|s| \leq 1),\ (y_1, z_1) : s,\ (y_2, z_2) : s\} \end{aligned}$$

In this example the ABox is violated but can be fixed by merging y_1 and y_2 together and z_1 and z_2. The only violated assertion is $x : succ(|r| \leq 1)$ hence we want to merge y_1 and y_2 to y. But then $y : (|s| \leq 1)$ becomes violated but can be fixed by merging z_1 and z_2. We could merge z_1 and z_2 first then $y : (|s| \leq 1)$ would not be violated but this would be too difficult to detect since the assertion regarding y_1 and y_2 are not violated. Therefore we want to *block* a merging of two variables if satisfied assertions regarding their successor become violated. To detect a possible violation we count how often the successors y_1, y_2 are counted in the assertion $x : succ(k \lesseqgtr l)$. The idea is that if y_1 and y_2 are counted more often in l than in k then there is a chance for the assertion's violation. More precisely: If there is a cardinality $n \cdot |s|$ in l (or k) and

- $(x, y_1) : s \in S$ and $(x, y_2) : s \in S$, then $l^{\mathcal{I}(S)_x}$ decreases by n after merging
- $(x, y_1) : s' \in S$ and $(x, y_2) : s^* \in S$ and $s = s' \cap s^*$, then $l^{\mathcal{I}(S)_x}$ increases by n after merging
- $(x, y_1) : s' \in S$ and $(x, y_2) : s^* \in S$ and $s = s' \cup s^*$, then $l^{\mathcal{I}(S)_x}$ decreases by n after merging

Hence we denote the *merge coefficient* of two successor y_1 and y_2 of x in a cardinality term k by $\#k(x, y_1, y_2)$ and calculate it as follows (respectively for $\#l(x, y_1, y_2)$)

Algorithm 1 Compute $\#k(x, y_1, y_2)$

S, ABox
$\#k(x, y_1, y_2) := 0$
y_1 and y_2, different variables in $Var(S)$
x predecessor of y_1 and y_2
if $k = n_1 \cdot |s_1| + \cdots + n_i \cdot |s_i|$ **then**
  for each s_p, $1 \leq p \leq i$ **do**
    if $(x, y_1) : s_p \in S$ and $(x, y_2) : s_p \in S$ **then**
      $\#k(x, y_1, y_2) := \#k(x, y_1, y_2) + n_p$
    end if
    if $(x, y_1) : s' \in S$ and $(x, y_2) : s^* \in S$ and $s_p = s' \cap s^*$ **then**
      $\#k(x, y_1, y_2) := \#k(x, y_1, y_2) - n_p$
    end if
    if $(x, y_1) : s' \in S$ and $(x, y_2) : s^* \in S$ and $s_p = s' \cup s^*$ **then**
      $\#k(x, y_1, y_2) := \#k(x, y_1, y_2) + n_p$
    end if
  end for
  return $\#k(x, y_1, y_2)$
else
  return 0
end if

Note that the blocking assertion is satisfied and at least l must be of the form $n_1 \cdot |s_1| + \cdots + n_i \cdot |s_i|$. For a satisfied assertion $x : succ(k \leq l)$, where k is of the form $n_1 \cdot |s_1| + \cdots + n_i \cdot |s_i|$ but not l, a merging can not lead into its violation.
We need to compute for both cardinality term k and l the *merge coefficient*, because not only $l^{\mathcal{I}(S)_x}$ can change after merging but also $l^{\mathcal{I}(S)_x}$. However if $\#l(x, y_1, y_2) - \#k(x, y_1, y_2)$ is greater than $l^{\mathcal{I}(S)_x} - k^{\mathcal{I}(S)_x}$ then the merging leads into a violation.

Definition 10 (Blocking)**.** A satisfied assertion $x : succ(k \lesseqgtr l)$ *blocks* two variables $y_1 \neq y_2$ from merging if $\#l(x, y_1, y_2) - \#k(x, y_1, y_2) > l^{\mathcal{I}(S)_x} - k^{\mathcal{I}(S)_x}$.

Also like in [1] and [2] we have to be *safe* when introducing new variables otherwise we may end in a endless loop or with a false output. In case of $x : succ(n \, dvd \, l)$ we

do not have to consider any special cases because in worst case we have to add n-times successors, which counts in $l^{\mathcal{I}(S)_x}$. The same goes for assertions with cardinality constraints of the form $x : succ(k \lesseqgtr l)$, with k being a number e.g. it does not contain any set terms. For the remaining cases we could just increase l until this assertion is satisfied. However in case k and l can increases e.g. both contain set terms, we have to avoid cases where k increases faster then l. The increase depends on the set term u for which we want to introduce a new variable y and add $(x, y) : u$ to S. To determine whether u is *safe* we again 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 where $k = n_1 \cdot |s_1| + \cdots + n_i \cdot |s_i|$ and $l = m_1 \cdot |t_1| + \cdots + m_j \cdot |t_j|$, $i, j \in \mathbb{N}$. Let $u = t_1 \cap \cdots \cap t_o$ with $1 \leq o \leq j$. If $n_k(u) < n_l(u)$ then we call u *safe*. 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, cardinality term of the form $n_1 \cdot |s_1| + \cdots + n_i \cdot |s_i|$
u, set term of the form $t_1 \cap \cdots \cap t_o$
for each $n' \cdot |s'_1 \cup \cdots \cup s'_p|$, $p \in \mathbb{N}$, in k **do**
  if $\exists q, 1 \leq q \leq p : u = s'_q \cap t'$ **then**
    $n_k(u) := n_k(u) + n'$
  end if
end for
return $n_k(u)$

If k is a number e.g. it does not contain a set term, then all u are *safe*.

This says that it is only safe to add a variable if l increases faster then k.
We also 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 (still finite) chain of disjunction and conjunction, we want to add the assertions of y before any other rule application so that all $n(x, k, S)$, for which $|s|$ occurs in k, also count y. This is important because we want to know the correct number of successors at any time so we can avoid any violation of assertions.

2.2 Algorithm

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

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 $x : succ(k < l)$ or $x : succ(k \leq l)$ and $(x, y) : k'$, k' occurs in k, but $(x, y) : k \notin S$
   $\rightarrow$ 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 $x : succ(n \, dvd \, l)$, which is violated
   $\rightarrow$ introduce a new variable y, choose $|s|$ in l and $S := S \cup \{(x, y) : s\}$. Then jump to rule 9

6. $\leq$-rule: S contains $x : succ(c)$, with $c \in \{k \leq l, k < l\}$, which is violated, and there is a safe set term $s := |s_1 \cap \cdots \cap s_n|$:
   $\rightarrow S := S \cup \{(x, y) : s\}$. Then jump to rule 9

7. *merge*-rule: S contains $x : succ(c)$, with $c \in \{k \leq l, k < l\}$, which is violated. For two successor $y_1 \neq y_2$ we have $(x, y_1) : s_1 \in S$ and $(x, y_2) : s_2 \in S$, with s_1 and s_2 in k and y_1 and y_2 not being blocked by the same assertion
   $\rightarrow$ merge y_1 and y_2

8. *set*-rule: S contains $x : succ(|s_1 \cap \cdots \cap s_j| \leq 0)$ and $(x, y) : s_1, \ldots, (x : y) : s_i$, $i < j$, 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
      $\rightarrow S := S \cup \{y : C\}$

Note that:

- s in 6 can also be of the form $t^{\neg}$.
- if $n_1 \, dvd \, n_2 \cdot l$ and $mod(n_2, n_1) \neq 0$ then $n_1 \, dvd \, l$ eventually $\rightarrow$ no infinite application of rule 6

Definition 13 (Derived Set)**.** A *derived set* is an ABox S' where rule 9 is not applicable.

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. We use $n(x, k, S)$ to count the successors of x in k which is important for detecting and avoiding violations of assertions. Now there might be a successor y which satisfies only some part of k in the given S such that $n(x, k, S)$ does not count y:

Example 3.

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

However there might be an interpretation $\mathcal{I}'$ where y is also a s-successor of x and hence $n(x, k, S) \neq n_{\mathcal{I}'}(x, k, S)$. 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.
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 rule 4. 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. In the $cardinality_1$-rule we first make a choice whether we take k or l from a cardinality constraints $k \lesseqgtr l$, where the chosen cardinality term has at least a term of the form $|s|$ with s being a set term. Intuitively it would be better to only want to increase l but there are cases, where we have to add a successor in k instead.

Example 5.

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

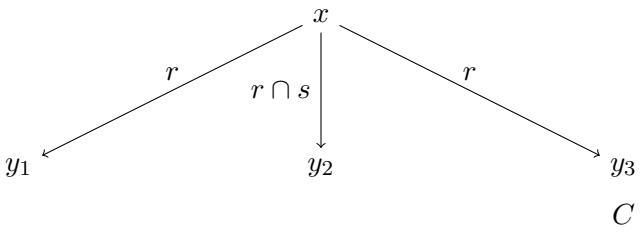
Note that $|s \cap r| + |s \cap r^{\neg}| + |s^{\neg} \cap r| = |s \cup r|$. No matter how often we add a successor in $|r|$ or $|s|$, the left hand side always increases as fast as the right hand side and therefore this assertion stays violated (hence it is not safe). However if we add a successor in $|s \cap r|$ the assertion becomes satisfied. Hence for this example we can choose the $cardinality_1$-rule and choose $i = |s \cap r| + |s \cap r^{\neg}| + |s^{\neg} \cap r|$. Then the rule pick one $|s^*|$ from i and look whether introducing a new successor y in s is safe. In our case let $s^* := s \cap r$. We know it is save to add this successor because $|s| + ||r|$ would increases faster. Therefore $(x, y) : s \cap r$ is added to S.
We consider now two examples to explain the $cardinality_2$-rule.

Example 6. Consider the following example

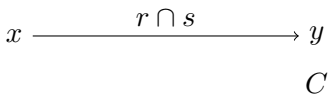
$$S = \{x : succ(|r| = 1) \sqcap succ(|r \cap s| = 1) \sqcap succ = (|r \cap C| = 1|)\}$$

First by the $\sqcap$-rule we split the whole assertion into three assertions and add them to S. By applying the $cardinality_1$-rule and add a new variable y we satisfy at least one

assertion but still at least one assertion remains violated. If we apply it a second time to satisfy the remaining assertion(s) then $x : succ(|r| = 1)$, which was satisfied before, becomes violated. In case we decided to apply the $cardinality_1$-rule on $x : succ(|r| = 1)$ and $x : succ(|r \cap s| = 1)$ we have the option of either applying rule the same rule on $x : succ(|r \cap C| = 1)$ next or applying the $cardinality_2$-rule on the two introduced variables next. We decide for the first choice:



In this case we have to merge three variables: y_1, y_2 and y_3. We can apply the $cardinality_2$-rule here because satisfied assertions regarding x do not become violated. We can also see that the order does not matter: In case we decided for the second choice and merge y_1 and y_2 to y before introducing y_3 we end up with the same results because then we have to merge y and y_3.



In the example we see a case where S is unsatisfiable.

Example 7.

$$S = \{x : succ(|r| < 2) \sqcap succ(|r| \geq 2)\}$$

First we use the $\sqcap$-rule to split the assertion into two assertions. The assertion $x : succ(|r| < 2)$ is satisfied therefore the algorithm tries to fix $x : succ(|r| \geq 2)$. Hence the $cardinality_1$-rule is applied. after that the first assertions remains satisfied but the second one remains violated. Therefore the algorithm applies the $cardinality_1$-rule again on $x : succ(|r| \geq 2)$ which result in $x : succ(|r| < 2)$ being violated. To fix this the algorithm tries to apply the $cardinality_2$-rule and tries to merge the two variable. However since $x : succ(|r| \geq 2)$ would become violated the merging is preempted and the algorithm stops with a clash. Since this sequence of rule applications is the only one possible we can say that all possible sequences leads to a clash which means that S in this example is unsatisfiable.
The applications of the *set.term*-rules eventually terminates because the number of concept names and role names are finite in S (since S is finite).

References

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

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