Extending SROIQ with Constraint Networks and Grounded Circumscription

MTP Report submitted to
Indian Institute of Technology, Mandi
for the award of the degree
of
B. Tech
by
Arjun Bhardwaj
under the guidance of
Prof. Deepak Khemani (Guide)
Prof. Dileep A.D. (Co-Guide)
Acknowledgments

I thank Prof. Deepak Khemani for his kind support and guidance, throughout the course of the project. I thank Prof. Dileep A. D. for his reviews of the work.

The problem dealt with, in the project, originated out of discussions with Sangeetha, AIDB Lab, IIT Madras. I thank her immensely for her expertise, helpful criticism and for her contributions towards framing and formalizing the research problem.

Arjun Bhardwaj
Abstract

Developments in semantic web technologies have promoted ontological encoding of knowledge from diverse domains. However, modelling many practical domains requires more expressiveness than what the standard description logics (most prominently SROIQ) support. In this paper, we extend the expressive DL SROIQ with constraint networks (resulting in the logic SROIQc) and grounded circumscription (resulting in the logic GC-SROIQ). Applications of constraint modelling include embedding ontologies with temporal or spatial information, while those of grounded circumscription include defeasible inference and closed world reasoning.

We describe the syntax and semantics of the logic formed by including constraint modelling constructs in SROIQ, and provide a sound, complete and terminating tableau algorithm for it. We further provide an intuitive algorithm for Grounded Circumscription in SROIQ, which adheres to the general framework of grounded circumscription, and which can be applied to a whole range of expressive logics for which no such specific algorithm presently exists.

**Keywords:** SROIQ, Constraint Networks, Grounded Circumscription, Description Logic, Tableau
# Contents

**Thesis Certificate**

**Abstract**

## 1 Introduction
1.1 Related Work .................................................. 2
1.2 Contribution of the work ...................................... 3
1.3 Organization of the Report .................................... 3

## 2 Augmenting SROIQ with Constraint Networks
2.1 Constraint Systems .............................................. 5
2.2 The Logic $SROIQ(C)$ ........................................... 5
   2.2.1 Syntax .................................................... 6
     The T-Box ................................................... 7
   2.2.2 Semantics ................................................ 8
     R-Box Interpretations ...................................... 8
     Concepts .................................................... 9
     T-Box interpretations .................................... 10
     A-Box interpretations .................................... 10
   2.2.3 Inference Problems ...................................... 10
   2.2.4 Knowledge Base Transformation ......................... 11
     Elimination of Universal Role ............................. 11
     A-Box partial reduction ................................ 11
     $R_a$ transformation ...................................... 11
     2.2.4.1 Negation Normal Form (NNF) ...................... 12
     RIA to automaton compilation ............................ 12
   2.2.5 An Augmented Tableau for $SROIQ(C)$ ................... 14
     Proposition : .............................................. 16
2.3 Tableau Algorithm for SROIQ(C) .............................. 16
   2.3.1 Preliminaries ............................................ 16
     Neighbours and Successors ............................... 17
     Marker Symbols and Constraint Template Set ........... 17
     Blocking .................................................. 17
     Blocking Check 1 (BC-1) ................................ 17
     Blocking Check 2 (BC-2) ................................ 18
Chapter 1

Introduction

The issue of knowledge representation lies at the heart of the quest for artificial intelligence. So, when First Order Logic (FOL) was formalized, at first it seemed like everything was solved. But it was soon observed that the semi-decidability of FOL and high time complexity of reasoning in FOL prohibited its use for reasoning over real-world-scale Knowledge Bases. This points us to the general phenomenon known as the representation-reasoning trade-off. If the logic used for representing knowledge is very expressive, and thus can be used for representing more complex pieces of information, then reasoning over such a logic would also be more difficult.

Description Logics (DLs) represent a comfortable compromise between representational expressiveness and ease of reasoning. They are a popular choices for modelling domain knowledge, given that inference procedures in these logics is decidable, sound and complete. The developments in DLs have directly fuelled semantic web technologies which depend on them to provide reasoning support over real world ontologies.

Recent years have seen a trend towards formalizing and capturing knowledge from diverse domains in the form of ontologies. Standard DLs are able to provide reasoning support for the vast majority of ontologies. However, there are some domains which require more expressive representation mechanisms to formalize their knowledge. We focus on two such popularly proposed extensions.

Several ontologies require the ability to embed spacial and temporal information about the entities being modelled. Further, many ontologies require the general constraint modelling ability to formally specify the constraints between entities in terms of a constraint network. An important result, provided by [3], allows us to merge the above two. Constraint networks have been proposed as a viable way to capture temporal and spacial information. Most alternative techniques compromise on either expressiveness of constraint modelling or come at the cost of heightened time complexity of the resulting logic. The importance of embedding temporal information is discussed in [1], which also provides a guideline on the constructs that must be made available to OWL to allow expressive modelling of temporal information.

On the other hand, DLs are all open domain. Intuitively this means that what is expressed in the Knowledge Base is true, but one can not assume that what is not mentioned to be false. This poses difficulties in reasoning in many cases, and hence several ways have been explored to facilitate closed world reasoning in DLs, which is in accordance with the assumption that only what is expressed is true. Additionally, many domains have rules which apply "in general", but not always. The strict logical rules offered by DLs are not able to correctly model these default
rules. Out of the several solutions that have been proposed to meet the above requirements, grounded circumscription \cite{4} is a formalism that provides an intuitive and elegant solution.

In this paper, we extend the expressive DL $SROIQ$ \cite{5} with constraint networks and grounded circumscription. The aim is to formalize the logic by defining its syntax and semantics and to propose a tableau decision procedure to support reasoning in it.

### 1.1 Related Work

Earlier attempts to embed temporal information into DLs have only been realized for less expressive logics like $ALC$. For example \cite{2}. The work by \cite{3} augments $ALC$ with $\omega$-admissible Constraint Systems to form a decidable DL. The $\omega$-admissible systems identified were the Allens’ relations (for temporal intervals) and RCC8 relations (for spatial regions).

Circumscription is a non-monotonic framework that allows closed world reasoning and default reasoning. However, minimization of roles is undecidable even for simpler DLs and tableau procedures exist only for concept minimization in logics with finite model property. \cite{4} uses grounded circumscription, using which both concept and role minimization in DLs is decidable. However, a tableau procedure exists only for ALCO and involves special rules to ensure the construction of a grounded model.

### 1.2 Contribution of the work

The current work incorporates into $SROIQ$, the constraint modelling constructs, including those mentioned in \cite{3}. We also specify the conditions which must be imposed on these constructs to ensure the existence of a tableau algorithm for the resulting logic. Besides these constructs we introduce the following features for constraint modelling:

- ability to explicitly name variables in constraint networks
- simple hierarchies in concrete roles
- constraint modelling constructs that make use of the named variables
- number restrictions on concrete roles.

Also provided are sketches of the proofs of correctness of the resulting tableau algorithm.

We further provide the details of a intuitive, general algorithm that can be used to allow circumscription circumscription in $SROIQ$ and a range of other logics. Unlike existing algorithms, it simply uses some of the constructs of $SROIQ$, to perform the tasks outlined in the grounded circumscription framework \cite{4}.

### 1.3 Organization of the Report

In chapter 2, we explore the extension of $SROIQ$ to include constraint networks. We provide the syntax, semantics and the tableau inference procedure of the resulting logic. We argue for its
correctness by providing sketches of corresponding proofs in the Appendix. We argue that more expressive constraint modelling constructs pose several fundamental, unavoidable difficulties towards the design of a tableau algorithm for the logic.

In the following chapter we take up the case of grounded circumscription and provide an algorithm that permits grounded circumscription in $\mathcal{SROIQ}$. 
Chapter 2

Augmenting SROIQ with Constraint Networks

Here we present the logic that provides constraint modelling constructs to SROIQ.

2.1 Constraint Systems

We use the notion of a constraint system as defined in [3]. Let $\text{Var}$ be a countably infinite set of variables and $\text{Rel}$ a finite set of relation symbols. While describing the algorithm, we may use the term concrete domain to denote the set $\text{Var}$.

A $\text{Rel}$-constraint is an expression of the form $(v \ r \ v')$, where $v, v' \in \text{Var}$, $r \in \text{Rel}$. A $\text{Rel}$-network is a finite set of $\text{Rel}$-constraints. For a $\text{Rel}$-network $N$, let $V_N$ denote the variables used in the network.

$N$ is complete if for every $v, v' \in V_N$, there is exactly one constraint $(v \ r \ v') \in N$. Let $\mathfrak{M}$ be the set of all complete networks, possible according to the semantics of the domain being modelled.

A network $N'$ is a model of a network $N$, if $N' \in \mathfrak{M}$ and there is a mapping $\tau : V_N \to V_N'$ such that $(v \ r \ v') \in N$ implies $(\tau(v) \ r \tau(v')) \in N'$. A network $N$ is satisfiable in $C$ if $\mathfrak{M}$ contains a model of $N$. We define a constraint system $C$ as the tuple $\langle \text{Rel}, \mathfrak{M} \rangle$.

Next, we define the some terms that help identify the set of constraint systems which can be used with the algorithm, without the loss of decidability.

Given any finite constraint networks $N, M$, their intersection is defined as follows: $I_{N,M} := \{(v \ r \ v') \mid v, v' \in V_N \cap V_M \text{ and } (v \ r \ v') \in N\}$

$C$ has the patchwork property if for any finite and satisfiable networks $M, N$ with complete and equal intersections (i.e. $I_{M,N}$ and $I_{N,M}$ are complete networks, $I_{M,N} = I_{N,M}$), $N \cup M$ is satisfiable. $C$ has the compactness property if the following holds: a network $N$ with infinite set of variables $(V_N)$ is satisfiable, if and only if, for every finite $V \subseteq V_N$, the network $N \mid_V = \{(v rv') \in N \mid v, v' \in V\}$ is satisfiable. We say that $C$ is $\omega$-admissible if the following holds

1. Satisfiability in $C$ is decidable
2. $C$ has patchwork property
3. $C$ has the compactness property

5
To ensure decidability, only \( \omega \)-admissible constraint system are permitted to be used in tableau algorithms for constraint modelling logics described here.

### 2.2 The Logic \( \mathcal{SROIQ}(C) \)

We describe \( \mathcal{SROIQ}(C) \): a logic that provides relatively controlled constraint modelling constructs on top of \( \mathcal{SROIQ} \) \([5]\), in order to ensure the existence of a tableau for inference over the language. In this section, we describe the constructs of the logic, their semantics and provide a tableau decision procedure for \( \mathcal{SROIQ}(C) \)-KB satisfiability.

#### 2.2.1 Syntax

Let \( N_c \) be the set of concept names, \( Roles \) the set of abstract role names (including the universal role \( U \)), \( N_cR \) the set of concrete role names, \( N_{ai} \) names of abstract individuals, \( Nom \) the set of nominal concept names (also called nominals), \( Var \) the set of variables which participate in the Rel-constraints and \( N_{ci} \subset Var \) the set of constraint individual names.

A \( \mathcal{SROIQ}(C) \) Knowledge Base (\( \mathcal{K} \)) consists of a tuple \( \langle A, T, R \rangle \) where \( A, T, R \) are respectively the A-Box, T-Box and R-Box, which are described as follows. Since the logic is built on top of \( \mathcal{SROIQ} \), it shares most of its constructs with it. We include description of these here for completeness, and the user may refer to \([5]\) for more detailed treatment.

**R-Box** \( R \) is defined as the tuple \( \langle R_h, R_{ch}, R_{ah} \rangle \). We define these, in the subsequent paragraphs.

The Inverse of a role is defined as follows :

\[
\text{Inv}(R) = \begin{cases} R^- & \text{if } R \in Roles \\ S & \text{if } R = S^- \text{ for some } S \in Roles \end{cases}
\]

We define \( N_{cR} = Roles \cup \{R^- \mid R \in Roles \} \).

A role chain is an expression of the form \( R_1 \ldots R_n \) with \( n \geq 1 \) and each \( R_i \in N_{cR} \). The notion of \( \text{Inv} \) is extended to role chains as :

\[
\text{Inv}(R_1R_2\ldots R_n) = \text{Inv}(R_n)\ldots \text{Inv}(R_2)\text{Inv}(R_1)
\]

\( R_{ch} \) is the set of concrete Role Inclusion Axioms (c-RIA) each of which is of the form \( g \sqsubseteq g' \) where \( g, g' \in N_{cR} \). This defines a hierarchy between the concrete roles. \( R_{ah} \) is a finite set of abstract Role Inclusion Axioms (a-RIA) of the form \( w \sqsubseteq R \) where \( w \) is a role chain, possibly of unit length and \( R \in N_{aR} \). However, in order to ensure decidability of the tableau algorithm, the set \( R_{ah} \) must be a Regular. The notion is explained below. \( \sqsubseteq \) is the closure of \( \sqsubseteq \) relation between roles.

A regular order \( < \) is an irreflexive transitive binary relation on the set of roles \( N_{aR} \) satisfying :

\( R_1 < R_2 \) iff \( \text{Inv}(R_1) < \text{Inv}(R_2) \). \( R_{ah} \) is Regular if there is a regular order on \( N_{aR} \) such that each RIA in \( R \) is one of the following forms:

1. \( R_1 \ldots R_n \sqsubseteq R \) with \( R_i < R \) for all \( i \leq i \leq n \)
2. \( RR_1 \ldots R_n \sqsubseteq R \) with \( R_i < R \) for all \( i \leq n \)

3. \( R_1 \ldots R_n R \sqsubseteq R \) with \( \forall R_i < R \) for all \( i \leq n \)

4. \( RR \sqsubseteq R \)

5. \( \text{Inv}(R) \sqsubseteq R \)

Associated with the notion of regularity of an abstract is the notion of a simple role. Given a role hierarchy \( \mathcal{R}_{ah} \), the set of roles that are simple in \( N_{ah} \), is inductively defined as follows:

- A role name is simple if it does not occur on the R.H.S. of a RIA in \( \mathcal{R}_{ah} \)
- An inverse role \( R^- \) is simple if \( R \) is, and
- If \( R \) occurs on the R.H.S. of a RIA in \( \mathcal{R}_{ah} \), then \( R \) is simple if, for each \( w \sqsubseteq R \in \mathcal{R}_{ah} \), \( w = S \) for a simple role \( S \)

We further exclude the universal role \( U \) from being in the set of simple roles. An RIA is simple if it is of the form \( S \sqsubseteq S' \), where \( S, S' \) are simple roles.

A path is a sequence \( R_1, \ldots, R_n g \) consisting of simple roles \( R_1, \ldots, R_k \in N_{ah} \) and a concrete role \( g \in N_{cr} \). We define the length of a path as the number of abstract or concrete roles that appear in the sequence.

\( \mathcal{R}_a \) is a finite set of role assertions. A role assertion is of the form \( \text{Ref}(R) \) (reflexivity), \( \text{Irr}(S) \) (irreflexivity), \( \text{Dis}(S, S') \) (role disjointness), \( \text{Sym}(R) \) (symmetry), \( \text{Trans}(R) \) (transitivity), \( \text{Fxn}(X) \) (functionality), where \( S, R \in N_{ah} \), \( S \) is a simple role and \( X \in N_{ah} \cup N_{cr} \) must be a simple role if it is an abstract role.

In conclusion, the R-Box features novel to \( \text{SROIQ} \) w.r.t. \( \text{SROIQ} \) include the concrete role hierarchy \( (\mathcal{R}_{ah}) \), and the functional role assertion for concrete roles.

**Notational Conventions**  Unless mentioned otherwise, the following notational conventions are followed: \( r, r' \in \text{Rel}(C) \); \( g, g' \in N_{cr} \); \( R, R' \in N_{ah} \); \( S, S' \in N_{ah} \) are simple abstract roles; \( A, B, C, D \) are concepts; \( o \in \text{Nom} \) i.e. \( o \) is a nominal set; \( i \in N_{ci} \) i.e. \( i \) is a named variable - a constraint individual, \( a, b \in N_{ai} \) i.e. they are abstract named individuals from the domain being modelled; \( U \) is a path; \( G, G' \) are paths. The same applies to respective symbols with subscripts.

**The T-Box**  The set of \( \text{SROIQ}(C) \) concepts, \( \text{Concepts} \), is defined recursively as follows:

\[
A \mid o \mid (C \sqcap D) \mid (C \sqcup D) \mid \neg C \mid \exists R.C \mid \forall R.C \\
\geq nS.C \mid \leq nS.C \mid \exists S.Self \\
\leq_c \text{ng} \mid \geq_c \text{ng} \\
\exists U_1, U_2, r \mid \exists U_1, \{i\}, r \mid \exists \{i\}, U_1, r \\
\forall U_1, U_2, r \mid \forall_c U_1, \{i\}, r \mid \forall_c \{i\}, U_1, r
\]

Here \( A \in N_C \), \( C, D \in \text{Concepts} \). In the above, all the non-\( \text{SROIQ} \) concepts (the last three lines) are known as concrete concepts.
There are some constraints imposed on the paths that comprise the concrete concepts. For concepts of the form \((\exists U_1, \{i\}.r), (\exists \{i\}, U_1,r), (\forall U_1, \{i\}.r), (\forall \{i\}, U_1.r)\), \(U_1\) is restricted to be a path of length at most 2 i.e. it can be of the form \(Rg\) or \(g\). For the concepts \((\exists U_1, U_2.r)\) and \((\forall U_1, U_2.r)\), if one of the paths is of length 2, then the other must be of the length 1. This makes the permissible combinations of \((U_1, U_2)\) to be of the form \((Rg, g)\), or \((g, g)\) or \((g, Rg)\). This condition is the Path Normal Form, as defined in [3].

A SROIQ(C) TBox contains GCIs of the form \(C \subseteq D\), where \(C, D \in \text{Concepts}\).

In conclusion, concrete concepts are novel to SROIQ(C) w.r.t. SROIQ, and it is through their use that the logic allows constraint modelling.

**A-Box** The SROIQ A-Box consists of a finite set of assertions of the form

\[
C(a), \ R(a, b), \ \neg R(a, b), \ a \neq b
\]  

(2.2)

Here \(a, b \in N_{at}, i_1, i_2 \in N_{ct}, r \in \text{Rel}\) and \(\text{Rel}\) is the set of relations defined for the constraint system. Allen’s relation are a common choice for \(\text{Rel}\), which enables us to define temporal constraint.

We extend the SROIQ A-Box to include assertions of the form \((i_1, r i_2)\) and \(g(a, i_i)\), where \(i_1, i_2 \in N_{ct}\). We impose the constraint that for every \(i \in N_1\) there must be some \(g, a\) such that ABox contains \(g(a, i)\). We assume Unique Name Assumption(UNA) for constraint individuals \((N_{ct})\), but not for abstract individuals \((N_{at})\).

### 2.2.2 Semantics

An interpretation \(\mathcal{I}\) is a tuple \((\Delta_I, \cdot^I, M_I)\), where \(\Delta_I\) is the abstract domain, \(\cdot^I\) is the interpretation function, and \(M_I \in \mathfrak{M}\), is a complete constraint network of the constraint system \(C\).

The interpretation function maps

- Each concept name \(C \in N_C\) (atomic concept) to a subset \(C^I\) of \(\Delta_I\)
- Each abstract role \(R \in N_{ar}\) to a subset \(R^I\) of \(\Delta_I \times \Delta_I\)
- Each concrete role \(g \in N_{cr}\) to a subset \(g^I\) of \(\Delta_I \times V_{M_I}\)
- Each named abstract individual \(a \in N_{at}\) to an element \(a^I \in \Delta_I\)
- Each constraint individual \(i \in N_{ct}\) to an element \(i^I \in V_{M_I}\)

This notion is extended to other concepts and roles, as defined in the following paragraphs.

**R-Box Interpretations** The interpretation is extended to inverses as follows: \((R^-)^I = \{\langle x, y \rangle \mid \langle y, x \rangle \in R^I\}\). For a path \(U = R_1 \ldots R_k g\) and \(d \in \Delta_I\), \(U^I(d)\) is defined as

\[
\{x \in V_{M_I} \mid \exists e_1, \ldots, e_{k+1} : d = e_1, \langle e_i, e_{i+1} \rangle \in R_i^I \text{ for } 1 \leq i \leq k, \text{ and } \langle e_{k+1}, x \rangle \in g^I\}
\]

\(\mathcal{I}\) satisfies (is a model of) the concrete RIA \(g_1 \sqsubseteq g_2\) iff \(g_1^I \subseteq g_2^I\). An interpretation \(\mathcal{I}\) satisfies the abstract RIA \(R_1 R_2 \ldots R_n \sqsubseteq R\), if \(R_1^I \circ R_2^I \circ \ldots \circ R_n^I \subseteq R^I\), where \(\circ\) is teh binary composition operator. The satisfaction of the role assertions is along the same lines as defined in [5].
An interpretation is a model of R-Box if it satisfies all concrete and abstract RIAs and the role assertions.

**Concepts**  The interpretation function is extended to arbitrary concepts as follows:

\[ (¬C)^I := \Delta^I \setminus C^I, \]
\[ o^I := \{d^I\}, \text{ if } o = \{a\}, \text{ else } \{x\} \text{ for } c \in \Delta^I \]
\[ \top^I := \Delta^I, \]
\[ \bot^I := \emptyset, \]
\[ (C \cap D)^I := C^I \cap D^I, \]
\[ (C \cup D)^I := C^I \cup D^I, \]
\[ (\exists R.C)^I := \{d \in \Delta^I \mid \text{there is some } d, e \in \Delta^I \]
\[ \quad \text{with } (d, e) \in R^I \text{ and } e \in C^I\}, \]
\[ (\forall R.C)^I := \{d \in \Delta^I \mid \text{for all } e \in \Delta^I \text{ if } \]
\[ \quad (d, e) \in R^I \text{ then } e \in C^I\}, \]
\[ (\exists R Sel f)^I := \{x \mid (x, x) \in R^I\}, \]
\[ (\geq n R.C)^I := \{x \mid \#\{y | (x, y) \in R^I \text{ and } y \in C^I\} \geq n\}, \]
\[ (\leq n R.C)^I := \{x \mid \#\{y | (x, y) \in R^I \text{ and } y \in C^I\} \leq n\} \]

\[ (\exists U_1, U_2, r)^I := \{d \in \Delta^I \mid \text{there exist some } x_1 \in U_1^I(d) \]
\[ \quad \text{and } x_2 \in U_2^I(d) \text{ such that } (x_1 r x_2) \in M^I\}, \]
\[ (\forall U_1, U_2, r)^I := \{d \in \Delta^I \mid \text{for all } x_1 \in U_1^I(d) \text{ and } \]
\[ \quad x_2 \in U_2^I(d), \text{ we have } (x_1 r x_2) \in M^I\}, \]
\[ (\exists U_1, U_2, r)^I := \{d \in \Delta^I \mid \text{there exist some } x_1 \in U_1^I(d) \]
\[ \quad \text{and } i \in N_{cl}, \text{ such that } (x_1 r i) \in M^I\}, \]
\[ (\exists U_1, U_2, r)^I := \{d \in \Delta^I \mid \text{there exist some } i \in N_{cl} \]
\[ \quad \text{and } \text{x} \in U_2^I(d), \text{ such that } (i r x) \in M^I\}, \]
\[ (\forall U_1, U_2, r)^I := \{d \in \Delta^I \mid \text{for all } x_1 \in U_1^I(d) \text{ and for } \]
\[ \quad i \in N_{cl}, \text{ we have } (x_1 r i) \in M^I\}, \]
\[ (\forall U_1, U_2, r)^I := \{d \in \Delta^I \mid \text{for } i \in N_{cl} \text{ and all } \]
\[ \quad x_1 \in U_1^I(d), \text{ we have } (i r x_1) \in M^I\}, \]
\[ (\leq c n g)^I := \{x \mid \#\{c | \langle x, c \rangle \in g^I \text{ and } \]
\[ \quad c \in V_{M^I}\} \leq n\}, \]
\[ (\geq c n g)^I := \{x \mid \#\{c | \langle x, c \rangle \in g^I \text{ and } \]
\[ \quad c \in V_{M^I}\} \geq n\} \]
An interpretation \( I \) is a model of a concept \( C \) iff \( C^I \neq \emptyset \).

**T-Box interpretations** \( I \) is a model of a T-Box \( T \) iff it satisfies \( C^I \subseteq D^I \) for all \( C \sqsubseteq D \) in \( T \).

**A-Box interpretations** An interpretation \( I \) satisfies (is a model of) an Abox \( A(I \models A) \) if for all individual assertions \( \phi \in A \) we have \( I \models \phi \), where

\[
\begin{align*}
I \models C(a) & \quad \text{if } a^I \in C^I; \\
I \models a \neq b & \quad \text{if } a^I \neq b^I; \\
I \models R(a, b) & \quad \text{if } \langle a^I, b^I \rangle \in R^I; \\
I \models \neg R(a, b) & \quad \text{if } \langle a^I, b^I \rangle \notin R^I; \\
I \models g(a, i) & \quad \text{if } a^I \in \Delta^I, i^I \in V_{M^I} \text{ and } \langle a^I, i^I \rangle \in g^I \\
I \models (i_1^I r i_2^I) & \quad \text{if } i_1^I, i_2^I \in V_{M^I}, r \in \text{Rel}; \langle i_1^I, r, i_2^I \rangle \in M^I
\end{align*}
\]

The constraint individuals provide a way to directly address the variables of the constraint network, which would be formed by the tableau algorithm. It allows us to explicitly name concrete nodes of the completion system.

A knowledge base \( \langle T, R, A \rangle \) is satisfiable (consistent) if there exists an interpretation which is a model for each of \( T, R \) and \( A \).

### 2.2.3 Inference Problems

Consider the common inference problems.

1. KB Satisfiability : A knowledge base \( \mathcal{K} = \langle T, R, A \rangle \) is satisfiable (consistent) if there exists an interpretation which is a model for each of \( T, R \) and \( A \).

2. Concept Satisfiability : A concept \( C \) is satisfiable w.r.t. a knowledge base \( \mathcal{K} \) if there exists an interpretation which is a model of the KB and for which \( C^I \) is not empty.

3. Concept Subsumption : A concept \( C \) subsumed by a concept \( D \) (\( C \sqsubseteq D \)) if \( C^I \subseteq D^I \) holds for all models of the \( \mathcal{K} \).

4. Instance Checking : An instance \( a \) belonging to a concept \( C \) is denoted as \( C(a) \). The check succeeds if \( a^I \in C^I \) for all models of \( \mathcal{K} \).

Let \( \mathcal{K} \) be a SROIQ(C) knowledge base and let \( C, D \) be concepts in \( \mathcal{K} \). Let \( a \) be an individual, such that \( a \notin N_{al} \) i.e. it is a named individual, new to the KB. Common inference problem can be converted to KB satisfiability as described below.

- **Concept satisfiability** : A concept \( C \) is satisfiable iff \( \mathcal{K} \cup C(a) \) is satisfiable, for a novel \( a \), not already mentioned in \( \mathcal{K} \).

- **Concept subsumption** : \( \mathcal{K} \models (C \sqsubseteq D) \) iff \( \{a\} \sqsubseteq (C \cap \neg D) \) is unsatisfiable.

- **Instance checking** : the problem \( \mathcal{K} \models C(b) \) is equivalent to the concept subsumption problem \( \{b\} \sqsubseteq C \), which is converted as mentioned above.
Thus a single tableau algorithm to decide the satisfiability of a $SROIQ(C)$ KB can be used to handle the above inference problems, as well.

### 2.2.4 Knowledge Base Transformation

The KB needs to be in a specific form before the Algorithm can operate. The following reductions are performed, in order, before the tableau algorithm operates on the KB. The transformed, reduced KB is then input to the tableau algorithm. We point out that all notions (e.g. $N_{aR}$ etc.) correspond to this reduced KB.

#### Elimination of Universal Role

Let $C, D$ be concepts. Let $U' \neq U$ be a role that does not occur in the knowledge base $\mathcal{K}$. We modify the R-Box as defined below:

- $\mathcal{R}^{U'}_{ah} := \mathcal{R}_{ah} \cup \{R \subseteq U' \mid R \text{ occurs in } \mathcal{K}\}$
- $\mathcal{R}^{U'}_{a} := \mathcal{R}_{a} \cup \{Tra(U'), Sym(U'), Ref(U')\}$
- $\mathcal{R}^{U'} = \mathcal{R}^{U'}_{ah} \cup \mathcal{R}^{U'}_{a}$

Hereafter, when we mention $N_{aR}$, we do not include the universal role in it.

#### A-Box partial reduction

We replace some of the $\mathcal{A}$ assertions with an set of SGI axioms, to form a new T-Box $\mathcal{T}_{A}$. Named individual, say $a$ used in assertions are converted to corresponding nominal set $o_a$. The assertions are converted to semantically equivalent axioms:

1. $C(a)$ to $o_a \subseteq C$
2. $R(a, b)$ to $o_a \subseteq \exists R.o_b$
3. $\neg R(a, b)$ to $\top \subseteq \forall R. \neg\{o_b\}(\{o_a\})$
4. $a \neq b$ to $o_a \subseteq \neg o_b$

The two new types of assertions introduced in $GC-SROIQ(C)$ are, however, not reduced.

#### $\mathcal{R}_{a}$ transformation

This involves converting the role assertions into equivalent forms. The reduced RBox consists of only assertions of the form $Dis(S, S')$ and $Irr(R)$.

The role assertion $Ref(S)$ is converted to an equivalent T-Box axiom

$$\top \subseteq \exists S. Self$$

The role assertion $Fxnl(S), S \in N_{aR}$ is converted to an equivalent T-Box axiom

$$\top \subseteq \leq 1S$$

The analogue for functional restriction on concrete role $g$ would be:

$$\top \subseteq \leq_c 1g$$

The role assertions $Sym(R), Tra(R)$ is converted to an equivalent R-Box axiom
Next, we describe the translation of concepts into their NNF equivalents and the compilation of complex RIA into automata.

### 2.2.4.1 Negation Normal Form (NNF)

The tableau algorithm expects the concept constructs to be in NNF. The negation appears only in front of "primary" concepts in NNF. The set of primary concepts consists of all atomic concepts ($N_C$ and $Nom$), $\exists S . \text{Self}$, $\exists_c U_1, U_2 . r$, $\exists_c U_1, \{i\} . r$ (and its analogue $\exists_c \{i\}, U_1 . r$). For the other constructs, NNF equivalents are defined as below. In some of these cases, the NNF is not the logical equivalent, but preserves (un)satisfiability.

For the non-atomic primary concepts, NNF is not defined, for lack of a feasible closed expression. The semantics of these irreducible constructs are handled by the tableau algorithm by means of completion rules and special clash conditions, as mentioned later. Some special constructs ($\exists_c U_1 . q$, $\forall_c U_1 . q$) would be introduced by the tableau completion rules. NNF are not defined for these as they will never be present with a negation in front and are used for reasons internal to the tableau algorithm.

The NNF rules are shown:

\[
\begin{align*}
\neg(\neg C) & \rightarrow C \\
\neg(C \sqcap D) & \rightarrow \neg C \sqcup \neg D \\
\neg(C \sqcup D) & \rightarrow \neg C \sqcap \neg D \\
\neg(\exists R . C) & \rightarrow \forall R \neg C \\
\neg(\forall R . C) & \rightarrow \exists R \neg C \\
\neg(\leq n R . C) & \rightarrow (\geq (n+1) R . C) \\
\neg(\geq (n+1) R . C) & \rightarrow (\leq n R . C) \\
\neg(\geq 0 R . C) & \rightarrow \bot \\
\neg(\leq ng) & \rightarrow (\geq (n+1)g) \\
\neg(\geq_c (n+1)g) & \rightarrow (\leq_c ng) \\
\neg(\geq_c 0g) & \rightarrow \bot \\
\neg(\forall_c U_1, U_2 . r) & \rightarrow \bigcup_{r' \in \text{Rel}, r' \neq r} \exists_c U_1, U_2 . r' \\
\neg(\forall_c U_1, i_1 . r) & \rightarrow \bigcup_{r' \in \text{Rel}, r' \neq r} \exists_c U_1, i_1 . r'
\end{align*}
\]

**RIA to automaton compilation** This section is a direct adaptation of the work in [5] and is presented for completeness of the text. It describes the construction of a non-deterministic automaton for every complex (non-simple) role of the RBox, using the complex RIA’s of $\mathcal{R}_{abs}$. Given a complex role $R$, the constructed automaton $\mathcal{B}_R$ accepts all strings $\mu$ such that $\mu$ is a role chain.
which for all model interpretations $I$, satisfies: $\mu^f \subseteq R^f$. $B_R$ captures all implications between (paths of) roles and $R$ that are consequences of $R_{ah}$.

The automata $B_R$ is built in steps.

Firstly, for each non-simple role name $R$ occurring in $K$, define an NFA $A_R$ as follows: $A_R$ contains a state $i_R$ and a state $f_R$ with the transition $i_R \xrightarrow{R} f_R$. The state $i_R$ is the only initial state and $f_R$ is the only final state. Moreover, for each $(w \sqsubseteq R) \in R_{ah}$, $A_R$ contains the following states and transitions:

1. If $w = RR$, then $A_R$ contains $f_R \xrightarrow{\epsilon} i_R$, and
2. If $w = R_1 \ldots R_n$ and $R_1 \ldots R_n \neq R$, then $A_R$ contains
   
   $i_R \xrightarrow{\epsilon} i_w \xrightarrow{R_1} f_w \xrightarrow{R_2} f_w \xrightarrow{R_3} \ldots \xrightarrow{R_n} f_w \xrightarrow{\epsilon} f_R$

3. If $w = RR_2 \ldots R_n$, then $A_R$ contains
   
   $f_R \xrightarrow{\epsilon} i_w \xrightarrow{R_2} f_w \xrightarrow{R_3} f_w \xrightarrow{R_4} \ldots \xrightarrow{R_n} f_w \xrightarrow{\epsilon} f_R$

4. If $w = RR_2 \ldots R_n$, then $A_R$ contains
   
   $f_R \xrightarrow{\epsilon} i_w \xrightarrow{R_2} f_w \xrightarrow{R_3} f_w \xrightarrow{R_4} \ldots \xrightarrow{R_n} f_w \xrightarrow{\epsilon} f_R$

5. If $w = R_1 \ldots R_{n-1} R$, then $A_R$ contains
   
   $i_R \xrightarrow{\epsilon} i_w \xrightarrow{R_1} f_w \xrightarrow{R_2} f_w \xrightarrow{R_3} \ldots \xrightarrow{R_{n-1}} f_w \xrightarrow{\epsilon} i_R$

where all $f_w^i, i_w$ are assumed to be distinct.

To construct a mirror copy $\tilde{A}_R$ of an automaton $A_R$, the following procedure is followed:

- Make final states to non-final states
- Make initial states to non-initial but final states
- Replace each transition $p \xrightarrow{S} q$ with $q \xrightarrow{\text{Inv}(S)} p$, where $S$ is a (possibly inverse) role.
- Replace each transition $p \xrightarrow{\epsilon} q$ with $q \xrightarrow{\epsilon} p$

Secondly, we define the NFAs $\tilde{A}_R$ as follows:

- If $R^- \not\sqsubseteq R \not\in R$ then $\tilde{A}_R := R_a$
- If $R^- \not\sqsubseteq R \in R$ (R is transitive), then $\tilde{A}_R$ is obtained as follows:

  1. Take a disjoint union of $A_S$ with a mirrored copy of $A_S$
  2. Make $i_R$ the only initial state and $f_R$ the only final state
  3. For $f_R'$ the copy of $f_R$ and $i_R'$ the copy of $i_R$, add transitions $i_R \xrightarrow{\epsilon} f_R'$, $f_R' \xrightarrow{\epsilon} i_R$
Thirdly, the NFAs $B_R$ are defined inductively over $\prec$:

- If $R$ is minimal w.r.t. $\prec$ (i.e., there is no $R'$ with $R' \prec R$), we set $B_R := \hat{A}_R$
- Otherwise, $B_R$ is the disjoint union of $\hat{A}_R$ with a copy $B'_S$ of $B_S$ for each transition $p \xrightarrow{\delta} q$ in $\hat{A}_R$ with $\delta \neq R$. Moreover, for each such transition, we add $\epsilon$-transitions from $p$ to the initial state in $B'_S$ and from the final state in $B'_S$ to $q$, and we make $i_R$ the only initial state and $f_R$ the only final state $B_R$.

We will assume that final, reduced and converted KB formed by following the above series of operations if given by $\mathcal{K} = \langle R, T, \mathcal{A} \rangle$.

### 2.2.5 An Augmented Tableau for SROIQ(C)

We propose an **augmented tableau** for SROIQ(C), $T_A = \langle T, N \rangle$ where $T = \langle V_a^T, V_c^T, L^T, E^T_a, E^T_c \rangle$ is a tableau and $N$ is a constraint network with $V_N = V_c^T$. The augmented tableau for a KB is a graphical representation of a model of the KB $I = \langle \Delta_T, T, \mathcal{A} \rangle$.

The elements of the non-empty set $\hat{V}_a^T$ forms the nodes of tableau $T$. $\text{Nom}(T) \subset \hat{V}_a^T$ is the set of nominal nodes and represents named individuals. $V_c^T$ forms the variables which participate in constraint relations that compose $N$. Further, we define the following mappings:

- $L^T : \hat{V}_a^T \rightarrow 2^{\text{close}(T, \mathcal{R})}$
- $E^T_a : N_R \rightarrow 2^{V_c^T \times V_a^T}$
- $E^T_c : N_{\mathcal{L}} \rightarrow 2^{V_c^T \times V_a^T}$

For a concept $C$ and $D$, and a SROIQ(C) KB $\mathcal{K} = \langle \mathcal{A}, T, \mathcal{R} \rangle$, we define $\text{sub}(D)$, the set of sub-expressions of $D$. The intuitive significance of $\text{sub}(D)$ is in describing the contents of the label sets of graph nodes i.e. the concepts formed that label the nodes of graph, if the KB consists of no other concept other than $D$. Consider the following:

1. If $D$ is of the form $\neg C, \exists R.C, \forall R.C, \leq nR.C$ or $\geq nR.C$, then $\text{sub}(D) = \{C, D\}$
2. If $D$ is of the form $C_1 \sqcap C_2$ or $C_1 \sqcup C_2$, then $\text{sub}(D) = \{D, C_1, C_2\}$
3. Otherwise $\text{sub}(D) = \{D\}$

$\text{Sub}(D)$ is the closure over the sub-expressions of $D$ and their negations (NNF forms) [6]. For a concept $C$, and $\mathcal{K} = \langle T, \mathcal{R}, \mathcal{A} \rangle$, we define:

\[
\text{clos}(C, \mathcal{R}) := \text{Sub}(C) \cup \{\forall B_S(q), D | \forall S.D \in \text{Sub}(C) \text{ and } B_S \text{ has a state } q\}
\]

\[
\text{clos}(T, \mathcal{R}) := \bigcup_{I \in \text{Def}(T)} \text{clos}(\text{NNF}(\neg C \sqcap D), \mathcal{R})
\]

For the following we assume $s, t \in \hat{V}_a^T, C, C_1, C_2 \in \text{clos}(T, \mathcal{R})$, unless mentioned otherwise. Further, for a role $S \in N_R$, we define the set of successors of a node $s$ w.r.t. $S$, with $C$ in their label set as:

\[
S^T(s, C) := \{t \in \hat{V}_a^T | \langle s, t \rangle \in E^T_a(S); C \in L(t)\},
\]
For a path \( U = R_1 \ldots R_k g \), a node \( y \in V^T \) is called a \( U \) successor of a node \( x \in V^T \) if there exist \( e_1, \ldots, e_{k+1} \in V^T \) such that \( (x, e_1) \in E^T_a(R_1), (e_i, e_{i+1}) \in E^T_a(R_i) \) for \( 2 \leq i \leq k-1 \) and \( (e_k, y) \in E^T_a(g) \).

A valid augmented tableau \( T_a = (T, N) \), for \( K \), must satisfy the propositions below. Propositions relating to abstract domain constructs (denoted by \( P_a \)) are borrowed from \( SROIQ \) and included here for completeness. The propositions relating to the concrete domain constructs are denoted by \( P_c \).

- \((P_a \sqsubseteq)\): If \( (C \sqsubseteq D) \in T \), then \( NNF(\neg C \sqcup D) \in \mathcal{L}^T(s) \) for all \( s \in V_a^T \)
- \((P_a \neg)\): If \( C \in \mathcal{L}^T(s) \), then \( \neg C \notin \mathcal{L}^T(s) \) (\( C \) atomic or \( \exists R.S elf \) or, \( \top \) or, \( \bot \))
- \((P_a Self)\): If \( \exists S.S elf \in \mathcal{L}^T(s) \), then \( \langle s, s \rangle \in E^T_a(S) \)
- \((P_a \neg\exists S elf)\): If \( \neg \exists S.S elf \in \mathcal{L}^T(s) \), then \( \langle s, s \rangle \notin E^T_a(S) \)
- \((P_a \sqcap)\): If \( C_1 \sqcap C_2 \in \mathcal{L}^T(s) \), then \( C_1 \in \mathcal{L}^T(s) \) and \( C_2 \in \mathcal{L}^T(s) \)
- \((P_a \sqcup)\): If \( C_1 \sqcup C_2 \in \mathcal{L}^T(s) \), then \( C_1 \in \mathcal{L}^T(s) \) or \( C_2 \in \mathcal{L}^T(s) \)
- \((P_a \exists R.C)\): If \( \exists R.C \in \mathcal{L}^T(s) \) then there is some \( t \) with \( \langle s, t \rangle \in E^T_a(R) \) and \( C \in \mathcal{L}^T(t) \)
- \((P_a \forall R.C)\): If \( \forall R.C \in \mathcal{L}^T(s) \) and \( \langle s, t \rangle \in E^T_a(R) \) then \( C \in \mathcal{L}^T(t) \)
- \((P_a Inv)\): \( \langle x, y \rangle \in E^T_a(R) \) iff \( \langle y, x \rangle \in E^T_a(Inv(R)) \)
- \((P_a \leq)\): If \( (\leq n S.C) \in \mathcal{L}^T(s) \), then \( \# S^T(s, C) \leq n \)
- \((P_a \geq)\): If \( (\geq n S.C) \in \mathcal{L}^T(s) \), then \( \# S^T(s, C) \geq n \)
- \((P_a \leq Neighbour)\): If \( (\leq n S.C) \in \mathcal{L}^T(s) \), and \( \langle s, t \rangle \in E^T_a(S) \), then \( C \in \mathcal{L}^T(t) \) or \( \text{nff}(\neg C) \in \mathcal{L}^T(t) \)
- \((P_a Dis)\): If \( \text{Dis}(R, S) \in \mathcal{R}_a \), then \( E^T_a(R) \cap E^T_a(S) = \emptyset \)
- \((P_a Ref)\): If \( \text{Ref}(S) \in \mathcal{R}_a \), then \( \langle s, s \rangle \in E^T_a(S) \) for all \( s \in V_a^T \)
- \((P_a Subrole)\): If \( \langle s, t \rangle \in E^T_a(R) \) and \( R \sqsubseteq S \) then \( \langle s, t \rangle \in E^T_a(S) \)
- \((P_a Nom)\): \( o \in \mathcal{L}^T(s) \) for some \( s \in V_a^T \), for each \( o \in \text{Nom} \cap \text{clos}(C_0, \mathcal{R}) \)
- \((P_a NomUng)\): If \( o \in \mathcal{L}^T(s) \) for some \( o \in \text{Nom} \) then \( s = t \)
- \((P_a \mathcal{B} \text{ transition})\): if \( \forall \mathcal{B}(p).C \in \mathcal{L}(s), \langle s, t \rangle \in E^T_a(R) \), and \( (p \xrightarrow{R} q) \in \mathcal{B}(p) \), then \( \forall \mathcal{B}(q).C \in \mathcal{L}^T(t) \)
- \((P_a \mathcal{B} \epsilon)\): if \( \forall \mathcal{B}.C \in \mathcal{L}(s) \), and \( \epsilon \in I(\mathcal{B}) \), then \( C \in \mathcal{L}^T(s) \)
- \((P_c \forall c U)\): If \( (\forall c U_1, U_2, r) \in \mathcal{L}^T(s) \), then for all \( x, y \) such that \( x \) is a \( U_1 \) successor and \( y \) is a \( U_2 \) successor of \( s \), we have \( (x r y) \in N \).
- \((P_c \exists c U)\): If \( (\exists c U_1, U_2, r) \in \mathcal{L}^T(s) \), then there must exist \( x, y \in V^T_c \) such that \( x \) is a \( U_1 \) successor and \( y \) is a \( U_2 \) successor of \( s \) and \( (x r y) \in N \).
• \((P, \forall, i)\): If \((\forall, U_1, \{i\}, r) \in \mathcal{L}^T(s)\), then for all \(x\) such that \(x\) is a \(U_1\) successor of \(s\) and \(i \in N_{s, i}\), we have \((x \ r \ i) \in N\). The symmetric case of \((\forall, i), U_1, r)\) is handled analogously.

• \((P, \exists, i)\): If \((\exists, U_1, \{i\}, r) \in \mathcal{L}^T(s)\), then there must exist \(x \in V_c^T\) such that \(x\) is a \(U_1\) successor of \(s\) and \((x \ r \ i) \in N\), where \(i \in N_{s, i}\). The symmetric case of \((\exists, i), U_1, r)\) is handled analogously.

• \((P, \neg \exists, U)\): If \((\neg \exists, U_1, U_2, r) \in \mathcal{L}^T(s)\), then there must not exist any \(y, y' \in V_c^T\) such that \(y\) is a \(U_1\) successor of \(s\) and \((y \ r \ y') \in N\).

• \((P, \neg \forall, i)\): If \((\neg \forall, U_1, \{i\}, r) \in \mathcal{L}^T(s)\), then there must not exist any \(y \in V_c^T\) such that \(y\) is a \(U_1\) successor of \(s\), and \((y \ r \ i) \in N\).

• \((P, \leq)\): If \((\leq n) \in \mathcal{L}^T(s)\), then \(#\{c_i | c_i \in V_c^T, (s, c_i) \in E^T_c(g)\} \leq n\).

• \((P, \geq)\): If \((\geq n) \in \mathcal{L}^T(s)\), then \(#\{c_i | c_i \in V_c^T, (s, c_i) \in E^T_c(g)\} \geq n\).

• \((P, \text{Subrole})\): If \((y_1, y_2) \in E_a^U(g)\) and \(g \sqsubseteq g'\) then \((y_1, y_2) \in E_a^U(g')\)

• \((P, \text{NSat})\): \(N\) should be satisfiable.

**Proposition:** There exists an augmented tableau for a SROIQ(C) KB iff there exists a model for it.

### 2.3 Tableau Algorithm for SROIQ(C)

The Tableau algorithm generates a completion system \(S = (G, N, Q)\), where \(G = (V_a^S, V_c^S, E_a^S, E_c^S, L^S, M^S, \#)\) is a completion graph, \(N\) is a finite constraint network with \(V_N = V_c^S\) and \(Q\) is the constraint template set. \(V_a^S\) is the set of abstract nodes, \(V_c^S\) is the set of concrete nodes. \(\text{Nom}(S) \subset V_a^S\) is the set of nominal nodes and represent named individuals. \(M^S\) labels each of the concrete nodes of \(S\) with a set of marker symbols. \(E_a^S\) is set of abstract edges of the form \((a, b)\), where \(a, b \in V_a^S\). \(E_c^S\) is set of concrete edges of the form \((a, c)\), where \(a \in V_a^S, c \in V_c^S\).

The completion system is a finite graphical representation of a (possibly infinite) tableau, with \(L^S\) labelling nodes with concepts, and edges with roles.

#### 2.3.1 Preliminaries

Here we discuss the terminology required to introduce the tableau algorithm. Let KB be a SROIQ(C) knowledge base consisting of \((A, R, T)\). The algorithm assumes that the KB has been reduced. The algorithm is for deciding the satisfiability of a SROIQ(C) KB. However, other common inference problems can be reduced to KB satisfiability, as described earlier.
Neighbours and Successors  If \( (x, y) \in E_a \) or \( E_c \), then y is called a successor of x, and x is called a predecessor of y. Further, if \( R \in L^S((x, y)) \), then y is the successor of x wrt R. Ancestor is the transitive closure of predecessor, and descendant is the transitive closure of successor.

For \( x, y \in V_a, R \in N_ar, y \) is called an \( R \)-successor of a node \( x \) if, for some \( R' \in N_ar \) with \( R' \subseteq R, R' \in L^S(x, y) \). Similarly, for \( x \in V_a, y \in V_c, g \in N_cR, y \) is called an \( g \) successor of a node \( x \) if, for some \( g' \in N_cR \) with \( g' \subseteq g, g' \in L^S(x, y) \). It may be noted that this notion is different from that of successor wrt \( R \), defined above.

A node \( y \) is called a \( R \)-neighbour of a node \( x \), if \( y \) is a \( R \)-successor of \( x \) or if \( x \) is an \( Inv(R) \)-successor of \( y \). That is, \( x \) has an outgoing \( R \) edge going towards \( y \), or has a incoming \( Inv(R) \) edge coming from \( y \).

For a path \( U = R_1 \ldots R_kg \), a node \( c \in V_c \) is called a \( U \)-successor of a node \( a \in V_a \) if there exist \( e_1, \ldots, e_k \in V_a \) such that \( e_1 \) is the successor of \( a \) wrt \( R_1 \), \( e_i \) is the successor of \( e_{i-1} \) wrt \( R_i \) for \( 2 \leq i \leq k \) and \( c \) is the \( g \) successor of \( e_k \). For a path \( U = R_1 \ldots R_kg \), a node \( c \in V_c \) is called a \( U \)-successor of a node \( a \in V_a \) if there exist \( e_1, \ldots, e_k \in V_a \) such that \( e_1 \) is the \( R_1 \) neighbour of \( a \), \( e_i \) is the \( R_i \) neighbour of \( e_{i-1} \) for \( 2 \leq i \leq k \) and \( c \) is the \( g \) successor of \( e_k \). Unlike in completion system, we do not maintain any distinction between successor and neighbour in the augmented tableau because, as elaborated later, in the tableau all subrole and inverse related information is made explicit.

Marker Symbols and Constraint Template Set  The constraint template set \( Q \) is a set of expressions of the form \( (q_1 \ r \ q_2) \) (the positive template) or \( \neg(q_1 \ r \ q_2) \) (the negative template), where \( q_1, q_2 \) are marker symbols and \( r \in Rel \). It may be noted that it is these marker symbols which label a concrete node using \( M^S \). Let \( c_1, c_2 \) be any two concrete nodes with \( q_1 \in M^S(c_1), q_2 \in M^S(c_2) \). If \( (q_1 \ r \ q_2) \in Q \), then the completion rules add a constraint \( (c_1 \ r \ c_2) \) to \( N \). In comparison, if \( \neg(q_1 \ r \ q_2) \in Q \), then the presence of \( (c_1 \ r \ c_2) \) in \( N \) leads to a clash. The negative template is used with clash conditions and completion rules to ensure the semantics of concepts of the form \( \neg\exists U_1, U_2, r \) or \( \neg\exists U_1, [i], r \) or \( \neg\exists [i], U_1, r \).

Blocking  Without a blocking mechanism, the completion rules may produce a infinite completion graph composed of repeating units, which would compromise the termination of the tableau algorithm. The presented blocking scheme is an extension of the one used for \( SROIQ \).

A non-nominal abstract node of \( S \) is called a blockable node. In order for blocking to occur, a series of blocking checks need to be passed successfully. We take up them in the following paragraphs.

Blocking Check 1 (BC-1)  Let \( a, b, a_p, b_p \in V_a \). a and its descendant \( b \) are said to pass BC-1, if all of the following conditions are met:

1. \( a_p \) is a predecessor of \( a \), \( b_p \) is a predecessor of \( b \) and \( a, a_p, b_p \) are the ancestors of \( b \)
2. \( a, a_p, b, b_p \) are distinct, and all nodes on the path from \( a \) to \( b \) are blockable
3. \( L^S(a) = L^S(b) \) and \( L^S(a_p) = L^S(b_p) \)
4. \( L^S((a_p, a)) = L^S((b_p, b)) \)
This check ensures that there is indeed a repetition in the abstract node backbone. This check is the direct adaptation from SROIQ and is required since SROIQ(C) contains all the constructs of SROIQ. The distinctness of the nodes just ensures that there is at least one node (and two edges) between the blocker and the directly blocked node, and is used for arguing the correctness of the algorithm intuitively.

**Blocking Check 2 (BC-2)** For the above mentioned nodes, we define the following:

- \( relevantCNodes(a) = \{ c \mid c \in V^S_c; \text{c is the successor of } a \text{ or } a_p \} \)
- \( relevantCNodes(b) = \{ c \mid c \in V^S_c; \text{c is the successor of } b \text{ or } b_p \} \)
- \( relevantCNet(a) = \{(c \ r \ c') \mid \{c, c'\} \subseteq relevantCNodes(a); (c \ r \ c') \in N\} \)
- \( relevantCNet(b) = \{(c \ r \ c') \mid \{c, c'\} \subseteq relevantCNodes(b); (c \ r \ c') \in N\} \)

We define an injective mapping \( \phi : relevantCNodes(a) \rightarrow relevantCNodes(b) \) such that

- for every \( c \in relevantCNodes(a) \) which is the successor of \( a_p \), \( L^S(\langle a_p, c \rangle) = L^S(\langle b_p, \phi(c) \rangle) \)
- for every \( c \in relevantCNodes(a) \) which is the successor of \( a \), \( L^S(\langle a, c \rangle) = L^S(\langle b, \phi(c) \rangle) \)
- for every \( c \in relevantCNodes(b) \) which is the successor of \( b_p \), \( L^S(\langle b_p, c \rangle) = L^S(\langle a_p, \phi^-(c) \rangle) \)
- for every \( c \in relevantCNodes(b) \) which is the successor of \( b \), \( L^S(\langle b, c \rangle) = L^S(\langle a, \phi^-(c) \rangle) \)

In the above discussion \( \phi^- \) is the inverse mapping of \( \phi \).

A blockable node \( a \) and its descendant \( b \) pass BC-2 if:

- they pass BC-1
- the mapping \( \phi \) exists
- for every \( c, c' \in relevantCNodes(a) : (c \ r \ c') \text{ iff } (\phi(c) \ r \ \phi(c')) \)
- for every \( c, c' \in relevantCNodes(b) : (c \ r \ c') \text{ iff } (\phi^-(c) \ r \ \phi^-(c')) \)

This check is required to ensure that when the blocked region is unravelled into an infinite structure while constructing the tableau, then the semantics of the concrete concepts are indeed enforced. We return to this in the soundness proof.

**Blocking Check 3 (BC-3)** A node \( a_2 \) is a strict descendant of \( a_1 \), if it is a non-nominal descendant of \( a_1 \) and can be reached from \( a_1 \) without encountering a nominal node on the connecting path. This notion of ”strict descendant” differs from ”descendant” because unlike the latter, it does not transitively extend to nodes whose path to \( a_1 \) has a nominal in it.
For nodes $a$ and $b$, which pass BC-2, we define the following terms:

$$\text{internalCNodes}(a, b) = \{ c \mid c \in V^S_c; \text{c is a strict descendant of } a \text{ but not of } b \}$$

$$\text{externalCNodes}(a, b) = \{ c \mid c \notin \text{internalCNodes}(a, b); c \notin \text{strict descendant}(b); \text{for some } c' \in \text{internalCNodes}(a, b)$$

either $\langle c, r, c' \rangle \in \mathcal{N}$ or $\langle c', r, c \rangle \in \mathcal{N} \}$$

$$\text{associatedCNodes}(b) = \{ c \mid c \in V^S_c \text{ is the successor of } b \}$$

$$\text{associatedCNodes}(a) = \{ c \mid c \in V^S_c \text{ is the successor of } a \}$$

$$\text{relativeCNodes}(a, b) = \text{associatedCNodes}(a)$$

$$\text{fixedCNodes}(a, b) = \text{externalCNodes}(a, b) - \text{relativeCNodes}(a, b)$$

We could alternatively define $\text{fixedCNodes}(a, b)$ as:

$$\text{fixedCNodes}(a, b) = \{ c \mid c \text{ is the successor of a nominal node } r; \text{ and for some } c' \in \text{internalCNodes}(a, b)$$

either $\langle c, r, c' \rangle \in \mathcal{N}$ or $\langle c', r, c \rangle \in \mathcal{N} \}$$

Though the requirement for the above definitions will become clearer in the soundness proof, we explain the intuition behind the above sets. If $a$ and $b$ pass the remaining tests, and $a$ blocks $b$, then the nodes between them would form a "unit". Units isomorphic / similar in structure to this unit would get repeated over and again to form an infinite structure when the completion system is unravelled into a tableau. The concrete nodes belonging to one such unit in the tableau would map back to the nodes of $\text{internalCNodes}$. The concrete nodes of one such unit may enter into constraints with the nodes of the unit immediately before it. These nodes of the preceding unit map back to $\text{associatedCNodes}(b)$. Also, sometimes, the nodes of different units all form constraints with the same node, external to all of them. The set of such external concrete nodes maps back to $\text{fixedCNodes}(a, b)$.

We define:

$$\text{cNetNodes}(a) = \text{associatedCNodes}(a) \cup \text{fixedCNodes}(a, b)$$

$$\text{cNetNodes}(b) = \text{associatedCNodes}(b) \cup \text{fixedCNodes}(a, b)$$

We define a mapping $\theta : \text{cNetNodes}(a) \to \text{cNetNodes}(b)$, as follows:

$$\theta(c) = \begin{cases} 
\phi(c) & \text{if } c \in \text{associatedCNodes}(a) \\
\phi(c) & \text{if } c \in \text{fixedCNodes}(a, b)
\end{cases}$$

On account of BC-2, this exists.

We call the constraint networks formed by variables in $\text{cNetNodes}(a)$ as $\text{cNet}(a)$ and those by the variables in $\text{cNetNodes}(b)$ as $\text{cNet}(b)$. $\text{cNet}(a)$ and $\text{cNet}(b)$ are said to be isomorphic if for $c, c' \in \text{cNetNodes}(a)$ and $(c, r, c') \in \mathcal{N}$, we have $(\theta(c), r, \theta(c')) \in \mathcal{N}$.
A blockable node $a$ and its descendant $b$ pass BC-3 if the following holds:

- they pass BC-2
- the constraint networks $cNet(a)$ and $cNet(b)$ are complete and isomorphic.

If $a$ and its descendant $b$ pass all the checks, $a$ is said to directly block $b$. $a$ is termed the blocker, $b$ and its descendants are said to be blocked. The notion of indirectly blocked is as defined in SROIQ.

This check is required to be able to apply the patchwork property of the constraint system, in order to argue for the satisfiability of the constraint network of the unravelled augmented tableau.

**Merging and pruning** The merging carried out here is similar to [5].

The following steps must be carried out when merging abstract node $y$ into $x$:

1. If $(x, c) \in E^S_c$, add $g$ to $L^S((x, c))$ i.e. if $x$ already has a edge with $c$ (only way this could happen is if $c \in N_{cI}$), then the label set of this edge is augmented with that from $(y, c)$.

2. Else, create a new edge between $x$ and $c$, and set $L^S((x, c)) = L^S((y, c))$

Informally, we move all the concrete node successors of $y$ to now be the successors of $x$.

We point out that while merging an abstract node ($a$) into another ($b$), we drop/remove all the abstract edges outgoing from $a$; but we have, however, retained the concrete edges. This is required to avoid a threat to termination. Consider that a abstract successor to node $a$ w.r.t $R$, $a_s$, created by completion rules acting on $(\exists, Rg, g.r) \in L^S(a')$. Assume $c$ is the $g$ successor of $a_s$. If it so happens that $a_s$ is merged back into $a$, then if we drop the concrete edges then we return to exactly the same state that existed before the completion rule acted on $(\exists, Rg, g.r) \in L^S(a')$, and this ”loop” could compromise termination. Hence, we retain the concrete edges and nodes.

Consider the case of merging a concrete node $c_1$ into another concrete node $c_2$, where there exists no relation $c_1 \neq c_2$. The constraints of the representation scheme ensure that $c_1$ and $c_2$ are the successors of the same abstract node, and are not both constraint individuals simultaneously. The following steps must be performed:

1. For $a \in V^S_a$ such that $(a, c_1) \in E^S_c$, the following steps are performed:
   
   (a) Set $L^S((a, c_2)) = L^S((a, c_2)) \cup L^S((a, c_1))$
   
   (b) Remove $(a, c_1)$ from $E^S_c$

2. Set $M^S(c_2) = M^S(c_1) \cup M^S(c_2)$

3. Rename $c_1$ to $c_2$ in all constraints in $N$.

4. Add $c_2 \neq c'$ for all $c'$ such that $c' \neq c_1$

Pruning is done similar to SROIQ, but when an abstract node is pruned, the concrete nodes ($\not\in N_{cI}$) successors of the nodes are also pruned from $S$. 
Clash Conditions For completeness, the clash conditions of SROIQ are mentioned here. These are adapted directly into our algorithm, since SROIQ(C) contains all constructs of SROIQ.

1. There exists $a \in V^S_a$, such that $\bot \in L^S(a)$
2. There exists $a \in V^S_a$, such that $\{C, \neg C\} \subseteq L^S(a)$
3. $\neg \exists S.Self \in L^S(a)$ and $S \in L^S((a, a))$.
4. There is some $Dis(S, S') \in R_a$ and for some $a, b \in V^T_a$, $b$ is both $S$ and $S'$ neighbour of $a$.
5. There is some concept $\leq nS.C \in L^S(a)$, and there exist more than $n S$ neighbours of $a$.
6. For some nominal $o \in Nom$, $o \in L^S(a) \cap L^S(b)$ for distinct $a \neq b$.

Apart from the above, the completion system is said to contain a clash in either of the following cases:

1. There exists $a \in V^S_a$, such that $\leq g \in L^S(a)$, and there exist $c_1 \ldots c_k \in V^S_c$ such that $k > n$, $c_i \neq c_j$ for $1 \leq i < j \leq k$ and each $c_i$ is a $g$ successor of $a$ for $1 \leq i < k$
2. There exist $c_1, c_2 \in V^S_c$, such that $(c_1 r c_2) \in N$ even though $\neg (q_1 r q_2) \in Q$ for some $q_1 \in M^S(c_1), q_2 \in M^S(c_2)$.
3. $N$ is not satisfiable

2.3.2 Algorithm Initialization

If $o_1, \ldots, o_K \in Nom$, then the tableau algorithm starts with the completion system $S = (G, N, Q)$, where $G = (V^S_a, V^S_c, E^S_a, E^S_c, L^S, M^S, \neq)$. Here, $V^S_a = \{r_1, \ldots, r_K\}$, where $L^S(r_k) = \{o_k\}$ for $1 \leq k \leq K$, and $o_k = \{p_k\}$ for $p_k \in N_{al}$ being an named individual. In case there are no nominals in the KB, then a seed node $r_0$ is the sole member of $V^S_a$.

For every A-Box assertion of the form $g(p_k, i_l)$, where $p_k \in N_{al}, i_l \in N_{cl}, \{p_k\} = o_k$ and $o_k \in L^S(r_k)$, do the following: add a concrete node named $i_l$ to $V^S_c$, add an edge $(r_k, i_l)$ to $E^S_c$ and set $L^S((r_k, i_l)) = \{g\}$. Further, for every assertion of the form $(i_1 r i_2)$, add $(i_1, r, i_2)$ to $N$. For all such concrete nodes $i_1 \ldots i_L \in N_{cl} \cap V^S_c$, add $i_l \neq i_m$ for $1 \leq (l, m) \leq k$.

$G$ is then expanded by repeatedly (and non-deterministically) applying the applicable expansion(completion) rules. This stops if either a clash occurs or if no more rules are applicable, in which case $S$ is said to be complete. The completion system $S$, if complete and clash-free, can be unravelled to form an augmented tableau for the Knowledge Base $(\mathcal{A}, \mathcal{T}, \mathcal{R})$. The algorithm returns a complete and clash-free completion system, iff the KB is consistent.

2.3.3 The Completion Rules

For completeness we first mention the completion rules of SROIQ, all of which are valid completion rules in the tableau procedure for SROIQ(C).

Unless mentioned otherwise, assume, $C, D, C_1, C_2$ are concepts. $a, b \in V^S_a$; $c, c_1, c_2 \in V^S_c$; $i, i_1, i_2 \in N_{cl}$; $R, R' \in N_{ar}$ are abstract roles; $S, S'$ are simple abstract roles; $U_1, U_2$ are paths;
Analogously, for concrete node creation.

\[ g, g' \in N_{\text{cl}} \] are concrete roles. \( G, G' \) are paths, role chains or \( \epsilon \). \( q \) is a \textit{marker} and \( q_{\text{a}} \) is a marker symbol new to \( S \). When creating a new node \( a \) which is an \( R \) successor of existing node \( b \), we use \( \mathcal{L}^S((b, a)) = \{ R \} \) to imply \( V^S_a = V^S_b \cup \{ a \} \) and \( E^S_a = E^S_b \cup \{ (b, a) \} \) in addition to \( \mathcal{L}^S((b, a)) = \{ R \} \). Analagously, for concrete node creation.

- **R∪:** If \( C_1 \cap C_2 \subseteq \mathcal{L}^S(a) \), \( a \) is not blocked, and \( \{ C_1, C_2 \} \not\subseteq \mathcal{L}^S(a) \), then \( \mathcal{L}^S(a) \to \mathcal{L}^S(a) \cup \{ C_1, C_2 \} \)
- **R∩:** If \( C_1 \cup C_2 \subseteq \mathcal{L}^S(a) \), \( a \) is not blocked, and \( \{ C_1, C_2 \} \cap \mathcal{L}^S(a) = \emptyset \), then \( \mathcal{L}^S(a) \to \mathcal{L}^S(a) \cup \{ E \} \) for some \( E \in \{ C_1, C_2 \} \)
- **R gci:** If \( C \subseteq D \in T \) and \( (\neg C \cup D) \not\subseteq \mathcal{L}^S(a) \), then set \( \mathcal{L}^S(a) := \mathcal{L}^S(a) \cup \mathrm{NNF}(\neg C \cup D) \)
- **R Self-Ref:** If \( \exists S. C \in \mathcal{L}^S(a) \), \( a \) is not blocked, and \( a \) has no \( R \)-neighbour \( b \) with \( C \in \mathcal{L}^S(b) \), then create a new node \( b \) with \( \mathcal{L}(\langle a, b \rangle) := \{ R \} \) and \( \mathcal{L}^S(b) := \{ C \} \)
- **R V1:** If \( \forall S. C \in \mathcal{L}^S(a) \), \( a \) is not indirectly blocked, and \( \forall \mathcal{B}_S. C \not\in \mathcal{L}^S(a) \), then \( \mathcal{L}^S(a) \to \mathcal{L}^S(a) \cup \{ \forall \mathcal{B}_S.C \} \)
- **R V2:** If \( \forall \mathcal{B}(p). C \in \mathcal{L}^S(a) \), \( a \) is not indirectly blocked, and \( p \overset{S}{\to} q \) in \( \mathcal{B}(p) \) and there is an \( S \)-forward-neighbor \( b \) of \( a \) with \( \forall \mathcal{B}(q). C \not\in \mathcal{L}^S(b) \), then \( \mathcal{L}^S(b) \to \mathcal{L}^S(b) \cup \{ \forall \mathcal{B}_S(q).C \} \)
- **R V3:** If \( \forall \mathcal{B}. C \in \mathcal{L}^S(a) \), \( a \) is not indirectly blocked, \( \epsilon \in L(\mathcal{B}) \), and \( C \not\in \mathcal{L}^S(a) \), then \( \mathcal{L}^S(a) \to \mathcal{L}^S(a) \cup \{ C \} \)
- **R choose-rule:** If \( (\leq n S. C) \in \mathcal{L}^S(a) \), \( a \) is not blocked, and there is an \( S \)-neighbor \( y \) of \( x \) with \( \{ C, \neg C \} \cap L(y) = \emptyset \), then \( \mathcal{L}(y) \to \mathcal{L}(y) \cup \{ E \} \) for some \( E \in \{ C, \neg C \} \)
- **≥-rule:** If 1. \( (\geq n S. C) \in \mathcal{L}(z) \), \( x \) is not blocked, and 2. There are not \( n \) safe \( S \)-forward-neighbors \( y_1, \ldots, y_n \) of \( a \) with \( C \in L(y_i) \) and \( y_i \neq y_j \) for \( 1 \leq i < j \leq n \), then create \( n \) new nodes \( y_1, \ldots, y_n \) with \( L(\langle a, y_i \rangle) = \{ S \}, L(y_i) = \{ C \}, \) and \( y_i \neq y_j \) for \( 1 \leq i < j \leq n \)
- **≤-rule:** If 1. \( (\leq n S. C) \in L(z) \), \( z \) is not blocked, and 2. \( \#S^G(z, C) > n \) and there are two \( S \)-neighbors \( x, y \) of \( z \) with \( C \in L^S(a) \cap L(y) \) and \( x \neq y \) then 1. If \( x \) is a nominal node, then \text{Merge}(y, x) \) 2. Else if \( y \) is a nominal node or an ancestor of \( x \), then \text{Merge}(x, y) \) 3. Else \text{Merge}(y, x) \)
- **o-rule** If for some \( o \in N_f \) there are 2 nodes \( a, b \) with \( o \in \mathcal{L}^S(a) \cap \mathcal{L}^S(b) \) and not \( a \neq b \) then \text{Merge}(a, b) \)
• NN-rule: If $1. (\leq n S. C) \in L^S(a)$, $a$ is a nominal node, and there is a blockable $S$-forward-
neighbor $b$ of $a$ such that $C \in L^S(b)$ and $a$ is a successor of $b$
2. There is no $m$ such that $1 \leq m \leq n, (\leq m S. C) \in L^S(a)$ there exist $m$ nominal $S$-forward-neighbors $z_1, \ldots, z_m$ of $a$ with $C \in L(z_i)$ and $z_i \neq z_j$ for all $1 \leq i < j \leq m$ then
1. Guess $m$ with $1 \leq m \leq n$ and set $L^S(a) = L^S(a) \cup \{(\leq m S. C)\}.$
2. Create $m$ new nodes $y_1, \ldots, y_m$ with $L((a, y_i)) = \{S\}, L(y_i) = \{C, o_i\}$ for each $o_i \in N_l$ new in $G$, and $y_i \neq y_j$ for $1 \leq i < j \leq m$

Assume $c \in V^S_e$, $q, q_1, q_2$ are markers new to $S$. For the rules below, by ”create a $U = Rg$
w.l.o.g.) successor of $a^*$”, we mean that we create a successor of $a$ w.r.t $R$ (say $a'$) and then create
a $g$ successor of $a'$ (say $c$), such that $\langle a, a' \rangle = |R|$ and $\langle a', c \rangle = |g|$.

• $\exists_i U$: If $(\exists_i U, 1, U_2, r) \in L^S(a)$, $a$ is not blocked, and there are no $U_1$ and $U_2$ neighbourhoods of $a$ (say $c_1$ and $c_2$) such that $q_1 \in M^S(c_1), q_2 \in M^S(c_2)$ and $(q_1, r q_2) \in Q,$
Then, create a $U_1$ successor of $a$ (say $c_1$) and a $U_2$ successor of $a$ (say $c_2$), and add $(q_1, r q_2)$ to $Q.$ $q_1$ to $M^S(c_1)$ and $q_2$ to $M^S(c_2)$.

• $\forall_i U$: If $(\forall_i U, 1, U_2, r) \in L^S(a)$, $a$ is not blocked, there exist $U_1$ and $U_2$ neighbourhoods of $a$ (say $c_1$ and $c_2$), but there exist no markers $q_1$ and $q_2$, such that $q_1 \in M^S(c_1), q_2 \in M^S(c_2)$ and $(q_1, r q_2) \in Q,$
Then, add $q_1$ to $M^S(c_1), q_2$ to $M^S(c_2)$ and $(q_1, r q_2) \in Q.$

• $\exists_i i$: If $(\exists_i U, 1, i, r) \in L^S(a)$, $a$ is not blocked, and there is no $U_1$ neighbour of $a$ (say $c_1$) such that $q_1 \in M^S(c_1), q_2 \in M^S(i)$ and $(q_1, r q_2) \in Q,$
Then, create a $U_1$ successor of $a$ (say $c_1$) and add $q_1$ to $M^S(c_1), q_2$ to $M^S(i)$ and $(q_1, r q_2)$ to $Q.$ Analogously for the symmetric case of $(\exists_i, 1, U_1, r)$.

• $\forall_i i$: If $(\forall_i U, 1, i, r) \in L^S(a)$, $a$ is not blocked, there exists a $U_1$ neighbour of $a$ (say $c_1$), but there exist no markers $q_1$ and $q_2$, such that $q_1 \in M^S(c_1), q_2 \in M^S(i)$ and $(q_1, r q_2) \in Q,$
Then, add $q_1$ to $M^S(c_1), q_2$ to $M^S(i)$ and $(q_1, r q_2)$ to $Q.$ Analogously for the symmetric case of $(\forall_i, 1, U_1, r)$.

• $R \neg \exists_i U$: If $\neg(\exists_i U, 1, U_2, r) \in L^S(a)$, $a$ is not blocked, there exist $U_1$ and $U_2$ neighbours of $a$ (say $c_1$ and $c_2$), but there exist no markers $q_1$ and $q_2,$ such that $q_1 \in M^S(c_1), q_2 \in M^S(c_2)$ and $\neg(q_1, r q_2) \in Q,$
Then, add $q_1$ to $M^S(c_1), q_2$ to $M^S(c_2)$ and $\neg(q_1, r q_2) \in Q.$

• $R \neg \exists_i i$: If $\neg(\exists_i U, 1, i, r) \in L^S(a)$, $a$ is not blocked, there exist a $U_1$ neighbours of $a$ (say $c_1$), but there exist no markers $q_1$ and $q_2,$ such that $q_1 \in M^S(c_1), q_2 \in M^S(i)$ and $\neg(q_1, r q_2) \in Q,$
Then, add $q_1$ to $M^S(c_1), q_2$ to $M^S(i)$ and $\neg(q_1, r q_2) \in Q.$

• $R \geq$: If $(\geq, n, g) \in L^S(a)$ and there do not exist $n g$ successors of $a, c_1, \ldots, c_n$ such that $c_i \neq c_j$ for $1 \leq i < j \leq n$
Then create $n$ new concrete nodes $c_1, \ldots, c_n$ with $L^S((a, c_i)) = \{g\}$ and $c_i \neq c_j$ for $1 \leq i < j \leq n.$

• $R \leq$: If $(\leq, n g) \in L^S(a), a$ is not blocked, there exist more than $n g$-successors of $a$, and there are two $g$-successors $c_1, c_2$ of $a$ without $c_1 \neq c_2.$
Then, If $c_1 \in N_{cI}$, then $\text{Merge}(c_2, c_1)$; else if $c_2 \in N_{cI}$, then $\text{Merge}(c_1, c_2)$; else, either $\text{Merge}(c_1, c_2)$ or $\text{Merge}(c_2, c_1)$.

- **Rconnect:** If $(q_1 \rightarrow r q_2) \in Q$ and there are nodes $c_1, c_2$ such that $q_1 \in M^S(c_1)$, $q_2 \in M^S(c_2)$, $\text{Symbol}(q_1) = q_1'$ and $\text{Symbol}(q_2) = q_2'$, then add $(c_1 \rightarrow r c_2)$ to $N$.

- **Rcomplete:** If abstract node $a$ and its descendant $b$, pass the BC-2, then guess completions for $cNet(a)$ and $cNet(b)$.

We elaborate on the design of the above algorithm. While in rules like $R\exists I$, we have created the successors in one go, still, we have not connected the concrete nodes in a constraint straight-away. We do it indirectly by inserting markers, and the completion rule $R\text{connect}$ adds constraints depending upon markers. This strategy was required to ensure the case handled by $R\neg\exists U$ and $R\neg\exists I$.

### 2.3.4 Strategy of Rule Application

The tableau algorithm is initialized as described. The expansion rules are applied according to the following order of priority:

- If there is a clash, return unsatisfiable
- If the completion system is complete, return satisfiable
- If $R\text{connect}$ is applicable, it is applied with next highest priority
- If $R\text{complete}$ is applicable, it is applied with next highest priority
- If the $\alpha$-rule, is applicable, apply with next highest priority
- If the $\leq$- and the $NN$- rule is applicable, then apply it first to nominal nodes with lower levels (before they are applied to nodes with higher levels). In case they are both applicable to the same node, the $NN$- rule is applied first.
- All other rules are applied with lower priority

It may be noted that the relative ordering of the $SROIQ$ rules is still maintained. Rules are applied preferably to nodes of lower levels, as explained in $SROIQ$.

*The KB satisfiability of a $SROIQ(C)$ KB is sound, terminating and complete.* For sketches of the corresponding proofs, the reader is directed to the attached Appendix.

### 2.4 Issues with Extensions

Thus we have presented a logic which provides constraint modelling constructs to $SROIQ$. However, we may want to have even more expressive constraint modelling constructs in the logic.
However, we considered several of these expressive constructs and realised that they pose unavoidable, fundamental problems. These are ultimately related to the patchwork property.

In the current logic, when we have a blocking in $S$, while constructing the augmented tableau, we unravel the blocked region into a infinite structure of repeating units. When we construct the constraint network of the tableau $N$, we do it incrementally, one unit at a time. Each unit causes the introduction of some new constraints into $N$. Currently, nodes of these units form constraints with either the nodes of the preceding unit or with a finite set of concrete safe simplex nodes. These nodes form the common variables between the constraints that are associated with a unit, and the existing network $N$. Thus, this is a finite set, and we can ensure that the network formed by these common variables is complete, by way of being isomorphic to a network in the completion system ($cNet$). Thus we can safely apply the patchwork property (among others) to argue that the infinite constraint network associated with these repeating nodes is satisfiable.

**Non simple roles**

Consider for example, the restriction that the abstract role used in a path must be a simple role. Let us see why this causes problems. If the role is not restricted to be simple, then it can, for eg, be transitive. This means that the $R$ successor of some node could be any number of units away from it. It could happen that a concrete node in one repeating unit forms constraints with the concrete nodes of all the units above it. Thus we can no longer ensure that the common variables form a complete network, and hence cannot apply the patchwork property. Thus we can no longer ensure that $N$ is satisfiable.

**Abolish PNF form**

Consider another extension. Though the PNF form seems to allow us to model many of the constraints that one faces in real domains, it does seem pretty restrictive. So, let's see what happens if we allow this. We know that different units can have abstract role edges to nominal tableau nodes (say $R^-$ edges to node $e$). Now if $\forall Rg, Rg.r \in L^T(e)$, then potentially we can again have constraints between the concrete nodes of repeating units arbitrarily far away. We again can't use the patchwork property and lose out on satisfiability of $N$.

Perhaps a new class of constraint systems may be able to work with these conditions too, but that is left for later analysis.
Chapter 3

Grounded Circumscription for SROIQ

3.1 A decision procedure for $GC \cdot SROIQ(C)$

We present here, algorithms to implement Grounded Circumscription for $SROIQ(C)$. The algorithms are intuitive and applicable to any decidable logic as long as the logic supports GCIs, inverse and nominal constructs employed in the algorithms. Here we use $SROIQ(C)$ as the base logic to illustrate the algorithms.

A $GC \cdot SROIQ(C)$-KB is the tuple $(K, M)$ where $K$ is the $SROIQ(C)$ Knowledge Base (KB) and $M$ is the set of minimized predicates. The minimized predicates may be atomic concepts or roles. Let $N_I$ be the set of individuals, explicitly named in $K$. Let $Nom$ be the union of nominal concepts corresponding to the above named individuals.

$$Nom = \{a \mid a \in N_I\}$$

For any two models $I$ and $Q$ of $K$, $I$ is preferred over $Q$ w.r.t. $M$ (written $I <_M Q$), iff all of the following hold:

1. $\Delta^I = \Delta^Q$ and $a^I = a^Q$ for every $a \in N_I$
2. $W^I \subseteq W^Q$ for every $W \in M$; and
3. There exists a $W \in M$ such that $W^I \subset W^Q$

A model $I$ of $K$ is called a grounded model w.r.t. $M$ if all of the following hold:

1. $C^I \subseteq \{b^I \mid b \in N_I\}$ for each concept $C \in M$; and
2. $R^I \subseteq \{(a^I, b^I) \mid (a, b) \in N_I \times N_I\}$ for each role $R \in M$

An interpretation $I$ is a GC-model of $(K, M)$ if it is a grounded model of $K$ w.r.t. $M$, and $I$ is minimal w.r.t. $M$, i.e., there is no model $Q$ of $K$ with $Q <_M I$. A GC-$SROIQ(C)$-KB is said to be GC-satisfiable if it has a GC-model. A statement (GCI, concept assertion or role assertion) $\alpha$ is a logical consequence(a GC-inference) of $(K, M)$ if every GC-model of $(K, M)$ satisfies $\alpha$.

We have two Tableau procedures $InitTab$ and $minTab$ based on $Tableau1$ and $Tableau2$ of [4]. The $InitTab$ computes an initial grounded model of the knowledge base. Each iteration of the $minTab$ Tableau tries to produce a model (tableau) which is preferred over the model it was...
initialized with. These procedures are used in a coordinated way, as described by the algorithm `modelFinder`, to test the GC-satisfiability of the KB.

### 3.1.1 InitTab

We assume that all concepts which are not a part of \( M \), are allowed to freely vary. If it is not so, then we follow the procedure outlined in [4] to transform the GC Knowledge Base such that it has no fixed concepts. We assume Unique Name Assumption to be valid. We assume the concept \( C \) to be atomic. If \( C \) is not atomic, introduce a GCI \( A \equiv C \) and perform the inference procedure w.r.t. \( A \).

In the KB, let \( M_c \) represent the set of minimized predicates. Let \( M \) consist of \( C_i \) (minimized concepts) and \( R_j \) (minimized roles), for \( 1 \leq i \leq n \) and \( 1 \leq j \leq m \).

**Algorithm 1: InitTab**

**Input:** GC-SROIQ(C) Knowledge Base \((M_c, K)\)  
**Output:** Grounded model \( GM \) if it exists

\[
\begin{align*}
1 & \text{ For } C_i \in M_c \text{ assert } C_i \subseteq \text{Nom} \\
2 & \text{ For } R_j \in M_c \text{ assert } \exists R_j, \top \subseteq \text{Nom} \\
3 & \text{ For } R_j \in M_c \text{ assert } \exists R_j^\top, \top \subseteq \text{Nom} \\
4 & \text{ Run the SROIQ(C) tableau algorithm} \\
5 & \text{ if } \text{Clashes occur then} \\
6 & \quad \text{return } \text{Unsatisfiable} \\
7 & \text{else} \\
8 & \quad \text{return } \text{Grounded Model } GM
\end{align*}
\]

If InitTab produces a complete and clash-free completion system, then the resulting completion system is a grounded model for the GC-SROIQ(C) KB.

### 3.1.2 MinTab:

The algorithm is executed on success of initTab. Let the grounded model produced by initTab be \( GM \).

\( GM \) is given as input to minTab. minTab extracts the extensions of minimized predicates from the completion system. Let the extension of the concept \( C_i \) be \( C_i^{\text{ext}} \) for \( 1 \leq i \leq n \). Let the extension of the role \( R_j \) be \( R_j^{\text{ext}} \) for \( 1 \leq j \leq m \). From \( R_j^{\text{ext}} \) (the extension of the role \( R_j \)), we extract

- \( R_j^{\text{ext}}, \text{dom} \): the domain for the extension,
- \( R_j^{\text{ext}}, \text{range} \): the range for the extension.

For every \( p \in R_j^{\text{ext}}, \text{dom} \), we define \( R_j^{\text{ext}}, \text{range}, p \) as \( \{p' \mid (p, p') \in R_j^{\text{ext}}\} \).

**miniTab** algorithm involves performing the following for the KB \((K, M_c)\) :

1. Assert \( C_i \subseteq C_i^{\text{ext}} \) for each concept \( C_i \)
2. Assert $\exists R_j. \top \subseteq R^\text{ext, range}_j$ for each role $R_j$ (LHS represents the set of nodes with incoming $R$ edges)

3. Assert $\exists R_j. \top \subseteq R^\text{ext, domain}_j$ for each role $R_j$ (LHS represents the set of nodes with outgoing $R$ edges)

4. Assert for all role $R_j$ and for each $p \in R^\text{ext, domain}_j : R_j . \{ p \} \subseteq R^\text{ext, range}_j . p$ (LHS represents the set of nodes with incoming $R$ edges, where the edges start from the node $p$)

5. Activate the preference clash.

6. Run the $SROIQ(C)$ tableau algorithm.

We now describe the preference clash. Let $C^E$ be the extension of a concept $C \in M_c$ extracted from a (possibly incomplete) completion system which is currently being operated upon by completion rules as a part of the $\text{minTab}$ algorithm mentioned above. Similarly, we extract $R^E$. A preference clash is said to occur if for all $P \in M_c$, $P^\text{ext} = P^E$. It may be noted that the added clashes are in addition to the ones internal to the tableau algorithms. If any clash is encountered, $\text{minTab}$ backtracks in attempt to produce a clash-free completion system / model.

If $\text{minTab}$ produces a complete and clash-free completion system, then the resulting completion system represents a model preferred over the input model w.r.t. the circumscription pattern. If $\text{minTab}$ produces a complete and clash-free completion system, then this model is fed as an input to $\text{minTab}$ again. This continues until no more clash-free models can be obtained. This final model is a grounded circumscription model (GC-model) of $(K, M_c)$.

### 3.1.3 modelFinder algorithm

This algorithm specifies the interaction between $\text{minTab}$ and $\text{InitTab}$, and controls their execution. It receives a $\text{GC-SROIQ(C)}$ KB $(K, M_c)$ as input, and either returns unsatisfiable or produces a GC-model for the input KB.

First of all, $\text{InitTab}$ is run. If it produces a grounded model, then it is fed as input to $\text{minTab}$. If a clash-free model is thereby produced, then the resultant model is fed as an input back into $\text{minTab}$. This continues till no more clash-free models can be produced by $\text{minTab}$. The resultant model is the GC-model of the input KB.

It may be noted that both $\text{minTab}$ and $\text{InitTab}$ are non-deterministic algorithms. So, if required, $\text{modelFinder}$ may cause the algorithms to backtrack to an earlier non-deterministic choice point. As a result the 2 algorithms will strive to produce alternate models. This may be necessary in order to find a clash-free model if the model output by $\text{minTab}$ has clashes (preference clashes or entailment clashes).

### 3.1.4 Other Inference Problems

Unlike standard DLs, other inference problems are not reducible to KB satisfiability. Below we describe the extra clash conditions whose presence in the GC-model, have implications for other inference problems besides KB’s GC-satisfiability. For every inference problem, a specific clash condition is considered. These clash conditions are called the entailment clash conditions.
• Instance checking\((C(a))\) : The GC-model graph \(F\) contains a clash if \(F\) has \(C \in \mathcal{L}^S(a)\). If this clash is present in the final GC-model produced by \(\text{minTab}\), then backtracking must ensue.

Thus, \(a\) is an instance of concept \(C\), if there is no GC-model which is free from this clash.

• Concept satisfiability\((C)\) : The GC model graph \(F\) contains a clash if \(F\) has \(C \in \mathcal{L}(x)\) for any node \(x\).

The presence of this clash means that the concept is GC-satisfiable. Else, backtracking ensues in an attempt to produce another GC-model which may have this clash. If no GC-model has this clash condition, then \(C\) is not GC-satisfiable.

• Concept subsumption\((C \sqsubseteq D)\) : Reduces to concept proving unsatisfiability of \(C \cap \neg D\)

### 3.2 Conclusion and Future Work

We have presented an inference procedure for the expressive DL \(\text{GC-SROIQ}(C)\). It has expressive constraint modelling features and can be used to perform closed world reasoning.

The correctness of the algorithm is ensured because it adheres to the grounded circumscription framework. Using the constructs of \(\text{SROIQ}\), it just provides a new intuitive way, to perform the tasks outlined in the framework. In this way, it is an implementation of the abstract algorithm whose correctness is elaborated in [4].

However, this does not diminish the usefulness of the algorithm provided here, because a detailed algorithm exists only for \(\text{ALCO}\). Though the authors predicted that such algorithm exists for any decidable language, we currently do not have the specification of such an algorithm for \(\text{SROIQ}\).

Further, the algorithm that did exist for \(\text{ALCO}\) uses special completion rules for doing so. However with the provided algorithm, we can achieve grounded circumscription using existing implementations of \(\text{SROIQ}\) reasoners.

Many tasks still remain, including finding the complexity of the presented logic, optimizing the tableau decision procedure, extending circumscription to prioritized circumscription, amongst others. Discovering measures to cope with the non-determinism associated with the inference procedure would greatly ease the practical applications of the logic.
Appendix A

Analysis of $SROIQ(C)$ tableau algorithm

Here we present sketches of proofs of correctness for the $SROIQ(C)$ tableau decision procedure.

A.1 Termination

Termination of the tableau procedure is treated in a manner analogous to the treatment of the tableau procedures in [5] and [?]. The argument for termination, in case of $SROIQ$ and $SHOIQ$, was made on basis of the completion system meeting certain properties, which together, guaranteed termination of the decision procedure. $SROIQ$ termination was ensured on account of the following properties of the completion system and the completion rules:

1. The structure among the blockable nodes is tree shaped i.e. no blockable node has more than one predecessor node.
2. All rules, except the shrinking/merging rules, add new nodes and edges or append to label sets, but do not remove any nodes or edges and do not remove contents from any label set.
3. Only generating rules add new nodes, and these rules are triggered at most once for a given concept in the label set of a given node.
4. If any node $a$ has a blockable successor, then it must have been created by the application of a generating completion rule to the node $a$ i.e. blockable nodes are not inherited (during merging etc.).
5. The length of a path between a blockable node and its descendant is bounded.
6. There is a bound on the number of nominal nodes that can be added to the completion system using the $NN$ rule.
7. The constructs of $SROIQ$, which were not included in $SHOIQ$, do not affect the termination of the tableau algorithm.

We propose that the above properties are still met by the abstract parts of the completion system, inspite of new completion rules which manipulate the abstract nodes and edges of the completion system and the new blocking mechanism. Further, the completion rules that affect
concrete nodes and edges, manipulate the completion system in a way that $S$ meets certain other properties required to ensure termination. Lastly, we argue that the remaining completion rules do not affect termination.

First, we discuss the properties met by the abstract part of $S$ and how they still hold:

1. **The depth of the abstract node tree under a nominal node is bounded.**

   This related directly to the new blocking mechanism. We must show that the depth can not be increased beyond a certain point because by that point blocking must necessarily happen. Consider the blocking mechanism, and its checks. One intuitive way to conceptualize the blocking mechanism is to consider that a node $a$ directly blocks its descendant $b$, if $b$ is a "repetition" of $a$. The notion of "repetition" is well defined: $b$ is a repetition of $a$, if $a$ and $b$ have the same values for certain properties. The blocking checks systematically lay down what these properties are. We argue that after a particular depth from the abstract node $a$, there is guaranteed to be a descendant $b$, which has the same property values as $a$, and thus blocking is guaranteed. Hence there is a limit on the number of descendant abstract nodes that an abstract node can have, in the completion system.

Next we explain what the properties are and show why their values repeat.

**BC-1:**

This check requires all of the following:

- both $a$ and $b$ must have equal label sets ($L^S(a) = L^S(b)$),
- their predecessors must have equal label sets ($L^S(a_p) = L^S(b_p)$)
- they must have the same kind of edge connecting them to their respective predecessor i.e. $L^S(\langle a_p, a \rangle) = L^S(\langle b_p, b \rangle)$.

So, the properties that we consider in this BC check are the label sets of the node, its predecessor and the label set of the edge between them. Repetition in the values of these properties leads to the nodes passing BC-1. Hence, according to BC-1, a descendant $b$ would be considered as the repetition of a ancestor if it has the same label set as the ancestor, if it’s predecessor’s label set os the same as that of the ancestor’s predecessor and it if the edge joining the node to its predecessor is labelled just as the one joining the ancestor to its predecessor.

We first prove some related propositions ...

- The concepts that form the label set of a blockable node come from a finite, well-defined set.
For a given KB, the concepts that constitute the label set of a blockable node, must be from the set \( \text{clos}(T, R) \). A blockable node can, obviously, not contain any nominal concepts, and it therefore, can also not contain the extra concepts that NN-Rule introduces. All the non-internal concepts that the label set of any node can contain come from this finite set. Also, repetitions of any concept in the label set is not permissible.

So, there is a bound on the number of distinct concepts (concrete or otherwise) that can appear in \( L^S(a) \) for any blockable node \( a \). Let this number be \( \text{max_concepts} \). So, after having at most \( 2^{\text{max_concepts}} \) blockable descendants, if another descendant exists, it would have to have a label set which would be exactly the same as one of its blockable ancestors.

Thus for a KB, the set of distinct internal constructs that may appear in node’s label set is fixed and bounded.

- **The label set of an abstract edge is a subset of a finite, well-defined set**

The label set of abstract edges is labelled with a subset of the finite set \( N_{ar} \). The power-set of \( N_{ar} \) is a finite set, and hence the number of distinct labels of an abstract edge is bounded to a fixed value (say \( w_2 \)).

Using the results above, we can establish that we have a bounded number of choices for selecting a distinct/non-repeating label set for the descendant \( b \) (say, \( w_1 \) choices), a fixed number of choices for selecting the label set for the predecessor \( b_p \) of \( b \) (\( w_2 \)), and a fixed number of choices for selecting the label set for the edge \( (b_p, b) \) (\( w_1 \) again). Thus, after \( w_3 = w_1 \ast w_2 \ast w_1 \) distinct values for the properties BC-1 tests, there is guaranteed to be a repetition in all the three properties.

**BC-2**

According to this check \( b \) is the repetition of \( a \) if they have the same edges joining them to their respective concrete successors, if \( b_p \) has this property wrt \( a_p \), and if the network formed by \( \text{relevantCNodes}(b) \) is the same as \( \text{relevantCNodes}(a) \).

We present some helping propositions first.

*For an abstract node \( x \), the number of concrete nodes that can be generated is bounded.*

New concrete nodes can be introduced only by the following : \( \geq \text{ng} \) (corresponding rule fires once for node \( x \)) and \( (\exists U_1, U_2, r) \) or \( (\exists U_1, \{i\}, r) \) (rule fires once for the predecessor of \( x \)). Now, the number of such constructs in the KB is limited. Thus there can be only a fixed number of concrete nodes of any abstract node. For later analysis let this number be \( \text{max_cNode} \).

*For an abstract node, the number of abstract nodes that can be generated as successors is bounded.*
For a given abstract node, the no of ancestors is bound to be 1 (on account of the treelike property). Further, the number of successor abstract nodes is also bounded. This fact regarding the out-degree of an abstract node, follows from the proofs of SROIQ, if we treat $\exists U_1, U_2, r$ and $\exists U_1, \{i\}, r$ as being equivalent to $\exists R, \top$ in terms of how they manipulate the abstract node structure of the completion system. These rules fire once for the node in whose label set they are in present (say $x$), and even merging of the newly constructed node back into $x$ does not cause a re-fire. For later analysis, let this maximum number of abstract nodes be $\text{max}_a\text{Node}$.

We now return to our original task. The total number of successors for a descendant $b$, and its predecessor $b_p$ is $w_4 = 2 \times \text{max}_c\text{Nodes}$. So, we have $w_5 = 2^{w_4}$ distinct options about the concrete successors that $b$ and $b_p$ can have. Further let the number of concrete roles in KB be $\text{max}_c\text{Roles}$. This is a finite number because the set of concrete roles $N_{cR}$ is finite, and the label set of any concrete edge would have to be labelled using roles from this set only. Correspondingly, we would have $w_6 = 2^{\text{max}_c\text{Roles}}$ distinct label sets for concrete edges.

Now we can have at most $w_6$ label sets for each of the concrete edges for each of the $w_4$ concrete nodes associated with $b$ and $b_p$. Thus there are $w_4^{w_6}$ options.

Finally, there can be say $w_7$ constraints between the $w_4$ nodes, each of them capable of being labelled with any one of, say, $w_8 - 1$ constraint relations. We can treat the absence of a constraint as another option. So, the each of the $w_7$ constraints can be labelled in $w_8$ ways, giving us $w_8^{w_7}$ options.

Thus, after $w_9 = w_4^{w_6} \times w_8^{w_7}$ descendants, we will have a repetition in the properties BC-2 tests. After $w_{10} = w_9 \times w_3$ there would be a descendant $b$ which passes BC-2 with $a$.

**BC-3:**

The size of $\text{relativeCNodes}(b)$ for any descendant $b$ is finite. Similarly, the nodes constituting $\text{fixedCNodes}$ must be the concrete successors of nominal nodes (but not the nominal nodes created by the NN rule). Clearly the number of such nominal nodes is fixed, and hence the number of concrete nodes in $\text{fixedNodes}$ is also fixed. Thus, the number of nodes in $\text{cNetNodes}(b)$ is also fixed, and hence the number of pairs of such nodes is also fixed, say $w_{10}$.

Now any of these pairs can be labelled by any one of say $w_{11}$ constraint relations. Thus we have $w_{12} = w_{10}^{w_{11}}$ options. Thus, after at most $w_{10} \times w_{12}$ options, we would have to have a descendant $b$ that passes BC-3. This suggests that the depth of the abstract tree can not grow beyond a certain limit.

It may also be pointed out that the approximations used here are grossly large approximations of the actual quantity and we have ignored the correlations that do not allow different parameters to take values independently. Actually, the conditions of BC-1 highly increase the chances for BC-2 and BC-3 to pass, as well. Thus the upper bound is a very loose one, but does serve to establish that there does exist a bound.
2. There is a bound on the number of abstract nodes $S$ can have.

We have seen that the depth of an abstract node tree is bounded. This abstract node tree must be rooted at a nominal. Further, there are a limited number of nominal nodes in the system which are created during algorithm initialization. Other than these, nominal nodes can also be created by the $NN$ rule. However, results from $S\text{HOIQ}$ establish that there is a bound on the number of such nodes, and this fact remains unchanged for both $S\text{ROIQ}$ and $S\text{ROIQ}(C)$. Thus, we can say that the number of all abstract nodes in the completion system is bounded.

Next we discuss some properties pertaining to the novel aspects of $S\text{ROIQ}(C)$ completion system:

1. There is a bound on the number of concrete nodes $S$ can have.

We established the existence of a bound on the number of abstract nodes in the completion system. Further, there is a bound on the number of concrete node successors that an abstract node can have. Using both the above, we can say that there is a bound on the total number of concrete nodes that $S$ can have.

2. The size of the constraint template set $Q$ is bounded

Each of the concrete concepts $\exists \, U_1, U_2, r, \exists \, U_1, \{i\}, r, \forall \, U_1, U_2, r, \forall \, U_1, i, r$, trigger (at most once) their respective completion rule which adds an entry into $Q$. The number of concepts of the above form, in the KB, would be a fixed number, say $x_1$. We have established that the number of abstract nodes in the completion system is bounded, to say $x_2$. If all $x_1$ concepts are there in each of the $x_2$ abstract nodes, then also, there would be $x_1 * x_2$ i.e. a bounded number of entries of the form $(q'_1, r, q'_2)$.

Now, in light of the above established properties, we discuss how the number of times each completion rule can be applied is bounded for a given KB.

- $R\exists \, U, R\exists \, i, R\neg \exists \, U, R\neg \exists \, i, R\forall \, U, R\forall \, i$: Each of the first two rules is triggered once, for each instance of their respective trigger concepts in the label set of an abstract node. In the worst case, the last 4 rules could fire once for every pair of $U_1, U_2$ successors or for every pair of $U_1$ successor and constraint individual $i$. But since the number of such $U_i$ successors is also fixed, the rules fire a finite number of times.

Since the number of abstract nodes in $S$ is bounded, and the number trigger concepts in the label set of an abstract node is bounded, hence, the number of times these rules get triggered is bounded too.

- $R\exists \, i$: This rule belongs to class of ”generating rules”. It adds new abstract edges to the completion system. Clearly there must be a bound on the number of times this rule could be applied. If it was not so, then the bound on the total number of abstract nodes in the completion system, that we obtained earlier, would not have been possible.

- $R\forall \, i$: Analogous to the case for $R\exists \, U$ with the difference that it adds appends to the label sets of its neighbours.
• **Rconnect:** This rule introduces a constraint relation between two existing concrete nodes, based on the patterns in $Q$. But we already know that there can be at most a fixed, finite number of concrete nodes in the completion system. Further, there are a finite number of constraint relations. Thus, it means that this rule can, at most, introduce all possible constraints between all pairs of concrete nodes. Thereafter, it can't fire any more.

• **R\geq_c, R\leq_c:** Analogous to treatment of $R\exists_c$.g.

• **Rcomplete_c:** This rule just introduces new constraints between nodes of a fixed size set, once. Does not affect termination since it applies only when two nodes pass BC-3, which in turn can happen only a finite number of times, given that there is a bound to the depth to which abstract node trees can grow.

### A.2 Soundness

To establish the soundness of the tableau algorithm, we need to prove the following:

*If the tableau algorithm returns satisfiable, then the knowledge base $\langle T, R, A \rangle$ is consistent.*

Alternatively,

*If the tableau Algorithm produces a complete and clash-free completion system $S$, then there exists a valid augmented tableau for the KB.*

We begin by describing a procedure to construct an augmented tableau, using the completion system produced by the tableau algorithm. We then provide a sketch of the proof regarding the validity of the above constructed augmented tableau.

As proposed, an augmented tableau for $S\text{ROIQ}(C)$ is the tuple $T_A = \langle T, N \rangle$ where $T = \langle V^T_a, V^T_c, L^T, E^T_a, E^T_c \rangle$ is a tableau and $N$ is a constraint network with $V_N = V^T_c$. It is easy to argue that the existence of a model of the KB $I = (\Delta_I, \cdot_I, M_I)$ is an implication of the existence of an augmented tableau for the KB. The construction of the augmented tableau may be divided into steps: the construction of $T$, the addition of constraint relations between the concrete nodes of the tableau to construct $N$.

#### A.2.0.1 Procedure for Constructing $T_A$

**Constructing $T$**

$T$ is obtained from the completion system by standard unravelling of the completion graph $G$ [3]. We explore how the completion system can be used to create a possibly infinite tableau for the knowledge base. The method we employ is an extension of the one used in [?] and [5].

To uniquely identify every non-nominal node in the unravelled tableau, we use the notion of an "id-path". An *id-path* is defined as a possibly empty sequence of a pair of nodes of the form: $p = ((a_1, b_1), (a_2, b_2) \ldots (a_i, b_i) \ldots (a_n, b_n))$, where,

- $a_i, b_i \in V^S_a \cup V^S_c$
- $a_i, b_i$ are never descendants of directly blocked nodes.
• only $a_1, b_1$ may possibly be from the set of nominal nodes in completion graph $Nom(S)$.
  Thus, nominal node pairs, if present, may come only as the first pair of a path.

• For all $i \geq 1$, $b_{i+1}$ is a successor of $a_i$ for an a role $R \in N_R \cup N_{c,F}$ i.e. the edge from $a_i$ to $b_{i+1}$ is explicitly present in $E^S_a \cup E^S_c$.

• Further,
  \[ a_i = \begin{cases} 
  \text{Blocker}(b_i) , & \text{if } b_i \text{ directly blocked} \\
  b_i , & \text{otherwise} 
  \end{cases} \]

For such an id-path $p$, we define $Head(p) = a_1$, $Head'(p) = b_1$, $Tail(p) = a_n$ and $Tail'(p) = b_n$. In following paras we will use terms like $p, x, y$ to represent paths.

We define the following sets based on the above definitions:

\[ Paths(S) = \{ p \mid p \text{ is a id-path} \} \]
\[ Nodes(S) = \{ p \mid p \in Paths(S) \} \]
\[ Head(p) = Head'(p) \in Nom(S) \]
\[ V^T_a = \{ x \mid x \in Nodes(S) \} \]
\[ Tail(x), Tail'(x) \in V^T_a \]
\[ V^T_c = \{ y \mid y \in Nodes(S) \} \]
\[ Tail(y), Tail'(y) \in V^T_c \]
\[ Nodes_{Nom}(S) = \{ e \mid e \text{ of the form } ((o, o)) \text{ for } o \in Nom(S) \} \]
\[ Nodes_{NN}(S) = \{ x \mid x \in Nodes(S); x \neq ((o, o)) \} \]
\[ \text{for any } o \in Nom(S) \] \]

$Nodes(S)$ is the set of all nodes in the augmented tableau, including both concrete and abstract nodes. $Nodes_{Nom}(S)$ is the set of nominal, and $Nodes_{NN}(S)$ the set of non-nominal (hence $NN$) abstract nodes in the tableau.

We introduce some more definitions:

A tableau node $n \in Nodes(S)$ is said to be simplex in case for all pairs $(a_i, b_i)$ in $n$, we have $a_i = b_i$.

A node $n \in Nodes(S)$ is said to be associated with the tableau nominal node $e$, if $Head(n) = Head'(n) = e$.

$p = (q|(a, a'))$ indicates that the id-path $p$ is formed by appending the tuple $(a, a')$ to the end of $q$.

A node $x$ is the edge-predecessor of node $x'$ if $x' = (x \mid (d, d))$ for some $d \in V^S_a \cup V^S_c$. Predictably, $x'$ is the edge-successor of $x$. A safe simplex node is a simplex node $n$ such that $n$ is the edge successor of nominal tableau node $e$.

Consider a concrete safe simplex nodes such that $y \in V^T_c$ and it is of the form $((r, r), (i, i))$ where $r \in V^S_a$ is a nominal node, $i \in V^S_c \cap N_{c,t}$ is a constraint individual. For such nodes with id-paths $((r, r), (i, i))$, we use $i$ as an alias for them. This is essential because in many concepts in the label sets of tableau nodes such as $\exists_i U, [i].r$, a reference is made to the concrete node $i$. 

36
Assume $R \in N_{ar}$ and $g \in N_{cR}$. We define the remaining tableau structures:

$$\mathcal{L}^T(p) = \mathcal{L}^S(Tail(p))$$

$$\mathcal{E}^T_a(R) = \{(x, x') \in \text{Nodes}_{NN}(S) \times \text{Nodes}_{NN}(S) |$$

$$x' = (x|(a, a')) \text{, and } Tail'(x') \text{ is a } R \text{ neighbour of } Tail(x), \text{ OR}$$

$$x = (x'|(a, a')) \text{, and } Tail'(x) \text{ is a } \text{Inv}(R) \text{ neighbour of } Tail(x')\}$$

$$\cup$$

$$\{(x, e) \in \text{Nodes}_{NN}(S) \times \text{Nodes}_{Nom}(S) |$$

$$Tail(e) \text{ is a } R \text{ neighbour of } Tail(x)\}$$

$$\cup$$

$$\{(e, x) \in \text{Nodes}_{Nom}(S) \times \text{Nodes}_{NN}(S) |$$

$$Tail(x) \text{ is a } R \text{ neighbour of } Tail(e)\}$$

$$\cup$$

$$\{(e, e') \in \text{Nodes}_{Nom}(S) \times \text{Nodes}_{NN}(S) |$$

$$Tail(e') \text{ is a } R \text{ neighbour of } Tail(e)\}$$

$$\mathcal{E}^T_c(g) = \{(x, y) \in V^T_c \times V^T_c |$$

$$y = (x|(c, c)) \text{, and } c = \text{Tail}(y) \text{ is a } g \text{ successor of } Tail(x)\}$$

One may notice that the tableau no longer maintains any treelike property between its non nominal nodes. For eg. assume $a'$ is the R successor of $a$, and $R \subseteq R'$. Let $x$ be a tableau node with $Tail(x) = a$. Now there would be a node $x' = (x | (a', a'))$. ($a''$ would be the same as $a'$ depending on whether $a'$ node is blocked or not in $S$). Now, $(x, x') \in \mathcal{E}^T_a(R), (x, x') \in \mathcal{E}^T_a(R'), (x', x) \in \mathcal{E}^T_a(\text{Inv}(R))$ and $(x', x) \in \mathcal{E}^T_a(\text{Inv}(R'))$. Except the complex roles, all other information has been made explicit. Whatever remains of the treelike structure information is captured in the form of their edge-successor and edge-predecessor relations.

Also, notice that in the above two definitions, we have used $Tail(x)$ and not $Tail'(x)$, which means that the properties of the tableau node $x \in \text{Nodes}_{NN}(S)$ with $Tail'(x) = b = (\text{some blocked node in } S)$, are decided by the node which blocked $b$.

**Adding the constraints**

Next we explain a way to link the concrete nodes of the tableau by constraint relations. This would be a little more involved than the above, and we need to introduce some concepts before proceeding further.

An abstract, non-nominal tableau node $h \in V^T_a$ is termed a hook if $Tail(h) \neq Tail'(h)$. Further, all nominal nodes of the tableau are defined to be hooks, by default. The set of all hooks in the
tableau is termed \textit{Hooks}(S). Hook \(h'\) is the successor hook of \(h\) (\(h\) is the predecessor hook of \(h'\)), if, \(h' = hl\), where all \(a_i = b_i\) in \(l\), except \(a_n = \text{Tail}(h')\) and \(b_n = \text{Tail}'(h')\) for which of course \(a_n \neq b_n\) (by definition of a hook). We define:

\[
\text{Hooks}(S,e) = \{ h \mid h \in \text{Hooks}(S) \text{ and } h \text{ is associated with nominal tableau node } e \}
\]

It is possible to define an ordering on the members of the set \(\text{Hooks}(S,e)\) depending upon the increasing distance of the hook from the nominal tableau node \(e\).

We attempt to project the entire constraint network \(N\) as being the union of smaller, individually satisfiable constraint networks. Firstly, we decompose \(N\) into the following two constraint networks:

Formally, the above translates into:

\[
N = N_{\text{simplex}} \cup N_{\text{hook}}
\]

\[
N_{\text{simplex}} = \{ (y \text{ r } y') \mid y, y' \in V^T_e \text{ are simplex nodes;} (\text{Tail}(y) \text{ r } \text{Tail}'(y')) \in N \}
\]

\[
N_{\text{hook}} = \bigcup_{e \in \text{Nodes}_{\text{Nom}}(S)} N^e_{\text{hook}}
\]

\(N_{\text{hook}}\) consist of constraints in which at least one of the concrete nodes is non-simplex. In these constraints, either both the nodes in the constraint can be non-simplex; else, if only one of them is non-simplex, then the other must be a simplex node. \(N^e_{\text{hook}}\) places the extra restriction that the non-simplex node must be associated with \(e\). The next few paras are devoted to explaining just which constraints form a part of \(N^e_{\text{hook}}\).

Hooks are used to divide the non-simplex nodes of the augmented tableau, into disjoint sets. To formalize the above, we define:

\[
\text{PathFragments}(S) = \{ p \mid p \in \text{Paths}(S); \text{Head}(p), \text{Head}'(p) \notin \text{Nom}(S); \text{all pairs } (a_i, b_i) \text{ in } p \text{ satisfy } a_i = b_i; \}
\]

\[
\text{Nodes}_h = \{ p \mid p \in \text{Nodes}(S); p = hl; h \text{ is a hook}; l \in \text{PathFragments}(S) \}
\]

It may be noted that the set \(\text{Nodes}_h(S)\) is finite, and includes the hook \(h\) itself, but not its successor hook. For purposes of discussions, each of the nodes constituting \(\text{Nodes}_h(S)\), is said to be "\textit{in the domain of } h". One may notice here that all simplex nodes are in the domain of the nominal tableau node that they are respectively associated with.

The set of non-simplex nodes \textit{associated} with a particular nominal tableau node \(e\), can be broken into disjoint sets, each of which would be associated with some non-nominal hook node. This property is used to decompose the possibly infinite \(N^e_{\text{hook}}\), into finite constraint networks (say,
\(N_h\), each of which corresponds to a non-nominal hook \(h\), such that :

\[
N^{\text{hook}}_c = \bigcup_{h \in \text{Hooks}(S_e, \{c\})} N_h
\]

We now describe the construction of \(N_h\). Assume \(\text{Tail}(h) = a\) and \(\text{Tail}'(h) = b\). We introduce the following networks :

\[
N_h = N^{\text{int}}_h \cup N^{\text{pre}}_h \cup N^{\text{ss}}_h
\]

\(N^{\text{int}}_h\) consists of constraints between concrete nodes, both of which fall in the domain of the hook \(h\). (int comes from “internal”) It is defined as :

\[
N^{\text{int}}_h = \{(y \text{ r } y') | y, y' \in N_{\text{Nodes}}_h \cap V_c; (\text{Tail}(y) \text{ r } \text{Tail}(y')) \in N\}
\]

In above case, \(\text{Tail}(y)\) as well as \(\text{Tail}(y')\) will both point to nodes in \(\text{internalCNodes}(a, b)\)

\(N^{\text{ss}}_h\) consists of constraints such that one of the concrete nodes is in the domain of the hook \(h\), while the other is some safe simplex node. (ss comes from safe simplex)

\[
N^{\text{ss}}_h = \{(y \text{ r } y') | y \in N_{\text{Nodes}}_h; y' \text{ is a safe simplex node}
\]

\[
(Tail(y) \text{ r } Tail(y')) \in N;
\]

\[
\text{Tail}(y') \in \text{fixedCNodes}(a, b) \}
\]

\(N^{\text{pre}}_h\) is a bit more complicated, and to define it we will use the mapping \(\theta\) we introduced in BC-3. Since this is an injective mapping, its inverse is well defined. We define \(\mu = \theta^{-1}\). Consider a non-nominal hook \(h\) and its predecessor hook \(p\). \(N^{\text{pre}}_h\) consists of constraints between concrete tableau nodes, one of which is in the domain of the hook \(h\), while the other is in the domain of the hook predecessor \(p\) (hence the term \text{pre}).

We define :

\[
N^{\text{pre}}_h = \{(y \text{ r } y') | y \in N_{\text{Nodes}}_h; y' \in N_{\text{Nodes}}_p;
\]

\[
\text{for}
\]

\[
(Tail(c) \text{ r } \mu(Tail(c'))) \in N\}
\]

\(\cup\) the symmetric case

Note that in \(S\), the constraint is not between \(c = \text{Tail}(y)\) and \(c' = \text{Tail}(y')\) but between \(c\) and \(\mu(c')\).

\subsection*{A.2.0.2 Validity of \(T_A\)}

Here, we try to prove the following :

\(\text{The structure created above is a valid augmented tableau for } \langle R, T, A \rangle\)

It suffices to prove that the structure satisfies all the propositions that must hold in an aug-
mented tableau. We discuss them one by one ...

• \((P_c NSat)\): \(N\) should be satisfiable.

It is in proving this property that the \(\omega\) admissibility of the completion system comes into use. We use the compactness property to pose the possibly infinite \(N\) as the union of finite, satisfiable constraint networks. Since some of these networks may share variables, hence we use the patchwork property to argue the following: since the individual networks are satisfiable and their common variables form complete networks, therefore, \(N\) would also be satisfiable.

We of course have to prove that the individual networks are satisfiable and that their intersection networks (network formed by common variables) are indeed complete. As stated earlier:

\[
N = N_{simplex} \cup \bigcup_{e \in Nodes_{Nom}} \bigcup_{h \in Hooks(S,e)-\{e\}} N_h
\]

The complete and clash free property of \(S\) ensures that \(N_{simplex}\) is satisfiable. This is so because it is defined in a way that there is a 1-to-1 mapping from the variables of \(N_{simplex}\) to those of \(N\) (given by \(Tail(y)\)), and clearly \(N\) has to be satisfiable to ensure that \(S\) is clash-free.

For similar reasons, each of the networks \(N_{int}^h\), \(N_{pre}^h\) and \(N_{ss}^h\) are satisfiable, as well.

Now we focus attention on the variables common between these networks. Consider the constraint network \(N_{hook}^e\). It is constructed incrementally, hook by hook. Assume that, at the time, \(N = N_{simplex}\). In following lines, we attempt to observe how more constraints are added to \(N\).

We have discussed that the hooks can be ordered in terms of their depth from the nominal \(e_1\). Consider the first non-nominal hook: \(h_1\) i.e. \(h_1\) is the successor hook w.r.t \(e\). The constraint network associated with it is \(N_{h_1}\). There are some variables common between \(N_{h_1}\) and \(N_{simplex}\). Now, by def, there are no variables common between \(N_{simplex}\) and \(N_{int}^h\). But there may be some variables common between \(N_{simplex}\) and \(N_{pre}^h \cup N_{fixed}^h\). These common variables are such that their \(Tail\) points to the set \(cNetNodes(b)\), where \(b\) is blocked by \(a\) (also, \(Tail(h_1) = a, Tail'(h_1) = b\)).

More specifically, for \(N_{pre}^h\), the common vars are those which have their \(Tail()\) pointing to nodes of \(associatedCNodes(b)\). The must be concrete edge successor nodes of the edge predecessor of the hook \(h\). For \(N_{fixed}^h\), the common vars are the safe simplex nodes which have their \(Tail()\) pointing to nodes of \(fixedCNodes(a, b)\). These must be the edge successors of nominal tableau nodes.

Clearly, the network these nodes form is ensured to be complete because the completion rules ensure that the network formed by \(cNetNodes(b)\) is a complete one. Using patchwork property we can say that \(N = N \cup N_{h_1}\) is satisfiable. This directly follows since \(N\) and \(N_{h_1}\) are both satisfiable, and their intersections is a complete, satisfiable network.
Next, when we add the constraints of $N_h$, to $N$, they will again have common variables which can be again mapped to the variables $c\text{NetNodes}(b)$. $N_{h_i}^{pre}$ may have variables common with $N_h$. These would be nodes in the domain of $h$ such that their tails point to nodes of $\text{associatedCNodes}(b)$. $N_{h_i}^{in}$ may have variables common with the variables of $N_{\text{simplex}}$, such that their tails would point to nodes of $\text{fixedCNodes}(a, b)$.

In this way, we can see that $N = N \cup N_{h_2}$ is satisfiable. This follows because $N$ was satisfiable. $N_{h_2}$ is a satisfiable constraint network, because it is isomorphic to a satisfiable network, (which is a subset of the satisfiable network $N$). And the intersection of the above networks is complete and satisfiable because the network formed by their common variables is isomorphic to $c\text{Net}(a)$ and $c\text{Net}(b)$.

We point out that the above intersection network is indeed complete. The constraints between the simplex nodes had been introduced as a part of $N_{\text{simplex}}$, The links between the nodes in the domain of $h_1$ must have been introduced as a part of $N_{h_1}^{int}$. The links between the nodes in domain of $h_1$ and the simplex nodes, would have been introduced as a part of $N_{h_1}^{ex}$. Thus, $N$ is satisfiable, even after addition of constraints from $N_{h_2}$.

Working this way we can include the constraint networks for other hooks and nominal nodes, and argue that the resultant $N$ would be satisfiable.

- $(P_\alpha)$: For all the propositions of the form $P_\alpha$, please refer to the proof of soundness for $\text{SROIQ}$. The augmented tableau for $\text{SROIQ}(C)$ shares these propositions with the tableau for $\text{SROIQ}$.

- $(P_\epsilon \exists, U)$: If $(\exists, U_1, U_2, r) \in \mathcal{L}^T(x)$, then there must exist $y_1, y_2 \in V^T_{\epsilon}$ such that $y_1$ is a $U_1$ successor and $y_2$ is a $U_2$ successor of $x$ and $(y_1, r, y_2) \in N$.

Without loss of generality, assume that $U_1 = R_1g$ and $U_2 = g'$. Thus, its given that $(\exists, Rg, g', r) \in \mathcal{L}^S(x)$.

Clearly, $(\exists, Rg, g', r) \in \mathcal{L}^S(a)$ for $a = \text{Tail}(x)$. Assume $x \in V^T_{\alpha}$, and $\text{Tail}(x) = a$ and $\text{Tail}(x) = a'$, where $a$ and $a'$ are not necessarily the same (For eg: if $x$ is a hook, then they would be different and $a'$ would be the blocker of $a$ in the completion system).

For the purposes of the following discussion, a node exists in $S$ if it is in the set of non blocked nodes of the completion system $S$. If they exist in $S$, let $c_1$ be the $U_1$ neighbour of $a$ and $c_2$ the $U_2$ neighbour of $a$, such that $(c_1, r, c_2)$. If $c_1$ and $c_2$ exist, we can also propose the existence of $a_1$, the $R$ neighbour of $a$, such that $c_1$ is the $g$ successor of $a_1$.

$\text{Tail}(x) (= a)$ can never point to a blocked node ($\text{Tail}'(x)$ may, but not $\text{Tail}(x)$). From this, we may infer that $c_2$ will always point to a non blocked node.

However, $a_1$, the successor of $a$ can have two cases: its either the directly blocked node, or its not blocked at all. If $a_1$ is blocked then so will be $c_1$. Thus, the case when $c_1$ doesn’t exist in $S$ (because of being blocked) is equivalent to the case when $a_1$ is directly blocked.

We now take cases depending upon the nature of $x$, and $c_1$.

**Case 1**: $x$ is a simplex node.

- **Case 1a**: both $c_1$ and $c_2$ exist ($\in V^S_{\epsilon}$ and not blocked)

  We first show that $x$ has a $Rg$ successor $x_1$. Now there are two cases for $a_1$:
* $a_1$ could be the successor of $a$, such that $R'_1 \in \mathcal{L}^\delta(\langle a, a_1 \rangle)$ for some $R'_1 \subseteq R_1$. By construction of the tableau, there will exist an abstract tableau node $x_1$ such that $x_1 = (x \mid (a_1, a_1))$. Also, by construction, $\langle x, x_1 \rangle \in E'_0(R'_1) \cap E'_0(R_1)$.

* Alternatively, $a$ could be the successor of $a_1$, with $R''_1 \in \mathcal{L}^\delta(\langle a_1, a \rangle)$ for some $R''_1 \subseteq \text{Inv}(R_1)$. By construction, there will exist an abstract tableau node $x_1$ such that $x = (x_1 \mid (a, a))$. Also, by construction, $\langle x_1, x \rangle \in E'_0(R''_1) \cap E'_0(\text{Inv}(R_1))$, and $\langle x, x_1 \rangle \in E'_0(R_1)$.

In comparison, $c_1$ is constrained to be the successor of $a_1$. That is, $g' \in \mathcal{L}^\delta(\langle a_1, c_1 \rangle)$ for some $g' \subseteq g$. By construction of the tableau, there will exist an abstract tableau node $y_1$ such that $y_1 = (x \mid (c_1, c_1))$. Also, by construction, $\langle x_1, y_1 \rangle \in E'_0(g') \cap E'_0(g)$.

Similarly, we can prove the existence of $y_2$ such that $\text{Tail}(y_2) = c_2$.

Since $x$ is simplex and $c_1$ and $c_2$ are both present in $S$, therefore we can conclude that $y_1$ and $y_2$ are simplex nodes as well.

The constraint between $y_{U_1}$ and $y_{U_2}$ is included in the constraint network $N_{\text{simplex}}$, $(y_{U_1} \ r\ y_{U_2}) \in N$ because : $(\text{Tail}(y_{U_1}) \ r\ \text{Tail}(y_{U_2})) \in N$ i.e $(c_{U_1} \ r\ c_{U_2}) \in N$.

### Case 1b: $c_1$ does not exist

As mentioned earlier, this is possible only if $a_1$ is directly blocked, by (say) $b_1$. Since $x$ is simplex, $\text{Tail}(x) = \text{Tail}'(x) = a_1$. This means that the situation in $S$ is such that $b_1$ blocks $a_1$, and $a$ is the predecessor of $a_1$ and $b_1^p$ is the predecessor of $b_1$. Then, there must exist $x_1 = (x \mid (b_1, a_1))$.

But the question is, is $c_1$ in $V^\delta_c$? it may be blocked but is it there? Now, we know that $S$ is complete and clash-free (CCF). $a$ does not have a non-blocked $Rg$ neighbour $c$ such that $(c \ r\ c')$ where $c'$ is its $g$ successor. But $a$ is not blocked and the $S$ is CCF, thus we can conclude that $a_1$ was created by some rule like $R\iota_1$, which would have created the $R$ successor of $a (a_1)$ and the $g$ successor of $a_1 (c_1)$, all in one fell swoop.

Thus we can now conclude that if $\exists Rg, g'.r$ is in some non-blocked node of $S$, and it does not have any non blocked $Rg$ and $g'$ successors $c, c'$ such that $(c \ r\ c') \in N$, then : $c$ is indeed a member of the set $V^\delta_c$ but is blocked. Further, the constraint $(c \ r\ c')$ must be in $N$ because $\text{Rconnect}$ rule works irrespective of whether the nodes it connects in a constraint are blocked or not.

Now we return to our original task. We have conjectured the existence of the hook $x_1 = (x \mid (b_1, a_1))$. Since $(c_1 \ r\ c_2) \in N$, by BC-2, we must have a $g'$ successor of $b_1^p$ (say $c_1'$) and a $R$ neighbour of $b_1^p$ (i.e. $b_1$) and a $g$ successor of $b_1$ (say $c_1'$) such that $(c_1' \ r\ c_2') \in N$.

Thus, we will have a $y_1 \in V^T_c$ such that $y_1 = (x_1 \mid (c_1', c_1'))$. $y_1$ is clearly in the domain of $x_1$. We already have a $y_2$ such that $y_2 = (x \mid (c_2, c_2))$.

The constraint between $y_1$ and $y_2$ would be made as a part of $N^\text{pre}_{V_{y_1}}$. $(y_{U_1} \ r\ y_{U_2})$ must be
in $N$, because there exists in $N : (\text{Tail}(y_1) \triangleright \text{Tail}(y_2)) = (c'_1 \triangleright c_2) = (c'_1 \triangleright c'_2)$.

**Case 2:** $x$ is not a simplex node

Let $x$ be associated with the tableau nominal $e$, and be in the domain of the non-nominal hook $h$. The case demands that $\text{Tail}(x) \in \text{internalCNodes}(b_1, b_2)$, where $b_1$ directly blocks $b_2$ in $S$. We can conclude that $c_2$ will always be in $\text{internalCNodes}(b_1, b_2)$.

For the $Rg$ successor ($c_1$), there are some options regarding where it can exist in $S$. It may be in $\text{internalCNodes}(b_1, b_2)$, $\text{fixedCNodes}(b_1, b_2)$, $\text{relativeCNodes}(b_1, b_2)$ or it may be blocked. We have to take cases for all of these cases.

- **Case 2a:** $c_1 \in \text{internalCNodes}(b_1, b_2)$
  
  Arguing as in Case 1a, we can propose the existence of a $U_1$ successor of $x$ ($y_1$) and a $U_2$ successor of $x$ ($y_2$), such that $\text{Tail}(y_1) = c_1$ and $\text{Tail}(y_2) = c_2$. Furthermore, $y_1$ and $y_2$ can both be argued to be in the domain of the hook $h$.
  
  The constraint between them would be made as a part of $N^\text{int}_g$ i.e. ($y_1 \triangleright y_2$) $\in N$ because $(\text{Tail}(y_1) \triangleright \text{Tail}(y_2)) \in N \implies (c_1 \triangleright c_2) \in N$.

- **Case 2b:** $c_1 \in \text{fixedCNodes}(b_1, b_2)$
  
  We can argue the existence of the $g'$ successor of $x$ ($y_2$). $y_2$ would be in the domain of $h$.
  
  Lest consider the existence of $y_1$. $c_1 \in \text{fixedCNodes}(a, b)$ means that there exists a nominal node which is the $R$ successor of $a$, (say $r$), and it has $g$ successor $c_1$ such that $(c_1 \triangleright c_2) \in N$. Using the above fact we can establish that $\langle x, e \rangle \in E^g_0(R)$, where $e = (r, r)$ is a nominal tableau node. We can further argue about $y_1$ such that $y_1 = (e \mid (c_1, c_1))$. Clearly, $y_1$ is a safe simplex node.
  
  The constraint between $y_1$ and $y_2$ them would be made as a part of $N_{ss}$. ($y_1 \triangleright y_2$) $\in N$ because $(\text{Tail}(y_1) \triangleright \text{Tail}(y_2)) \in N \implies (c_1 \triangleright c_2) \in N$.

- **Case 2c:** $c_1 \in \text{relativeCNodes}(b_1, b_2)$
  
  Consider the completion system. The case suggests that $c_1$, the $Rg$ neighbour of $a$ is in $\text{relativeCNodes}(b_1, b_2)$. By definition of $\text{relativeCNodes}(b_1, b_2)$, this means that $c_1$ is the $g$ successor of $b_1^p$ (the predecessor of $b_1$). This means $a$ is a blocker node (i.e. $b_1 = a$).
  
  On account of BC-2, one can also argue the existence of some $g$ successor of $b_2^p$ (say $c'_1$, such that $c'_1 = \phi(c_1)$) and some $g'$ successor of $b_2$ (say $c'_2$, such that $c'_2 = \phi(c_2)$) such that $(c'_1 \triangleright c'_2) \in N$.
  
  Now, consider the tableau. We can argue the existence of the $g'$ successor of $x$ ($y_2$), which would be in the domain of $h$. Its also known that $x$ is not simplex, and $\text{Tail}(x)$
points to $a$, hence it must be a hook, with $\text{Tail}(x) = b_1 = a$ and $\text{Tail}'(x) = b_2$ (for this case $a = b_1$). Now there exists a node $x_1$ (the edge predecessor of $x$) such that $x = (x_1 | (b_1, b_2))$, and $\text{Tail}(x_1) = \text{Tail}'(x_1) = b_2^p$. Further, we define $y_1$ as $y_1 = (x_1 | (c'_1, c'_1))$. We can see that $y_1$ and $x_1$ belong in the domain of the predecessor hook (say $h_p$).

Now, the connection between $y_2$, $g'$ successor of $x$, and $y_1$, the $Rg$ successor of $x$ is made as a part of $N_{h_p}^{\text{pre}}$, where $y_1$ is in the domain of $h_p$ while $y_2$ is in the domain of $h$. $(y_1 r y_2)$ must in $N$, since there exists in $N$, the following constraint : $(\theta^-(\text{Tail}(y_1)) r \text{Tail}(y_2)) = (\theta^-(c'_1) r c_2) = (c_1)$

- **Case 2d: $c_1$ is blocked**

Consider $S$. We know that a tableau node cannot have its $\text{Tail}()$ point to any node that is blocked. Given that $\text{Tail}(x)$ points to $a$, $a$ cannot be blocked. However, the premise of the case suggests that $c_1$, the $Rg$ successor of $a$, is blocked. This can happen only in case $a_1$ is blocked. However, $c_2$ is not blocked and we do we have $(c_1 r c_2) \in N$.

With regards to blocking the situation is that $b_1$ blocks $a_1$, their respective predecessors are $b_1^p$ and $a$.

Consider now the Tableau. Using BC-2 we can propose the existence of the $g$ successor of $b_1$ (say $c'_1 = \phi^-(c_1)$), and the $g'$ successor of $b_1^p$ (say $c'_2 = c_2$) such that $(c'_1 r c'_2) \in N$. We can propose the existence of $y_2$ such that $y_2 = (x | (c_2, c_2))$. Further there exists $x_1$ such that $x_1 = (x | (b_1, a_1))$, and $y_1 = (x_1 | (c'_1, c'_1))$. As we can see, $x_1$ is a hook, and $y_2$ is in the domain of $x_1$, while $y_1$ is in the domain of $h$, same as $x$. The link between $y_1$ and $y_2$ is introduced as a part of $N_{x_1}^{\text{pre}}$. $(y_1 r y_2) \in N$ because there exists in $N$ the following relation : $(\text{Tail}(y_1) r \theta^- (\text{Tail}(y_2)))$ i.e. $(c'_1 r \theta^- (c_2))$ i.e. $(c'_1 r c'_2)$.

- $(P_c \forall_c U)$: If $(\forall_c U_1, U_2, r) \in \mathcal{L}^s(x)$, then for all $y_1, y_2$ such that $y_1$ is a $U_1$ successor, $y_2$ is a $U_2$ successor of $s$ and $(y_1 r y_2) \in N$.

Without loss of generality, assume that $U_1 = R_1g$ and $U_2 = g'$. Thus, its given that $(\forall_c Rg, g', r) \in \mathcal{L}^s(x)$.

Clearly, $(\forall_c Rg, g', r) \in \mathcal{L}^s$ (for $a = \text{Tail}(x)$). Assume $\text{Tail}(x) = a$ and $\text{Tail}'(x) = a'$, where $a$ and $a'$ are not necessarily the same (For e.g. if $x$ is a hook, then they would be different and $a'$ would be the blocker of $a$ in the completion system).

Let $x$ have any number of $Rg$ and $g'$ successors in the tableau. We take pairs of them one by one. Consider any such $Rg$ successor $y_1$ and $g'$ successor $y_2$. We have to prove that $(y_1 r y_2) \in N$. Let $\text{Tail}(y_1)$ be $c_1$ and $\text{Tail}(y_2)$ be $c_2$. Further since $y_1$ is the $Rg$ successor, then there must be a $R$ successor of $x$ (say $x_1$) such that $Y_1$ is the $g$ successor of $x_1$. Let the $\text{Tail}(x_1)$ be $a_1$.

We take cases depending upon the nature of $x$ and the location of the successors. We point out that in any case $y_1$ would be in the domain of the same hook as $x$. We can always propose the existence of $c_2$ such that $c_2$ is the successor of $a$ and $y_2 = (x | (c_2, c_2))$, where $a = \text{Tail}(x)$ (not necessarily $\text{Tail}'(x)$). So, we only need to worry about $y_1$. 

44
– **Case 1**: \( x \) is a simplex node.

Let \( x \) be in the domain of the nominal tableau node \( e \). Let \( \text{Tail}(x) = \text{Tail}'(x) = a \).

* **Case 1a**: \( y_1 \) is a simplex node i.e. its in the domain of \( e \).

Its presumed that \( x \) has a \( Rg \) successor. Let the \( R \) successor be \( x_1 \). Now, if \( x_1 \) is the edge-successor of \( x \), then \( \text{Tail}(x_1) \) is the \( R \) successor of \( x \), and \( R' \in \mathcal{L}^S((\text{Tail}(x), \text{Tail}(x_1))) \) for some \( R' \subseteq R \). Else, \( R' \in \mathcal{L}^S((\text{Tail}(x_1), \text{Tail}(x))) \) for some \( R' \subseteq \text{Inv}(R) \). Further since \( y_1 \) must be the \( g' \) successor (edge successor) of \( x_1 \), we have \( g' \in \mathcal{L}^S((\text{Tail}(x_1), \text{Tail}(y_1))) \) for some \( g' \in g \).

Now, \( Y_1 \) is bound to be such that \( y_1 = (x_1 \mid (c_1, c_1)) \). This case implies the presence of a \( Rg \) neighbour of \( a \) (say \( c_1 \)) such that its not blocked, and \( \text{Tail}(y_1) = \text{Tail}(y_1) = c_1 \).

Now, we have conjectured the presence of \( a \), its \( Rg \) successor \( c_1 \) and \( g' \) successor \( c_2 \). Clearly, CCF condition of \( S \) ensures \( (c_1 \ r \ c_2) \in \mathcal{N} \). Because of this, \((y_1 \ r : y_2)\) form a part of \( N_{\text{simplex}} \).

* **Case 1b**: \( y_1 \) is not a simplex node i.e. its in the domain of some successor hook \( h \).

\( x_1 \) cannot be the edge predecessor of \( x \) in this case because then \( y_1 \) would be simplex too, which is contrary to the supposition of the case. Thus \( x_1 \) must be the edge successor.

Further \( y_1 \) can be in the domain of another hook only if its edge predecessor \( x_1 \) is in the domain of another hook. Now, \( x \) is simplex, \( y_1 \) is in the domain of its successor hook \( h \), and so must be its edge predecessor \( x_1 \). Thus we conclude that \( x_1 \) is a hook. Let \( \text{Tail}(x_1) = b_1 \) and \( \text{Tail}'(x_1) = b_2 \) where \( b_1 \) blocks \( b_2 \).

In terms of the blocking scenario, we can conclude that \( b_1 \) blocks \( b_2 \), \( b_1^p \) is the predecessor of \( b_1 \) and \( a \) is the predecessor of \( b_2 \).

Now, \( \text{Tail}(x) = \text{Tail}'(x) = a \), and \((\forall, Rg, g'.r) \in \mathcal{L}^S(a) \). We know that \( a \) has the \( g' \) successor \( c_2 = \text{Tail}(y_2) \). Further, it has the \( R \) neighbour \( b_2 = \text{Tail}'(x_1) \). Also, \( b_1 = \text{Tail}(x_1) \) has the \( g \) successor \( c_1 = \text{Tail}(y_1) \).

By BC-2, this directly means that \( b_2 \) must also have a \( g \) successor \( c_2^{\text{down}} = \phi(c_1) \).

Also, there must be a \( g' \) successor of \( b_1^p \), say \( c_2^{up} = \phi'(c_2) \).

CCF condition (the **Reomplete** rule) ensure that \( (c_1^{\text{down}} \ r \ c_2) \in \mathcal{N} \). By BC-2, using the above, we conclude that \((c_1 \ r \ c_2^{up}) \in \mathcal{N} \).

Finally, \( (y_1 \ r \ y_2) \in \mathcal{N} \) would be as a part of \( N_{\text{pre}}^{\text{pre}} \), where \( y_1 \) is in the domain of \( h \) and \( y_2 \) is in the domain of \( e \) (same as \( x \)). \( (y_1 \ r \ y_2) \in \mathcal{N} \) because \((\text{Tail}(y_1)r \theta^r(\text{Tail}(y_2))) \in \mathcal{N} = (c_1 \ r \ theta(c_2)) = (c_1 \ r \ c_2^{up}) \in \mathcal{N} \).

– **Case 2**: \( x \) is not a simplex node.

Let \( x \) be in the domain of hook \( h \). Clearly, \( y_2 \) is also in the domain of \( h \). We now take
cases about \( y_1 \).

* **Case 2a**: \( y_1 \) is in the domain of \( h \).

This is a simple case and can be argued just like 1a.

* **Case 2b**: \( y_1 \) is in the domain of a predecessor hook \( h_p \).

\( x_1 \) cannot be the edge successor of \( x \) in this case, because then \( y_1 \) would have to be in domain of \( h \), which is contrary to the supposition of the case. Thus \( x_1 \) must be the edge predecessor.

Further \( y_1 \) can be in the domain of another hook only if its edge successor \( x_1 \) is in the domain of another hook. Now, \( x_1 \) is in the domain of \( h_p \) and its edge successor \( x \) is in the domain of \( h \). This is possible only if \( x \) is a hook. Let \( \text{Tail}(x) = a \) and \( \text{Tail}^\prime(x) = a' \) where \( a \) blocks \( a' \). So, \( x = (x_1 \mid (a,a')) \).

In terms of the blocking scenario, we can conclude that \( a \) blocks \( a' \), \( a^{p} \) is the predecessor of \( a \) and \( a_1 \) is the predecessor of \( a' \).

Now, \( \text{Tail}(x) = a, \text{Tail}^\prime(x) = a' \), and \((\forall \, Rg',g',r) \in L^S(a) = L^S(a'). \) Further, \( a_1 \), the predecessor of \( a' \) is the \( R \) neighbour of \( a \). BC-1 ensures \( a^{p} \) is the \( R \) neighbour of \( a \).

We know that \( a \) has the \( g' \) successor \( c_2 = \text{Tail}(y_2) \). We also know that \( a_1 \) has the \( g \) successor \( c_2 = \text{Tail}(y_2) \). By BC-2, this directly means that \( a^{p} \) must also have a \( g \) successor \( c_1^{up} = \phi^{-}(c_1) \). Also, there must be a \( g' \) successor of \( a' \), say \( c_2^{down} = \phi(c_2) \).

BC-2 and CCF condition (the Rcomplete) rule ensure that \((c_1 \, r \, c_2^{down}) \in N \) and \((c_1^{up} \, r \, c_2) \in N \).

Finally, \((y_1 \, r \, y_2) \in N \) would be as a part of \( N_b^{pre} \), where \( y_1 \) is in the domain of \( h_p \) (predecessor hook of \( h \)) and \( y_2 \) is in the domain of \( h \) (same as \( x \)). \((y_1 \, r \, y_2) \in N \) because \( (\theta^{-}(\text{Tail}(y_1)) \, r \, \text{Tail}(y_2)) \in N = (\theta^{-}(c_1) \, r \, c_2) = (c_1^{up} \, r \, c_2) \in N \).

* **Case 2c**: \( y_1 \) is in the domain of a successor hook \( h_s \).

This is much the same case as 1b. The only difference is that \( x \) was simplex there (in domain of some nominal tableau node \( e \)), but here it is in the domain of some hook \( h \).

* **Case 2d**: \( y_1 \) is a safe simplex node.

This means that \( x, y_2 \) is in the domain of some hook \( h \). Further, \( x_1 \) must be a nominal tableau node, and \( a_1 \) a nominal node in the completion system.

\((\forall \, Rg, g') \in a, a_1 \) is the \( R \) neighbour of \( a \), \( c_1 \) is the \( g \) successor of \( a_1 \), \( c_2 \) is the \( g' \) successor of \( a \). The CCF condition ensures that \((c_1 \, r \, c_2) \in N \).

Thus \((y_1 \, r \, y_2) \in N \) because \((\text{Tail}(y_1) \, r \, \text{Tail}(y_2)) = (c_1 \, r \, c_2) \) is in \( N \).

* \((P_c \, \neg \exists U)\):

The overall logic would be to show that if \( \neg(\exists U, U_1, U_2, r) \in L^T(x) \) and \( x \) has some \( U_1, U_2 \)
successors \(Y_1, Y_2\) such that \((y_1, r, y_2) \in \mathcal{N}\), then using cases just like in above propositions, we would be able to identify \(U_1, U_2\) neighbours of \(a = \text{Tail}(x)\) (say \(c_1, c_2\)) such that \((c_1, r, c_2)\) would have to be in \(\mathcal{N}\). But since this would have lead to a clash and \(S\) is CCF, therefore we prove by contradiction that none of the \(U_1\) and \(U_2\) successors of \(x\) are related by the constraint \(r\).

- \((P_c \leq c):\) If \((\leq n, g) \in \mathcal{L}^T(x)\), then \#\{\(y_i | y_i \in V^c, (x, y_i) \in \mathcal{E}^T(g)\} \leq n.\)

Let \(a = \text{Tail}(x)\). \(a\) can not be a blocked node (\(\text{Tail}(x)\) may but not \(\text{Tail}(x)\)). Clearly, \((\leq n, g) \in \mathcal{L}^2(a)\). Now, by the CCF condition of \(S\), there are \(\leq n\) successors of \(a\) (say \(c_i\)).

If there were any more successors than \(n\), then it would be a clash. Correspondingly we will have \(\leq n\) edge-successor of \(x\) such that \(y_i = (x | (c_i, c_i))\). All the \(y_i\) are clearly the \(g\) successors of \(x\).

- \((P_c \geq c):\) If \((\geq n, g) \in \mathcal{L}^T(s),\) then \#\{\(c_i | c_i \in \mathcal{L}^c \cap \mathcal{E}^T(g)\} \geq n.\)

In the completion system, \((\geq n, g) \in \mathcal{L}^2(a)\), and by the CCF condition, we have \(n\) distinct \(g\)-successors of \(a\) (\(c_i\)). Corresponding to every one of those \(c_i\), we will have a concrete successor of \(x\), say \(y_i\) such that \(\text{Tail}(y_i) = c_i\). Thus, \(x\) also has greater than \(n\) distinct \(g\) successors.

- \((P_c \forall i)\) and \((P_c \exists i):\)

This can be treated in a manner very similar to \((P_c \exists i)\) and \((P_c \forall i)\). wlog, assume \(U_1 = Rg\).

The only difference would be that in this case the constraint would always be introduced as a part of \(N^h\), where \(h\) is the hook in whose domain \(y_1\), the \(Rg\) successor of \(x\), is.

### A.3 Completeness

*If the KB is satisfiable i.e. there exists an augmented tableau \(T_a\) for it, then the tableau algorithm returns a complete and clash free completion system.*

Let \(T_a = (T, N), T = (V^a, V^c, \mathcal{L}^a, \mathcal{E}^a, \mathcal{E}^c)\) for it. We need to prove that the Tableau rules can be applied in a way to yield a complete and clash-free completion system, so that the tableau algorithm returns satisfiable. To this end, we use the augmented tableau \(T_a\) to ”guide” the application of tableau rules.

We introduce the notion of \(T_a\) compatibility of a completion system \(S\). We define mappings : \(\pi_a : V^S_a \rightarrow V^T_a\) and \(\pi_c : V^S_c \rightarrow V^c\). By means of these mappings, every node in the \(S\) is mapped to a node in \(T_a\). A completion system is said to be \(T_a\) compatible to a augmented tableau \(T_a\) if it obeys the following propositions. For the following we assume: \(a, b \in V^S_a; c_1, c_2 \in V^S_c; g \in N_c, R \in N_a\).

- \((C_a):\) For \(C \in \text{clos}'(T, R);\) if \(C \in \mathcal{L}^2(a)\), then \(C \in \mathcal{L}^T(\pi_a(a))\)
- \((C_b):\) If \(b\) is \(R\)-forward-neighbour of \(a\), in \(S\), then \((\pi_a(a), \pi_a(b) \in \mathcal{E}^T_a(R)\)
- \((C_c):\) If \(a \neq b\), then \(\pi_a(a) \neq \pi_a(b)\)
- \((C_d):\) If \(\leq nS.C \in \mathcal{L}^2(a)\) and \(\leq nS.C \notin \text{clos}'(T, R)\), the \#\(\mathcal{S}^T(a, C) = n\)

New clauses for \(\text{SROIQ}(C)\) :
\( \text{\textbullet (C_e)}: \) If \( c \) is \( g \)-successor of \( a \), in \( S \), then \( (\pi_a(a), \pi_c(c)) \in E^T_a(g) \)

\( \text{\textbullet (C_f)}: \) If \( (c_1 \ r \ c_2) \in N \), then \( (\pi_c(c_1) \ r \ \pi_c(c_2)) \in N \)

A \( T_a \) compatible completion graph is clash-free.

We prove how these condition ensure that the clashes new to \( SROIQ(C) \) e.r.t \( SROIQ \) do not occur.

- **Clash 1:**
  - This follows directly from \( C_e \) and \( C_c \). If suppose that the clash condition did happen in \( S \) for node \( a \), then it would mean that the \( P \leq \) tableau proposition is violated for \( \pi_a(a) \). But this can’t happen because the augmented tableau is known to be a valid one.

- **Clash 2:**
  - Assume \( \text{neq}(q_1 \ r \ q_2) \in Q \), and there exist \( c_1, c_2 \) such that \( q_1 \in M^S(c_1) \) and \( q_2 \in M^S(c_2) \). Now these could have got into \( M^S \) of \( c_1 \) and \( c_2 \) only if there was a node \( a \) such that \( \neg(\exists \ U_1, U_2, r) \in L^S(a) \) and \( c_1 \) is its \( U_1 \) neighbour and \( c_2 \) is its \( U_2 \) neighbour. (The case could also be \( \neg(\exists \ U, \{i\}, r) \) or \( \neg(\exists \ \{i\}, U, r) \) but they can be dealt analogously) Now the clash would occur if \( (c_1 \ r \ c_2) \in N \).
  - But this would mean that there exists \( \pi_a(a) \) such that \( \neg(\exists \ U_1, U_2, r) \in L^T(\pi_a(a)) \), there exist \( U_1, U_2 \) successors of \( y_1, y_2 \) and \( \forall \ U \) \( y_1 \ r \ y_2 \in N \). But this would make the augmented tableau invalid since it would not satisfy the proposition \( P_c \neg \exists \ U \). But its given that tableau is, indeed, valid. Thus, this clash can’t happen.

- **Clash 3:**
  - We prove this by contradiction. Suppose that \( N \) was, in fact, not satisfiable. Then we would have nodes in tableau given by the mapping \( \pi_e \) such that the constraint network between them is not satisfiable. But this is not possible, since the augmented tableau is by definition satisfiable.

Applicable rules can be applied to a \( T_a \) compatible system in such a way as to yield a new \( T_a \) compatible system.

We show this for the new Tableau Rules. For the other rules, the reader may refer to [7]. Assume w.l.o.g., \( U_1 = g_1, U_2 = Rg_2 \) for the concrete constructs; \( a \in V^\pi_a \).

Since \( T_a \) is satisfiable, we guess a completion of it, \( N^\pi \).

- **(R\exists U)**
  - \( \exists U_1, U_2, r \in L^S(a) \). Let \( U_1 = Rg, U_2 = g' \). Then by \( C_a, \exists U_1, U_2, r \in L^T(\pi_a(a)) \). Since \( T_a \) is valid, by \( P\exists U \) proposition, there must exist a \( g' \) successor of \( \pi_a(a) \) (say, \( y_2 \)) a \( R \) successor of \( \pi_a(a) \) (say \( x_1 \)) and a \( g \) successor of \( x_1 \) (say \( y_1 \)), such that \( (x_1 \ r \ x_2) \in N \). After application of rule, we create node \( a_1, c_1, c_2 \) such that \( a_1 \) is a \( R \) successor of \( a \), \( c_1 \) is the \( g \) successor of \( a \) and \( c_2 \) is a \( g' \) successor of \( a_1 \).
  - We set \( \pi_a(a_1) = x_1; \pi_c(c_1) = y_1; \pi_c(c_2) = y_2 \). Clearly, the new system is still \( T_a \) compatible.

- **(R\forall U)**
  - This is a deterministic rule. \( \forall U_1, U_2, r \in L^S(a) \). Let \( U_1 = Rg, U_2 = g' \). Then by \( C_a, \exists U_1, U_2, r \in L^T(\pi_a(a)) \). Since \( T_a \) is valid, by \( P\forall U \) proposition, all the \( U_1 \) and \( U_2 \) successors
are related by $r$ constraint. Consider any such $U_1, U_2$ successors $y_1, y_2$ such that $y_1 = \pi_c(c_1)$ and $y_2 = \pi_c(c_2)$. Clearly, $c_1, c_2$ would have to be the $U_1, U_2$ neighbours of $a$, where $\pi_a(a) = x$. Then by adding $(c_1 \ r \ c_2)$ to $N$, we can not have any clash.

- $(\text{Rcomplete})$:  
  Clearly the involved nodes have their respective counterparts in tableau, given by the mapping $\pi$. The network in tableau is guaranteed to be satisfiable. We use the corresponding constraints in $N^c$ to guess completions for this rule.

- $(\text{R} \leq_c)$:  
  If $((\leq_c n, g))$ is in $L^5(a)$, then it is there in $L^T(\pi_a(a))$ as well. By $P_c \leq_c$ proposition, we can be sure that $\pi_a(a)$ does indeed have less than $n \ g$ successors $y_i$. Now even if $a$ has more than $n$ successors $c_i$, they would still have to point to these $y_i$. Thus, we merge the required nodes such that ultimately there are less than $n \ c_i$ that point to these $y_i$, in a on-to-one manner.

- The rest of the propositions can be handled in manners analogous to above.

Clearly, the initial graph is $T_a$ compatible, with the mapping $\pi_c$ empty and $\pi_a$ mapping the nominal nodes and constraint individuals to their images in the augmented tableau. Since any applicable rule can be applied in a way to give a $T_a$ compatible system, we continue doing so until no more rules are applicable. Thus the completion system at the end would be complete and clash free.
References

[1] V. Milea, F. Frasincar, and U. Kaymak, “towl: A temporal web ontology language,” *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, vol. 42, no. 1, pp. 268–281, 2012.

[2] C. Lutz, F. Wolter, and M. Zakharyaschev, “Temporal description logics: A survey,” in *15th International Symposium on Temporal Representation and Reasoning, TIME 2008, Université du Québec à Montréal, Canada, 16-18 June 2008*, 2008, pp. 3–14.

[3] C. Lutz and M. Milicic, “A tableau algorithm for description logics with concrete domains and general tboxes,” *J. Autom. Reasoning*, vol. 38, no. 1-3, pp. 227–259, 2007.

[4] K. Sengupta, A. A. Krisnadhi, and P. Hitzler, “Local closed world semantics: Grounded circumscription for owl,” in *The Semantic Web - ISWC 2011 - 10th International Semantic Web Conference, Bonn, Germany, October 23-27, 2011, Proceedings, Part I*, ser. Lecture Notes in Computer Science, vol. 7031. Springer, 2011, pp. 617–632.

[5] I. Horrocks, O. Kutz, and U. Sattler, “The even more irresistible SROIQ,” in *In KR*. AAAI Press, 2006, pp. 57–67.

[6] I. Horrocks, “Optimising tableaux decision procedures for description logics,” Ph.D. dissertation, University of Manchester, 1997.

[7] I. Horrocks and U. Sattler, “A tableau decision procedure for SHOIQ,” in *J. of Automated Reasoning*, 2007.