A HYBRID ABOX CALCULUS USING ALGEBRAIC REASONING FOR THE DESCRIPTION LOGIC $\mathcal{SHIQ}$

LALEH ROOSTA POUR

A THESIS

IN

THE DEPARTMENT

OF

COMPUTER SCIENCE AND SOFTWARE ENGINEERING

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF MASTER OF COMPUTER SCIENCE

CONCORDIA UNIVERSITY

MONTRÉAL, QUÉBEC, CANADA

JANUARY 2012

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By:	**Laleh Roosta Pour**
Entitled:	**A hybrid ABox calculus using algebraic reasoning for the Description Logic** $\mathcal{SHIQ}$

and submitted in partial fulfillment of the requirements for the degree of

Master of Computer Science

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

______________________________ Chair
Dr. Brigitte Jaumard

______________________________ Examiner
Dr. Leila Kosseim

______________________________ Examiner
Dr. Nematollaah Shiri

______________________________ Examiner

______________________________ Supervisor
Dr. Volker Haarslev

Approved ______________________________
Chair of Department or Graduate Program Director

__________ 20 ______ ______________________________

Dr. Robin A. L. Drew, Dean
Faculty of Engineering and Computer Science

Abstract

A hybrid ABox calculus using algebraic reasoning for the Description Logic $\mathcal{SHIQ}$

Laleh Roosta Pour

We present a hybrid tableau calculus for the description logic (DL) $\mathcal{SHIQ}$. The presented algorithm decides $\mathcal{SHIQ}$ ABox consistency and uses an algebraic approach for more informed reasoning about qualified number restrictions (QNRs). Benefiting from integer linear programming and several optimization techniques to deal with the interaction of QNRs and inverse roles, our approach provides a more deterministic and informed calculus. In addition, a prototype reasoner based on the hybrid calculus has been implemented that decides concept satisfiability for $\mathcal{ALCHIQ}$. We provide a set of benchmarks that demonstrate the effectiveness of our hybrid reasoner in comparison to other DL reasoners.

Acknowledgments

I would like to express my deep and sincere gratitude to my supervisor, Dr. Volker Haarslev, for his important support throughout this work. With his enthusiasm, inspiration, great efforts to explain things clearly, and immense knowledge he guided me in all the time of research and writing of this thesis. His unflinching courage and conviction will always inspire me.

During this work I have collaborated with many colleagues and lab mates for whom I have great regard. I wish to extend my warmest thanks to, Mahsa Orang, Iman Keivanlou, Jinan El-Hashem, Kejia Wu, Ming Zuo, Amina Kane, Nasim Farsiniamarj.

I owe my deepest loving thanks to my husband, Amir, who was there for me during the thesis. Without his support and understanding this thesis could not have been accomplished. This work is as much his as it is mine.

Lastly, and most importantly, I wish to thank my parents, Ziba and Rahmat and my brothers, Amin and Vahid. They supported me, taught me, and loved me. To them I dedicate this thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

It is well known that standard tableau calculi for reasoning with qualified number/cardinality restrictions (QNR/QCRs) in description logics (DLs) have no explicit knowledge about set cardinalities implied by QNRs. This lack of information causes significant performance degradations for DL reasoners if the values of the numbers occurring in QNRs are increased. Over the last years a family of hybrid calculi have been developed that address this inefficiency by integrating integer linear programming (ILP) with DL tableau methods [HTM01, FFHM08, FH10b, FH10a, Fad11]. ILP is used to express and reason about cardinalities implied by QNRs, i.e., a set of QNRs is satisfiable iff the corresponding system of linear inequations has a non-negative integer solution. The hybrid calculus presented in this thesis follows this methodology.

Inspired by the calculus for $\mathcal{SHQ}$ [FH10b] we present a novel algorithm that decides ABox consistency for the description logic (DL) $\mathcal{SHIQ}$ [HST00b], which extends $\mathcal{SHQ}$ with inverse roles. This new calculus is a substantial extension of the one for $\mathcal{SHQ}$ since the interaction between inverse roles and QNRs results in back propagation of information specifically QNRs, and the loss of the finite model property, and requires pairwise blocking to deal with infinite models and cycles.

Inverse roles and qualified number restrictions increase the expressiveness of a language. QNRs express number restrictions on relationships between individuals and by inverse roles a relationship from one individual to another individual implies the inverse relationship in the opposite direction. For instance, assume the following expressions:

$$DBKRLab \sqsubseteq \leqslant 8\, isOccupiedBy.(\geqslant 1\, hasSupervisor.(Supervisor) \sqcap$$

$$\forall hasSupervisor.(KRProf \sqcup DBProf))$$

$$KRProf \equiv \forall supervises.(\geqslant 45\, hasCredit.(CS \sqcup SOEN) \sqcap$$

$$\leqslant 45\, hasCredit.(CS \sqcup SOEN) \sqcap \geqslant 16\, hasCredit.CS)$$

$$DBProf \equiv \forall supervises.(\geqslant 45\, hasCredit.(CS \sqcup MTH \sqcup SOEN) \sqcap$$

$$\leqslant 45\, hasCredit.(CS \sqcup MTH \sqcup SOEN) \sqcap \geqslant 16\, hasCredit.CS)$$

where $occupies$ is the inverse of $isOccupiedBy$ and $hasSupervisor$ is the inverse of $supervises$. Consider the two assertions: $lab901 : DBKRLab$ and $(mary, lab901) : occupies$. According to the inverse roles $mary$ is in the relation $isOccupiedBy$ with $lab901$ and considering the expressions, for $mary$ we have $\geqslant 1\, hasSupervisor.(Supervisor) \sqcap \forall hasSupervisor.(KRProf \sqcup DBProf)$. And due to the expression of the concepts $KRProf$ and $DBProf$, the QNRs such as:

$$(\geqslant 45\, hasCredit.(CS \sqcup MTH \sqcup SOEN) \sqcap \leqslant 45\, hasCredit.(CS \sqcup MTH \sqcup SOEN) \sqcap$$

$$\geqslant 16\, hasCredit.CS)$$

are propagated to the individual $mary$. The example shows how the combination of QNRs and inverse roles adds more expressiveness to a language. Adding more expressiveness to a language in general results in more complexity in solving the problem in that language. As we will discuss later in Chapter 6, our reasoner can handle cases that current reasoners cannot answer or might answer after hours of CPU time. Since our algorithm benefits from ILP and handles QNRs by capturing all numerical restrictions, it shows a significant improvement in case of large numbers occurring in QNRs.

1.1 Thesis Objectives and Contributions

Since existing approaches such as [HST00b, HS05] show a lack of efficiency in dealing with qualified number restrictions and inverse roles, we propose a hybrid algorithm for an expressive DL language $\mathcal{SHIQ}$, using arithmetic (or algebraic) reasoning.

This research mainly pursues the following objectives:

- Designing a decidable tableau based algorithm which benefits from integrating integer linear programming (ILP) and featuring qualified number restrictions and inverse roles.
- Evaluating such an algorithm for practical aspects. To this end, implementing a prototype reasoner and evaluating the performance compared to existing reasoners.

The given objectives guided our research and led to the following contributions:

- A hybrid tableau-based calculus for $\mathcal{SHIQ}$ ABox consistency is proposed. The algorithm uses arithmetic reasoning to capture the numerical restrictions.
- The termination, soundness, and completeness of the hybrid algorithm are proved.
- The worst-case complexity of the algorithm is analyzed.
- A set of optimization techniques and heuristics are studied in order to improve the efficiency of the algorithm in practical.
- A prototype reasoner with proper optimization techniques is implemented and we evaluated the effectiveness of the algorithm for practical purposes.
- The performance of the implemented reasoner is evaluated based on a set of synthesized benchmarks, which depict the behavior of the reasoner according to different parameters.

1.2 Thesis Outline

This thesis is organized as follows. Chapter 2 introduces DL, and specifically the DL $\mathcal{SHIQ}$ and some preliminaries. In Chapter 3, the two features of this work, QNRs and inverse roles, and the issues for the interaction between them are discussed. An ABox calculus for $\mathcal{SHIQ}$ is presented in Chapter 4 with examples and the proof of completeness, soundness, and termination of the calculus. Chapter 5 focuses on the analysis of the algorithm's complexity, and related optimization techniques, and includes a section on the implemented prototype reasoner. In Chapter 6, a set of synthesized benchmarks is presented. These benchmarks are tested against state of the art DL reasoners, and clearly demonstrate the effectiveness of our calculus. Finally we present our conclusion in Chapter 7 followed by some suggestions for future work.

Chapter 2

Preliminaries

In this chapter we review some definitions and preliminaries to introduce Description Logic (DL) and various DL languages. In section 2.1.2 we highlight the DL $\mathcal{SHIQ}$ which is the focus of our work. The DL services are explained in section 2.3 and the tableau-based reasoning algorithm for DL $\mathcal{SHIQ}$ are introduced. We also describe some well known DL reasoners.

2.1 Description Logics

Description Logics (DLs) are a family of formal knowledge representation (KR) languages which represent the knowledge of an application domain (the ”world”) [BCM$^+$07]. The three basic syntactic components of DL are atomic concepts, atomic roles and individuals.

Definition 1 (Concept)**.** *Concepts are subsets of the domain. A concept, represented by a unary predicate symbol, is a set of domain elements with similar characteristics. For instance in the domain of a family,* ***Male****,* ***Female****,* ***Child****,* ***Mother****,* ***Father****, and* ***Family*** *can be concepts.*

Definition 2 (Role)**.** *Roles are used to express binary relationships between concepts. For instance,* $hasChild$ *is a role that represents the relationship between two concepts,* ***Mother***

TBox	**ABox**
$\textbf{Mother} \equiv \textbf{Female} \sqcap \exists hasChild.\textbf{Child}$ $\textbf{Father} \equiv \textbf{Male} \sqcap \exists hasChild.\textbf{Child}$ $\textbf{Child} \equiv (\textbf{Female} \sqcup \textbf{Male}) \sqcap \exists hasParent.(\textbf{Female} \sqcup \textbf{Male})$ $\textbf{Female} \sqsubseteq \neg\ \textbf{Male}$ $\text{Inv}(hasChild) = hasParent$	$\langle \textbf{mary}, \textbf{mia} \rangle : hasChild$ $\textbf{mary} : \textbf{Female}$ $\textbf{mia} : \textbf{Child}$

Figure 1: Example of TBox and ABox (see below for syntax and semantics)

and ***Child****.*

Definition 3 (Individual)**.** *Individuals are instances of concepts. For example, if* ***mary*** *is an individual that belongs to the concept* ***Female****, then* ***mary*** : ***Female****. Also a relationship can be defined between a pair of individuals, for example* $\langle \textbf{mary}, \textbf{mia} \rangle : hasChild$ *where* ***mia*** : ***Child****.*

DLs languages feature reasoning as a central service which allows one to infer implicitly represented knowledge from the knowledge that is explicitly contained in the knowledge base. For example, if a **Mother** is a **Female** which has a $hasChild$ relationship with a **Child** and we have **mary** : **Female** and $\langle \textbf{mary}, \textbf{mia} \rangle : hasChild$, then the KR system infers that **mary** : **Mother**. More complex descriptions can be built from these basic concepts and roles, inductively with concept constructors.

DL was first introduced into Knowledge Representation (KR) systems to overcome the lack of formal (logic-based) semantics of frames and semantic networks [BHS07]. The first DL-based KR system was KL-ONE (by Ronald J. Brachman and Schmolze, 1985) and later in early '90s, a new tableau based algorithm paradigm allowed efficient reasoning on more expressive DL.

A KR system presented in DL language cannot only store terminologies and assertions in a Knowledge Base (KB), but also offers services that reason about it. A typical DL-based Knowledge Base consists of two main parts, TBox and ABox.

Definition 4 (TBox)**.** *The terminological part of a DL knowledge base is called TBox which* *includes facts about concepts and roles. A TBox $\mathcal{T}$ is a finite set of axioms in form of $C \sqsubseteq D$,called General Concept Inclusion axioms (GCIs), and/or $C \equiv D$ which is a placeholder for $\{C \sqsubseteq D, D \sqsubseteq C\}$.*

Definition 5 (ABox)**.** *The assertional part of a DL knowledge base is known as* ABox *which includes facts about individuals. An* ABox *$\mathcal{A}$ is a finite set of assertions of the forms $a : C$, $(a, b) : R$, and $a \neq b$ where a, b are the individuals occurring in $\mathcal{A}$ and R is a role.*

An example of a $TBox$ and $ABox$ is shown in Fig. 1. Let N_C, N_R, and I be three mutually disjoint sets of concept names, role names, and individual names. In the following we refer to A, B as atomic concepts ($A, B \in N_C$), R as a role name $R \in N_R$, and C, D as concepts expression (possibly not atomic concepts). I is the set of all individual names, while $I_A \subseteq I$ is the set of individual names occurring in an ABox $\mathcal{A}$. The set of roles is defined as $N_R \cup \{R^- \mid R \in N_R\}$. We define a function Inv such that Inv$(R) = R^-$ if $R \in N_R$ and Inv$(R) = S$ if $R = S^-$. For a set of roles $RO = \{R_1, \ldots, R_n\}$, Inv$(RO) = \{Inv(R_1), \ldots,$ Inv$(R_n)\}$.

2.1.1 DL $\mathcal{ALC}$

[SSS91] introduced the DL language $\mathcal{ALC}$, Attributive Concept Language with Complements, in 1991 which is the basis of many more expressive DL languages [BCM$^+$07]. Concept descriptions in $\mathcal{ALC}$ [1] are formed based on the following grammar, where $\top$ and $\bot$ denote respectively $(C \sqcup \neg C)$ and $(C \sqcap \neg C)$:

[1]In fact, the DL ALC corresponds to the fragment of first-order logic obtained by restricting the syntax to formulas containing two variables. Description Logics are notational variants of certain propositional modal logics [BCM$^+$07]; specifically, the DL $\mathcal{ALC}$ is a syntactic variant of the multi-modal logic K_M [HM92].

$$
\begin{array}{lrr}
C, D \rightarrow & C \,| & \text{(atomic concept)} \\
& \neg C \,| & \text{(atomic negation)} \\
& C \sqcap D \,| & \text{(conjunction)} \\
& C \sqcup D \,| & \text{(disjunction)} \\
& \forall R.C \,| & \text{(universal restriction)} \\
& \exists R.C \,| & \text{(qualified existential restriction)}
\end{array}
$$

We assume an interpretation $\mathcal{I} = (\varDelta^{\mathcal{I}}, .^{\mathcal{I}})$, where the non-empty set $\varDelta^{\mathcal{I}}$ is the domain of $\mathcal{I}$ and $.^{\mathcal{I}}$ is an interpretation function which maps each concept to a subset of $\varDelta^{\mathcal{I}}$ and each role to a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. The interpretation function is extended to concept descriptions by the following equations:

$$
\begin{aligned}
\top^{\mathcal{I}} &= \varDelta^{\mathcal{I}} \\
\bot^{\mathcal{I}} &= \emptyset \\
(\neg C)^{\mathcal{I}} &= \varDelta^{\mathcal{I}} \setminus C^{\mathcal{I}} \\
(C \sqcap D)^{\mathcal{I}} &= C^{\mathcal{I}} \cap D^{\mathcal{I}} \\
(C \sqcup D)^{\mathcal{I}} &= C^{\mathcal{I}} \cup D^{\mathcal{I}} \\
(\forall R.C)^{\mathcal{I}} &= \{s \in \varDelta^{\mathcal{I}} \,|\, \forall t \in \varDelta^{\mathcal{I}} : \langle s, t \rangle \in R^{\mathcal{I}} \Rightarrow t \in C^{\mathcal{I}}\} \\
(\exists R.C)^{\mathcal{I}} &= \{s \in \varDelta^{\mathcal{I}} \,|\, \exists t \in \varDelta^{\mathcal{I}} : \langle s, t \rangle \in R^{\mathcal{I}} \text{ and } t \in C^{\mathcal{I}}\}
\end{aligned}
$$

The expressiveness of a DL language depends on a set of constructors for building complex concepts and roles. Adding more constructors to $\mathcal{ALC}$ results in more expressive languages. The expressivity is encoded in the label for a logic using some letters. Each $\mathcal{ALC}$-language is named by a string of the form $\mathcal{AL}[\mathcal{C}][\mathcal{E}][\mathcal{N}][\mathcal{U}]$. For instance, $\mathcal{N}$ represents number restrictions. The DL $\mathcal{ALC}$ extended with transitive roles ($\mathcal{S}$) is named $\mathcal{S}$.

2.1.2 DL $\mathcal{SHIQ}$

In this work we focus on the expressive DL $\mathcal{SHIQ}$ which extends $\mathcal{ALC}$ with role hierarchy ($\mathcal{H}$), qualified number restrictions ($\mathcal{Q}$), transitive roles ($\mathcal{S}$), and inverse roles ($\mathcal{I}$). The Syntax and semantics of the DL $\mathcal{SHIQ}$ are shown in Fig. 2. Let $\sharp R^{\mathcal{I}}(x, C)$ denote the cardinality of the set $\{x \mid (x, y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$ and $(R^{\mathcal{I}})^{+}$ the transitive closure of $R^{\mathcal{I}}$, the interpretation $\mathcal{I}$ must satisfy the concept expressions in Fig. 2.

A *role hierarchy* $\mathcal{R}$ is a set of axioms of the form $R \sqsubseteq S$ where $R, S \in N_R$ and $\sqsubseteq_*$ is transitive-reflexive closure of $\sqsubseteq$ over $\mathcal{R} \cup \{\text{Inv}(R) \sqsubseteq \text{Inv}(S) \mid R \sqsubseteq S \in \mathcal{R}\}$. R is called a sub-role of S and S a super-role of R if $R \sqsubseteq_* S$. A role R is called simple if R is neither transitive nor has a transitive sub-role. $N_{RS} \subseteq N_R$ and $N_{RT} \subseteq N_R$ are respectively a set of simple role names and a set of transitive role names with $N_{RS} \cap N_{RT} = \emptyset$.

The set of $\mathcal{SHIQ}$ concepts is the smallest set such that (i) every concept name is a concept, and (ii) if C and D are concepts, R is a role, S is a simple role, $n, m \in \mathbb{N}, n \geq 1, m \geq 0$, then $C \sqcap D$, $C \sqcup D$, $\neg C$, $\forall R.C$, $\exists R.C$, $\geqslant nS.C$, and $\leqslant mS.C$ are also concepts.

2.2 DL Reasoning

As mentioned in the previous section, a knowledge representation system based on DLs consists of an ABox and a TBox, and has the ability to perform specific kinds of reasoning. The various kinds of reasoning performed by a DL system are introduced in [BCM+07]. Different types of inferences are defined for concepts, TBoxes, ABoxes, and for TBoxes and ABoxes together.

The basic form of reasoning is a concept satisfiability test. A concept C is satisfiable w.r.t to a TBox T if there exists a model $\mathcal{I}$ of T with $C^{\mathcal{I}} \neq \emptyset$ and $\mathcal{I}$ is called a model of C.

Constructor	**Syntax**	**Interpretation** $.^{\mathcal{I}}$
$\mathcal{ALC}$		
Top	$\top$	$\Delta^{\mathcal{I}}$
Bottom	$\bot$	$\emptyset$
Negation	$(\neg C)$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
Conjunction	$(C \sqcap D)$	$(C^{\mathcal{I}} \cap D^{\mathcal{I}})$
Disjunction	$(C \sqcup D)$	$(C^{\mathcal{I}} \cup D^{\mathcal{I}})$
Value Restriction	$(\forall R.C)$	$\{s \in \Delta^{\mathcal{I}} \mid \forall t \in \Delta^{\mathcal{I}} : \langle s,t\rangle \in R^{\mathcal{I}} \Longrightarrow t \in C^{\mathcal{I}}\}$
Existential Restriction	$(\exists R.C)$	$\{s \in \Delta^{\mathcal{I}} \mid \exists t \in \Delta^{\mathcal{I}} : \langle s,t\rangle \in R^{\mathcal{I}} \wedge t \in C^{\mathcal{I}}\}$
Transitive Roles		
$\mathcal{S}$	Trans(R)	$R^{\mathcal{I}} = (R^{\mathcal{I}})^{+}$, ($R^{\mathcal{I}}$ is transitive)
Role Hierarchy		
$\mathcal{H}$	$R \sqsubseteq S$	$R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$
Inverse Roles		
$\mathcal{I}$	Inv(R)	$\{\langle s,t\rangle \mid \langle t,s\rangle \in R^{\mathcal{I}}\}$
Number Restriction		
$\mathcal{N}$	$\geqslant nR.\top$	$\{s \mid \sharp R^{\mathcal{I}}(s,\top) \geqslant n, n \geqslant 1, n \in \mathbb{N}\}$
	$\leqslant mR.\top$	$\{s \mid \sharp R^{\mathcal{I}}(s,\top) \leqslant m, m \geqslant 0, m \in \mathbb{N}\}$
Qualified Number Restriction		
$\mathcal{Q}$	$\geqslant nR.C$	$\{s \mid \sharp R^{\mathcal{I}}(s,C) \geqslant n, n \geqslant 1, n \in \mathbb{N}\}$
	$\leqslant mR.C$	$\{s \mid \sharp R^{\mathcal{I}}(s,C) \leqslant m, m \geqslant 0, m \in \mathbb{N}\}$

Figure 2: Syntax and Semantics for $\mathcal{SHIQ}$

- **Regarding TBox**
  - *Concept satisfiability test*: A concept C is satisfiable w.r.t to a TBox T if there exists model $\mathcal{I}$ of T with $C^{\mathcal{I}} \neq \emptyset$ and $\mathcal{I}$ is called a model of C w.r.t T.
  - *Subsumption test*: An interpretation $\mathcal{I}$ satisfies a TBox $\mathcal{T}$ *iff* $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for every GCI $C \sqsubseteq D \in \mathcal{T}$. Such an interpretation is called a model of $\mathcal{T}$. In other words, there exists a model $\mathcal{I}$ of $\mathcal{T}$ which satisfies all the axioms in $\mathcal{T}$ and role hierarchy $\mathcal{R}$ [2].

[2]An interpretation $\mathcal{I}$ holds for a role hierarchy $\mathcal{R}$ iff $R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$ for each $R \sqsubseteq S \in \mathcal{R}$.

- **Regarding ABox**
  - *Instance checking*: instance checking answers the questions whether an ABox individual a is a member of a concept C w.r.t the relevant ABox assertions as well as the TBox.
  - *KB consistency test*: The KB $\mathcal{K} = (\mathcal{A}, T)$ is consistent if there exists a common model for ABox $\mathcal{A}$ and TBox T. An Abox $\mathcal{A}$ is consistent iff there exists a model $\mathcal{I}$ of $\mathcal{A}$. An interpretation $\mathcal{I}$ satisfies an ABox $\mathcal{A}$ if it satisfies $\mathcal{T}$ and a role hierarchy $\mathcal{R}$ and all assertions in $\mathcal{A}$ such that $a^{\mathcal{I}} \in C^{\mathcal{I}}$ if $a : C \in \mathcal{A}$ and $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$ if $(a, b) : R \in \mathcal{A}$. Such an interpretation is called a model of $\mathcal{A}$.

2.3 Reasoning Services

A knowledge representation system based on DLs is able to perform specific kinds of reasoning. The purpose of a knowledge representation system goes beyond storing concept definitions and assertions. A knowledge base comprising TBox and ABox, has a semantics that makes it equivalent to a set of axioms in first-order predicate logic. Thus, like any other set of axioms, it contains implicit knowledge that can be made explicit through inferences [BCM$^+$07]. For example, from the TBox and ABox in Fig. 1, one can conclude that **mia** is in $hasParen$ relationship with **mary** although this knowledge is not explicitly stated as an assertion.

2.3.1 Tableau Reasoning

Among different DL reasoning approaches such as *structural subsumption* [BPS94], *tableau-based* [SSS91], *automata-based* [BHLW03], *semantic binary tree* [Luk05], and *resolution-based* [KM06], the tableau-based approach remains the most popular one. In the early '90s, the first tableau based algorithm was introduced for $\mathcal{ALC}$ which allowed efficient reasoning on more expressive DL [SSS91]. Afterwards, the DL-based systems have been implemented using these tableau based algorithms, such as KRIS (1991) [BH91] and CRACK (1995) [BFT95], and show acceptable reasoning performance on typical inference problems even though the worst case complexity is no longer PTIME but at-least NP- or PSPACE-Hard.

Tableau algorithms use a set of expansion rules, so-called clash triggers, and possibly a set of blocking strategies to decide the satisfiability of a concept expression in *Normal Negation Form (NNF)* [BCM+07]. In order to test the satisfiability of a given concept C, the tableau algorithm tries to construct a model for it. If there exists a corresponding interpretation with $C^{\mathcal{I}} \neq \emptyset$, then C is satisfiable.

Definition 6 (Negation Normal Form)**.** *The concept expressions are in negation normal form, if the negation (¬) occurs only immediately in front of atomic concepts. In order to obtain the NNF of concept expressions, the following equations are used:*

$$\neg(C \sqcap D) \equiv \neg C \sqcup \neg D\,,\ \neg(C \sqcup D) \equiv \neg C \sqcap \neg D$$

$$\neg(\geqslant nR.C) \equiv \leqslant (n-1)R.C\,,\ \neg(\leqslant nR.C) \equiv \geqslant (n+1)R.C$$

$$\neg(\forall R.C) \equiv \exists R.\neg C\,,\ \neg(\exists R.C) \equiv \forall R.\neg C$$

$$\neg(\neg C) \equiv C$$

In order to test the satisfiability of the concept C, a model is constructed by the tableau-based algorithm which is a data structure called *completion graph.* To each node in the completion graph, a label will be assigned, which is a subset of possible concept expressions. Therefore, we define *clos* as the closure of a concept expression.

Definition 7 ($clos()$)**.** *The closure for a concept expression E, denoted as $clos(E)$, is the* *smallest set of concepts such that:*

$$E \in clos(E)$$

$$(\neg D) \in clos(E) \textit{ implies } D \in clos(E)$$

$$(C \sqcup D) \in clos(E) \textit{ implies } C \in clos(E), D \in clos(E)$$

$$(C \sqcap D) \in clos(E) \textit{ implies } C \in clos(E), D \in clos(E)$$

$$(\forall R.C) \in clos(E) \textit{ implies } C \in clos(E)$$

$$(\bowtie nR.C) \in clos(E) \textit{ implies } C \in clos(E)$$

where $\bowtie nR.C$ represents $\geqslant nR.C$ or $\leqslant nR.C$.

For a TBox $\mathcal{T}$, if $(C \sqsubseteq D \in \mathcal{T})$ or $(C \equiv D)$, then $clos(C) \subseteq clos(\mathcal{T})$ and $clos(D) \subseteq clos(\mathcal{T})$. Likewise for an ABox $\mathcal{A}$, if $(a : C) \in \mathcal{A}$ then $clos(C) \subseteq clos(\mathcal{A})$.

Definition 8 (Completion Graph)**.** *The completion graph $G = (V, E, \mathcal{L})$ is a directed graph in which V is a set of nodes representing individuals in the domain and E is a set of edges representing the relationship between individuals. For each node x, a label $\mathcal{L}(x)$ is assigned with $\mathcal{L}(x) \subseteq clos(C)$ and every edge $\langle x, y\rangle \in E$ between two nodes x, y, is labeled by a set of role names, $\mathcal{L}(\langle x, y\rangle) \subseteq N_R$.*

Definition 9 (-successor, -predecessor, -neighbor)**.** *Given a completion graph, for nodes x and y with $R \in \mathcal{L}(\langle x, y\rangle)$ and $R \sqsubseteq_* S$, y is called S-successor of x and x is Inv(S)-predecessor of y. If y is an S-successor or an Inv(S)-predecessor of x, then y is a S-neighbor of x. Finally, ancestor is the transitive closure of predecessor.*

Definition 10 (-filler)**.** *Given a completion graph, for nodes x and y with $R \in \mathcal{L}(\langle x, y\rangle)$, y is an R-filler (role-filler) for x. The R-fillers of x are defined as $Fil(x, R) = \{y \mid (x^{\mathcal{I}}, y^{\mathcal{I}}) \in R^{\mathcal{I}}\}$.*

The algorithm starts with $\mathcal{L}(x_0) = C$ for a given concept expression in NNF C. The graph G is expanded by means of expansion rules. These tableau completion rules preserve and build the dependencies implied by C. Tableau rules are defined based on the constructors and semantics of the used DL language. Fig. 3 displays the tableau completion rules for $\mathcal{ALC}$. For example, the $\sqcap$-Rule imposes the semantics of conjunction, as shown in Fig. 3.

$\sqcap$-Rule	if $(C \sqcap D) \in \mathcal{L}(x)$, x is not blocked, and $\{C, D\} \not\subseteq \mathcal{L}(x)$ then set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C, D\}$
$\sqcup$-Rule	if $(C \sqcup D) \in \mathcal{L}(x)$, x is not blocked, and $\{C, D\} \cap \mathcal{L}(x) = \emptyset$ then set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{X\}$ for some $X \in \{C, D\}$
$\forall$-Rule	if $\forall R.C \in \mathcal{L}(x)$, x is not blocked, and there exists an R-successor y of x with $C \notin \mathcal{L}(y)$ then set $\mathcal{L}(y) = \mathcal{L}(y) \cup \{C\}$
$\exists$-Rule	if $\exists R.C \in \mathcal{L}(x)$, x is not blocked, and there exists no R-successor y of x with $C \in \mathcal{L}(y)$ then create a node y and set $\mathcal{L}(\langle x, y\rangle) = \mathcal{L}(\langle x, y\rangle) \cup \{R\}$, and $\mathcal{L}(y) = \mathcal{L}(y) \cup \{C\}$

Figure 3: Tableau Completion Rules for DL $\mathcal{ALC}$

Some completion rules such as the $\exists$-Rule create new nodes, and some others such as the $\sqcap$-Rule extend the label of a node. There are rules such as the $\sqcup$-Rule, called non-deterministic rules. The non-deterministic rules are built due to the non-deterministic nature of particular concept expressions (e.g. $C1 \sqcup C2$). Therefore, the non-deterministic rules yield more than one outcome. In other words, the expression $C1 \sqcup C2$ is satisfiable if $C1$ is satisfiable or $C2$ is satisfiable. The $\sqcup$-Rule opens two branches to proceed in the search space in order to test the satisfiability of the concept expression. Consequently, two different completion graphs will be created.

When no more rules are applicable, then all implicit knowledge has been made explicit and the completion graph is said to be complete. If all possible completion graphs lead to the contradiction, known as a clash (see Def. 11), then the concept is un-satisfiable. However, only one complete and clash-free completion graph is enough in order to show that the concept is satisfiable. The algorithm stops whenever the completion graph is complete, or a clash occurs.

Definition 11 (Clash Triggers)**.** *A node x contains a clash if for a concept $A \in N_C$, $\{A, \neg A\} \subseteq \mathcal{L}(x)$.*

Assume a TBox $\mathcal{T}$ and atomic concepts A and B occuring in $\mathcal{T}$. If B occurs on the right-hand side of the definition of A, then A *directly uses* B and the transitive closure of the relation *directly uses* is called *uses*. If $\mathcal{T}$ consists of an atomic concept that uses itself, then $\mathcal{T}$ contains a *cycle*; otherwise $\mathcal{T}$ is called *acyclic*. For instance, assume $\mathcal{T}$ consists of axiom 1.

$$C \sqsubseteq \exists R.C \tag{1}$$

Then the application of completion rules in Fig. 3 leads to the completion graph shown in Fig. 4 with an infinite number of nodes. In such a case (presence of cycles), blocking techniques are needed to guarantee the termination of the algorithm. Without any blocking technique, $\exists R.C$ propagates through the nodes infinitely. If a proper blocking method is used, the resuling model contains a cycle, as shown in Fig. 5. In such a case, the blocked individual y can use the role successors of x instead of generating new ones.

Definition 12 (Blocked Node (subset blocking))**.** *A node y is directly blocked by a node x, if it has an ancestor node x such that $\mathcal{L}(y) \subseteq \mathcal{L}(x)$. The node y is blocked if it is directly blocked or one of its ancestors is blocked.*

The blocking strategy varies depending on the DL language. The DL $\mathcal{SHIQ}$ which is the focus of this work needs a more sophisticated blocking technique which is explained in Def. 17. Also, a new clash trigger is necessary due to the algebraic reasoning, which is explained in Def. 11.

Extending a DL language with a new constructor leads to a more expressive language. Consequently, according to the characteristics of the new language extended clash triggers and blocking techniques are inevitable. The standard tableau for $\mathcal{SHIQ}$, presented by [HST00b], is the focus of this work, as shown in Fig. 6. Our proposed tableau is demonstrated in Fig. 13.

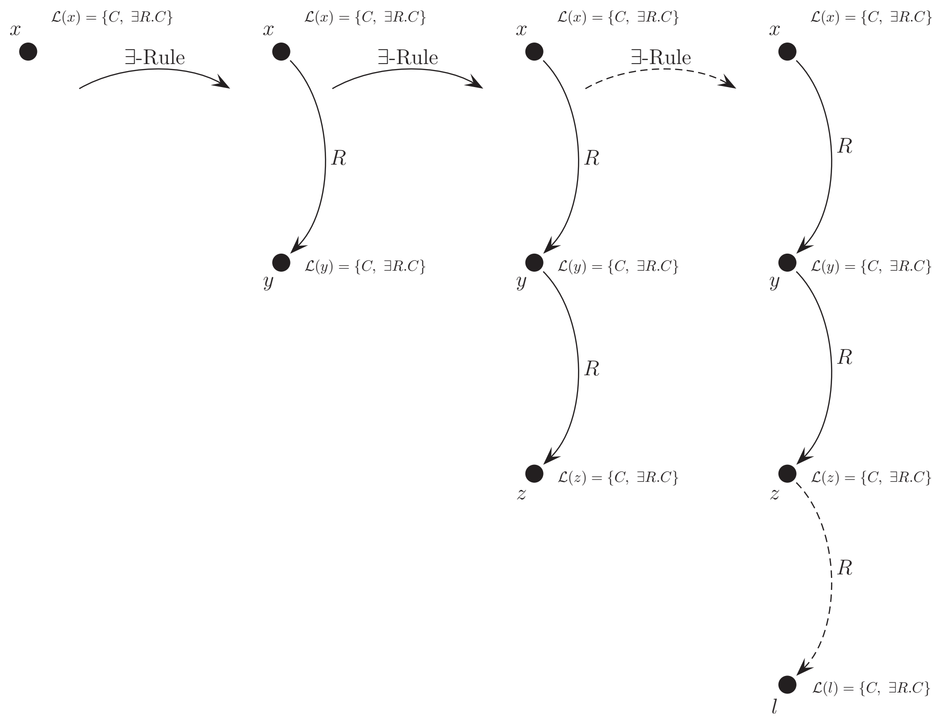

$\mathcal{L}(x) = \{C,\ \exists R.C\}$
x
$\exists$-Rule
$\exists$-Rule
$\exists$-Rule
R
$\mathcal{L}(y) = \{C,\ \exists R.C\}$
y
$\mathcal{L}(z) = \{C,\ \exists R.C\}$
z
$\mathcal{L}(l) = \{C,\ \exists R.C\}$
l

Figure 4: Application of completion rules without blocking

2.4 Complexity

As mentioned before, a KR system is expected to answer the query in reasonable time for a KB as input. Therefore, the reasoning procedures for a DL KR system from a decision procedure and should always terminate. Decidability and complexity of the inference problems depend on the expressive power of the supported DL. The more expressive a DL language is, the more likely it is that the DL reasoning is complex. Therefore, investigating a trade-off between the expressivity of DLs and their reasoning complexity has been one of the most important issues in DL research.

The complexity of a DL language is an inherent property of it. The complexity of a decidable language is usually determined based on the size of the completion model and the time needed to construct the model in the worst-case. The complexity of some DL languages are shown in Fig. 7. The presence of GCI, results in the EXPTIME-complete complexity for the DL languages which are extensions of $\mathcal{ALC}$. The complexity of different

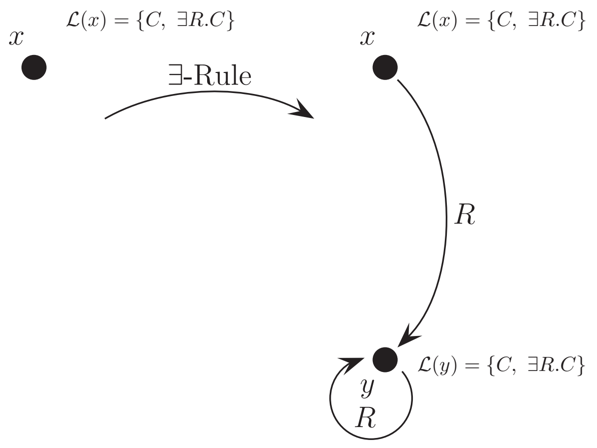

$\mathcal{L}(x) = \{C,\ \exists R.C\}$
x
$\exists$-Rule
$\mathcal{L}(x) = \{C,\ \exists R.C\}$
x
R
$\mathcal{L}(y) = \{C,\ \exists R.C\}$
y
R

Figure 5: Impact of blocking

possible DLs are represented in the DL Complexity Navigator[3].

The complexity analysis related to a DL language can be considered from different aspects. On one hand, it is computing of the complexity of a DL language on worst case, which preserves the theoretical aspect of DL reasoning. By complexity of a DL language we mean the complexity of the given problem for the corresponding language, and usually we talk about the satisfiability problem. On the other hand, it is the complexity of the reasoning algorithm in the worst case. There is a gap between these two aspects, and in order to find a reasonable threshold, the average cases will be considered. For instance [DM00] proposed a worst case optimal tableau-based procedure for the concept satisfiability problem of the DL $\mathcal{ALC}$. The proposed tableau-based procedure reduces the worst case complexity of 2ExpTime-complete to ExpTime-complete. It presented a *global sub-tableaux caching technique*, which has a significant improvement on practical tableau-based DL systems. The main challenge in DL language reasoning algorithm is to achieve a reasonable complexity for average cases in order to remain useful in practice.

[3]http://www.cs.man.ac.uk/ ezolin/dl/

$\sqcap$-Rule **if** $(C \sqcap D) \in \mathcal{L}(x)$, x is not indirectly blocked, and $\{C, D\} \nsubseteq \mathcal{L}(x)$
then set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C, D\}$

$\sqcup$-Rule **if** $(C \sqcup D) \in \mathcal{L}(x)$, x is not indirectly blocked, and $\{C, D\} \cap \mathcal{L}(x) = \emptyset$
then set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{X\}$ for some $X \in \{C, D\}$

$\forall$-Rule **if** $\forall R.C \in \mathcal{L}(x)$, x is not indirectly blocked,
and there exists an R-neighbour y of x with $C \notin \mathcal{L}(y)$
then set $\mathcal{L}(y) = \mathcal{L}(y) \cup \{C\}$

$\forall_+$-Rule **if** $\forall R.C \in \mathcal{L}(x)$, x is not indirectly blocked,
and there exists an S-neighbour y of x with $\forall S.C \notin \mathcal{L}(y)$,
where $Trans(S)$ and $S \sqsubseteq_* R$
then set $\mathcal{L}(y) = \mathcal{L}(y) \cup \{\forall S.C\}$

$\exists$-Rule **if** $\exists R.C \in \mathcal{L}(x)$, x is not blocked,
and there exists no R-successor y of x with $C \in \mathcal{L}(y)$
then create a node y and set $\mathcal{L}(\langle x, y\rangle) = \mathcal{L}(\langle x, y\rangle) \cup \{R\}$,
and $\mathcal{L}(y) = \mathcal{L}(y) \cup \{C\}$

choose-Rule **if** $(\bowtie nR.C) \in \mathcal{L}(x)$, x is not indirectly blocked,
and there exists an R-neighbour y of x with $\{C, \neg C\} \cap \mathcal{L}(y) = \emptyset$
then $\mathcal{L}(y) = \mathcal{L}(y) \cup \{E\}$ for some $E \in \{C, \neg C\}$

$\geqslant$-Rule **if** $\geqslant nR.C \in \mathcal{L}(x)$, x is not blocked,
and there are no $y_1, \ldots, y_n$ R-neighbours of x with $C \in \mathcal{L}(y_i)$,
and $y_i \neq y_j$ for $1 \leq i < j \leq n$
then create n new nodes $y_1, \ldots, y_n$ with $\mathcal{L}(\langle x, y_i\rangle) = \{R\}$, $\mathcal{L}(y_i) = \{C\}$,
and $y_i \neq y_j$ for $1 \leq i < j \leq n$

$\leqslant$-Rule **if** $\leqslant nR.C \in \mathcal{L}(x)$, x is not indirectly blocked,
and there are $y_1, \ldots, y_m$ R-neighbours of x with $C \in \mathcal{L}(y_i)$, $m \geqslant n + 1$,
and for y_i, y_j with not $y_i \neq y_j$ and y_j is not an ancestor of y_i
then $\mathcal{L}(y_i) = \mathcal{L}(y_i) \cup \mathcal{L}(y_j)$, $\mathcal{L}(\langle x, y_j\rangle) = \emptyset$,
and set $z \neq y_i$ for all z with $z \neq y_j$

Figure 6: Tableau Completion Rules for DL $\mathcal{SHIQ}$ TBox satisfiability [HST00b]

DL Language	Complexity
$\mathcal{ALC}, \mathcal{ALCQ}$	PSPACE-complete
$\mathcal{ALC}$ $+$ general TBox	EXPTIME-complete
$\mathcal{SHQ}, \mathcal{SHOQ}$	EXPTIME-complete
$\mathcal{SHIQ}$	EXPTIME-complete
$\mathcal{SHOIQ}$	NEXPTIME-complete
$\mathcal{SROIQ}$	2 NEXPTIME-complete

Figure 7: Complexity of different DL languages

2.4.1 DL Reasoner

DL reasoning is known to be very complex and will usually not terminate in reasonable time without a suitable set of optimization techniques. In order to improve the runtime of a DL reasoning algorithm, various optimization techniques were proposed [4] [Hor03], [THPS07].

In the following state of the art reasoners based on (hyper)-tableau algorithms are introduced.

- FaCT++ [5], is a new generation of the FaCT DL reasoner, which supports $\mathcal{SHF}$ and expressive language $\mathcal{SHIQ}$ [6]. FaCT++ is a highly optimized DL reasoner that supports OWL DL [7] and partially OWL 2.

- RacerPro [8] [HM01b], is a DL reasoner supporting the DL $\mathcal{SHIQ}$. RacerPro is a

[4]http://dl.kr.org/dig/optimisations.html
[5]http://owl.man.ac.uk/factplusplus/
[6]http://www.cs.man.ac.uk/ horrocks/FaCT/
[7]http://www.w3.org/TR/owl-ref/
[8]http://www.racer-systems.com/

highly optimized reasoner that uses algebraic reasoning [HTM01] and also the signature calculus [HM01a] in order to deal with QNRs. We will explain these two techniques in Chapter 4.3 and Chapter 3.

- Pellet [9] [SPG$^+$07], is a DL reasoner supporting OWL 2.

2.5 Summary

In this chapter we introduced Description Logic languages and some basic definitions and preliminaries. DL languages in terms of their syntax, semantics, and inference services were defined. The impact of increasing the expressiveness of a DL language on the complexity of the DL language was discussed. We distinguished between the theoretical complexity (language complexity) and practical complexity (algorithm complexity to reason the language). Some of the well known reasoners were introduced and it was explained that the naive implementation of tableau algorithm led to unreasonable reasoning time. In the next chapter we introduce more definitions required in order to present our algorithm.

[9]http://clarkparsia.com/pellet/

Chapter 3

DL Reasoning with Qualified Number Restrictions and Inverse Roles

This chapter discusses the extension of DL language with qualified number restrictions (QNRs) $\mathcal{Q}$ and inverse roles $\mathcal{I}$.

Adding Number Restrictions $\mathcal{N}$, to a DL language results in a more expressive language. By means of number restrictions, the ability of counting is added to the language. For instance, using cardinality restrictions these forms of axioms are expressible:

$$\begin{aligned} \textbf{Child} &\sqsubseteq \ \geqslant 1\, hasParent.\top \\ \textbf{GraduateCSStudent} &\sqsubseteq \ \geqslant 4\, passCourse.\top \end{aligned}$$

By using qualified number restrictions ($\mathcal{Q}$), the range of a role in number restrictions could be qualified as follows:

$$\begin{aligned} \textbf{Child} &\sqsubseteq \ \geqslant 1\, hasParent.(\textbf{Male} \sqcup \textbf{Female}) \\ \textbf{GraduateCSStudent} &\sqsubseteq \ \geqslant 4\, passCourse.(\textbf{SECourse} \sqcup \textbf{CSCourse}) \end{aligned}$$

Since qualified number restrictions are more expressive and complicated than unqualified ones, we focus on QNRs which automatically cover unqualified number restrictions. Extending $\mathcal{ALC}$ to $\mathcal{SHQ}$ leads to a complexity increase from PSPACE-complete to EXPTIME-complete. A DL reasoning algorithm for $\mathcal{SHQ}$ is studied in [FH10b], which uses algebraic reasoning and a set of optimization techniques and shows significant improvement in efficiency of practical reasoning. We extend that work, considering inverse roles, which complicates the reasoning due to its nature of back propagation. In contrast to $\mathcal{SHQ}$, the information may propagate back from a lower level of a tree model to a higher level.

3.1 Standard Tableau

The rules that deal with qualified number restrictions in standard tableau-based algorithms [HB91, BBH96, HS05, HST00b] are shown in Fig. 8.

choose-Rule	**if** $(\bowtie nR.C) \in \mathcal{L}(x)$, x is not blocked, and there exists an R-**successor** y of x with $\{C, \neg C\} \cap \mathcal{L}(y) = \emptyset$ **then** $\mathcal{L}(y) = \mathcal{L}(y) \cup \{E\}$ for some $E \in \{C, \neg C\}$
$\geqslant$-Rule	**if** $\geqslant nR.C \in \mathcal{L}(x)$, x is not blocked, and there are no $y_1, \ldots, y_n$ R-**successors** of x with $C \in \mathcal{L}(y_i)$, and $y_i \dot{\neq} y_j$ for $1 \leq i < j \leq n$ **then** create n new nodes $y_1, \ldots, y_n$ with $\mathcal{L}(\langle x, y_i \rangle) = \{R\}$, $\mathcal{L}(y_i) = \{C\}$, and $y_i \dot{\neq} y_j$ for $1 \leq i < j \leq n$
$\leqslant$-Rule	**if** $\leqslant nR.C \in \mathcal{L}(x)$, x is not blocked, and there are $y_1, \ldots, y_m$ R-**successors** of x with $C \in \mathcal{L}(y_i)$, $m \geqslant n+1$, and for y_i, y_j with not $y_i \dot{\neq} y_j$ and y_j is not an ancestor of y_i **then** $\mathcal{L}(y_i) = \mathcal{L}(y_i) \cup \mathcal{L}(y_j)$, $\mathcal{L}(\langle x, y_j \rangle) = \emptyset$, and set $z \dot{\neq} y_i$ for all z with $z \dot{\neq} y_j$

Figure 8: Tableau completion rules dealing with QNRs [HB91]

These three rules together satisfy the QNRs. The $\geqslant$-Rule preserves the *at-least* semantic by creating n distinct ($\dot{\neq}$) R-fillers for a corresponding node x and set their labels to C, if $\geqslant nR.C \in \mathcal{L}(x)$. If $\leqslant mR.C \in \mathcal{L}(x)$, it means that the R-fillers of x with D in their label should be counted and their number must be less than or equal to m. Due to the *open*

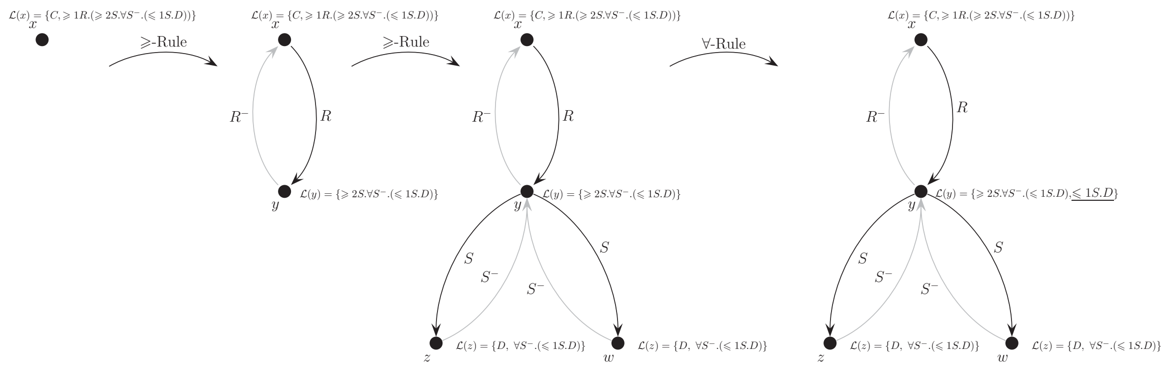

$\mathcal{L}(x) = \{C, \geqslant 1R.(\geqslant 2S.\forall S^{-}.(\leqslant 1S.D))\}$
x
$\geqslant$-Rule
$\geqslant$-Rule
$\forall$-Rule
R^{-}
R
y
$\mathcal{L}(y) = \{\geqslant 2S.\forall S^{-}.(\leqslant 1S.D)\}$
S
S^{-}
z
w
$\mathcal{L}(z) = \{D,\ \forall S^{-}.(\leqslant 1S.D)\}$
$\mathcal{L}(y) = \{\geqslant 2S.\forall S^{-}.(\leqslant 1S.D), \underline{\leqslant 1S.D}\}$

Figure 9: Impact of inverse roles interacting with QNR. A QNR is propagated back to a node.

world assumption of DLs, and in order to be sound and complete, the $choose$-Rule non-deterministically assigns C or $\neg C$ to the label of R-fillers of x. If x has k R-fillers, this semantic branching lead to 2^k branches in the search space for $\leqslant mR.C$. [Hor02] explains that this non-deterministic rule can be a major source of inefficiency in most DL-reasoners, by showing sample ontologies derived from UML diagrams.

The $\leqslant$-Rule preserves the *at-most* semantics. After counting k R-fillers, if $\leqslant mR.C$ is violated by $k > m$, the $\leqslant$-Rule merges two nodes in order to satisfy the *at-most* restriction. Whenever the $\leqslant$-Rule cannot merge nodes due to assertions of the form $y \dot{\neq} z$, and consequently cannot relax the source of the violation, then a clash occurs. Otherwise, it will non-deterministically try to merge $k \geqslant 1$ nodes. Since there exists $(k-m)$ extra successors, there are $\binom{k}{2}\binom{k-1}{2}\ldots\binom{m+1}{2}/(k-m)!$ ways to merge them. This number grows by a rate of $\binom{k+1}{2}/(k+1-m)$ when $(k-m)$ increases. Hence, the $\leqslant$-Rule can also be considered as a significant source of non-determinism and consequently, inefficiency. We extend the definition of clash triggers represented in Def. 11 to cover cases of violation in QNRs,when merging is required, but the $\leqslant$-Rule (a.k.a $merge$-Rule) fails to merge. Therefore, a clash occurs whenever for an individual x, an expression $x \neq x$ is added. This is due to the fact that, an at-most restriction forces merging between two individuals x and y with $x \neq y$, and $x \neq x$ implies that a merging is necessary but there exists no pair of nodes with required precondition, hence an individual is merged with itself.

In case of inverse roles, only the phrase ***-successor*** in Fig. 8 will be replaced by ***-neighbour*** and the condition for blocking distinguishes *indirectly blocked* nodes[1]. In other words, the nature of inverse roles would be considered in the definition of *-neighbour* (see Def. 9). Fig. 6 shows the rules in presence of inverse roles.

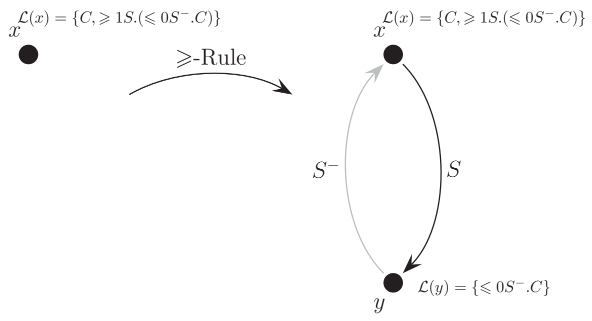
By adding inverse roles to the DL $\mathcal{SHQ}$, more expressiveness is added to the resulting DL $\mathcal{SHIQ}$, however, obtaining such an expressiveness comes with more complicated problems to deal with. In the following, challenges regarding the interaction between inverse roles and QNRs are discussed. Inverse roles provide the capability of back propagating information to the nodes in the completion graph.

A possible information that can be propagated back is a QNR. Fig. 9 demonstrates an example for the back propagation of a QNR. Assume the concept expression:

$$C \sqcap \geqslant 1R.(\geqslant 2\,S.\forall S^{-}.(\leqslant 1\,S.D)) \tag{2}$$

Fig. 9 displays the steps of generating a model by the concept satisfiability test of (2). As shown, the expression $\forall S.(\leqslant 1\,S.D)$ in the label of z and w propagates $(\leqslant 1S.D)$ back to the node y. This new QNR may affect the previous state of node y. For instance, the outgoing edges of node y may be affected due to the propagated QNR. In Fig. 9, $(\leqslant 1\,S.D)$ dose not violate any previous number restrictions and the related edges to y do not need any changes. Assume that instead of $(\leqslant 1\,S.D)$, concept expression (2) contained $(\leqslant 1\,S)$. In this case, the propagation of $(\leqslant 1\,S)$ violates the previous QNR in the label of y $(\geqslant 2S, \forall S^{-}.(\leqslant 1\,S.D))$ and makes the concept unsatisfiable. A new back propagated QNR may force the outgoing edges of y to be merged. Also, if a QNR such as $\geqslant 2T.D$ is propagated back, then the new T-successors should be created. Various cases may happen due to back propagated QNRs that the reasoning algorithm should deal with.

[1]A node y is indirectly blocked *iff* one of its ancestors is blocked, or it is a successor of a node x and $\mathcal{L}(\langle x, y\rangle) = \emptyset$. The second condition avoids wasted expansions after an application of the $\leqslant$-Rule (see Fig. 6 for the explanation of the $\leqslant$-Rule).



$\mathcal{L}(x) = \{C, \geqslant 1S.(\leqslant 0S^-.C)\}$
x
$\geqslant$-Rule
$\mathcal{L}(x) = \{C, \geqslant 1S.(\leqslant 0S^-.C)\}$
x
S^-
S
$\mathcal{L}(y) = \{\leqslant 0S^-.C\}$
y

Figure 10: Impact of inverse roles interacting with QNRs. The IBE S^- imposes a numerical restriction on node y.

In order to address another challenge, assume the concept expression:

$$C \sqcap \geqslant 1\, S.(\leqslant 0\, S^-.C) \tag{3}$$

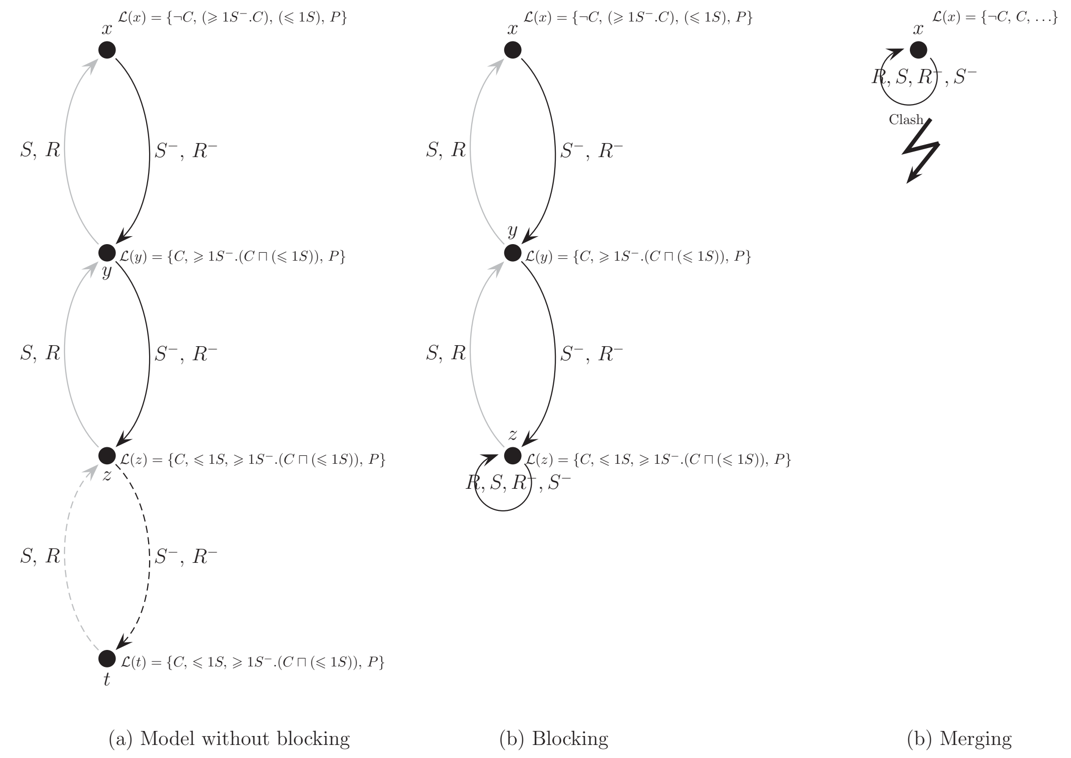
A model generated for this concept is shown in Fig. 10. The edge labeled with S from x to y, implies the edge from y to x, called *Implied Back Edge (IBE)*. Therefore the expression $(\leqslant\ 0\, S^-.C)$ in the label of node y cannot be satisfied due to the existence of the S^--neighbour, x, with C in its label. Therefore, the IBE imposes a numerical restrictions on y.

3.1.1 Pair-Wise Blocking

The combination of inverse roles and qualified number restrictions in $\mathcal{SHIQ}$ adds the infinite model property to the DL [HST99]. This means that there are satisfiable concepts for which, there exists no finite model. For such a case, the blocking defined in Def. 12 would block incorrectly and not discover satisfiability. To this end, a blocking technique called *pair-wise blocking*, is proposed by [HST00b], which requires two pairs of nodes and the edge between them instead of only two similar nodes.

Assume a satisfiable concept expression:

$$\neg C \sqcap (\geqslant 1S^-.C) \sqcap (\leqslant 1S) \sqcap \underbrace{\forall R^-.(\geqslant 1S^-.(C \sqcap (\leqslant 1S)))}_{P}$$



$\mathcal{L}(x) = \{\neg C, (\geqslant 1S^{-}.C), (\leqslant 1S), P\}$
x
S, R
S^{-}, R^{-}
$\mathcal{L}(y) = \{C, \geqslant 1S^{-}.(C \sqcap (\leqslant 1S)), P\}$
y
$\mathcal{L}(z) = \{C, \leqslant 1S, \geqslant 1S^{-}.(C \sqcap (\leqslant 1S)), P\}$
z
$\mathcal{L}(t) = \{C, \leqslant 1S, \geqslant 1S^{-}.(C \sqcap (\leqslant 1S)), P\}$
t
R, S, R^{-}, S^{-}
$\mathcal{L}(x) = \{\neg C, C, \ldots\}$
Clash
(a) Model without blocking
(b) Blocking
(b) Merging

Figure 11: Applying subset blocking on Infinite model

where R is a transitive role and $S \sqsubseteq R$. A model for this concept is demonstrated in Fig. 11. The expression $S \sqsubseteq R$ imposes the occurrence of R wherever S occurs. Since R is transitive then R^{-} is also transitive. Therefore the expression P and consequently $\geqslant 1S^{-}.(C \sqcap (\leqslant 1S))$ would be propagated along the model. As shown in Fig. 11(a), the model of this concept contains an infinite sequence of individuals and all individuals after z have the same labels. Considering the blocking preconditions in Def. 12, z blocks its successor and all incoming and outgoing edges of the successor (blocked node) will be transferred to individual z (blocking node). Existence of two edges with S in their labels violates the expression $\leqslant 1S$ (as demonstrated in Fig. 11(b)). In order to satisfy $\leqslant 1S$, the S-neighbors of z should be merged (see the $\leqslant$-Rule in Fig. 6). Therefore, the whole sequence collapses into a single node as shown in Fig. 11(c) and this results in a contradiction as both C and $\neg C$ will be in the node x's label. [HST99] also explained a case in which the algorithm with subset blocking defined in Def. 12 technique could not discover the unsatisfiability of a concept.

In order to ensure that the algorithm terminates correctly even for a concept with only finite models, a more sophisticated blocking technique is needed. For this purpose, [HST00b] proposed the *pair-wise blocking* technique. The pair-wise blocking technique, establishes blocks between pairs of nodes connected by the same role. Node y is blocked by node x, also called witness, if $\mathcal{L}(x) = \mathcal{L}(y)$ and for their successors y', x', $\mathcal{L}(y') = \mathcal{L}(x')$ and $\mathcal{L}(\langle x, x' \rangle) = \mathcal{L}(\langle y, y' \rangle)$. To ensure that the blocked node is not expanded anymore, the new strategy replaces the blocked node with the tree rooted at the blocking node recursively.

3.1.2 Optimization Technique

As it was mentioned before, a naive implementation for a DL reasoner even for not very expressive DL languages leads to poor efficiency. In order to get an answer in a reasonable time optimization techniques are necessary.

There exist various methods of optimization [BCM$^+$07] for tableau-based reasoning such as *absorption, pseudo-model merging* and *caching*. [KM06] proposes a resolution-based reasoning procedure which is proven to be weak to deal with large numbers in QNRs. Hyper-tableau presented in [MSH07], combines tableau and resolution-based [KM06] reasoning and were recently studied to minimize the size of created models and their degree of non-determinism in DL reasoning with no special treatment for QNRs.

There are some techniques that have been aimed at non-determinism due to the handling of disjunctions of concepts during a pre-processing phase. Pre-processing techniques are performed directly on the syntax of the input to render it more amenable to reasoning and processing. These techniques examine the syntactic structure of input concept expressions and exploit relations (tautology, clash) which are obvious, and can significantly speed up subsequent reasoning. Some of the widely used pre-processing techniques are *lazy unfolding*, *internalization*, and *absorption* [HT00]. First performance improvements for tableau-based DL systems addressing QNRs have been reported in [HM01a, HTM01] and more recently in [FFHM08] and [FH10b].

In the following, some well known optimization techniques more specified for numerical restrictions will be described. [HM01a], presents the *signature* calculus to improve the efficiency of the algorithm for the large values of QNRs. The algebraic reasoning [HTM01, OK99, FH10b, Fad11], provide the ability of choosing an arithmetically informed branch when creating successors. As opposed to the previous techniques, which optimize the algorithm when creating successors, *dependency directed backtracking* presented in [Hor02], detects the source of a clash in order to prevent the reproduction of the same clash again.

- **Signature Calculus**. The complexity of the standard tableau algorithms is evidently a function of the value of numbers occurring in QNRs; i.e., m and n in $(\leqslant mR.C)$ and $(\geqslant nR.C)$ (see Section 3). Increasing n in $(\geqslant nR.C)$, results in an increase of the number of R-successors of the corresponding node and an exponential increase of possible outcomes from the $choose$-Rule. On the other hand, the number of possible ways to merge n nodes into m nodes grows tremendously due to the increase of m and n. One way to handle large numbers of successors is to create a *proxy individual* to represent more than one R-successor when all the successors share the same label. The signature calculus presented in [HM01a], similarly, creates proxy individuals as role fillers. In other words, for each at-least restriction $\geqslant nR.C$ one proxy individual as instance of C is created, which represents n individuals. However, the calculus might later split the proxy individual into more than one proxy individual, in order to satisfy the constraints imposed by the restrictions on the role-fillers. For example, if $(\leqslant mR.C)$ is in the label of a proxy individual x, where $m < n$ it non-deterministically tries to split x into more than one proxy individual. In addition, it also requires a merge rule which non-deterministically merges extra proxy individuals that violate the at-most restriction.

- **Dependency Directed Backtracking**. Another way to optimize reasoning, in general, is dependency-directed backtracking (a.k.a. back-jumping) [Hor02]. By means of back-jumping, an algorithm can detect the source of a clash and prune the search space to avoid facing the same clash again. Note that the only source of branching in the search space is due to the non-deterministic rules. Non-determinism is the reason that makes tableau algorithms inefficient. Therefore, back-jumping can significantly improve the performance of the highly non-deterministic calculi. The rules handling qualified number restrictions are a considerable source of non-determinism; i.e., the $\leqslant$-Rule and the $choose$-Rule. Therefore, dependency directed backtracking can optimize these algorithms even in the absence of large numbers [Hor02]. The technique described in [Hor02], records the sources of a clash and jumps over choice points that are not related to the clash and tries to choose another branch at a non-deterministic point that is related to this clash.

- **Algebraic Reasoning**. Combining algebraic methods introduced in [OK99] with tableau-based approaches in [HTM01], a hybrid algorithm is proposed to decide consistency of general $\mathcal{SHQ}$ TBoxes [FH10b]. This approach partitions the set of role-fillers so that, successors are created in a more informed way in order to avoid merging them later. Therefore, merging, which is a significant source of non-determinism, is prevented.

3.2 Summary

This chapter explained QNRs, and completion rules dealing with QNRs in tableau based algorithms. Afterwards, the interaction between QNRs and inverse roles were discussed and the challenges were described. We also discussed some well known optimization techniques which we used in our work. In the next chapter, atomic decomposition, arithmetic reasoning and finally the hybrid calculus will be introduced.

Chapter 4

Hybrid Algebraic-Tableau Calculus for DL $\mathcal{SHIQ}$

This chapter proposes a hybrid tableau calculus for the DL $\mathcal{SHIQ}$ which is introduced in Section 2.1.2. The hybrid algorithm is a tableau-based algorithm which benefits from an algebraic component, while still maintaining termination, soundness, and completeness. The hybrid algorithm handles qualified number restrictions (QNRs) by means of an algebraic component and ensures that the interaction with inverse roles is preserved. In addition to QNRs, the algorithm captures other constraints that imply arithmetic constraints such as implied inverse roles, which will be explained in the following. Therefore, it is a hybrid algorithm which is more informed about arithmetic constraints imposed by concept descriptions.

The algorithm preserves the semantics of inverse roles as numerical restrictions. In other words, whenever an edge has been created, an implicit inverse edge is implied. This *Implied Back Edge* (IBE) is considered as a set of new number restrictions. For example, in order to model the concept expression $(\geqslant 1\, R.C)$, a node x will be created and $(\geqslant 1\, R.C)$ will be added to $\mathcal{L}(x)$. Then to satisfy $(\geqslant 1\, R.C)$, an R-successor y for x will be generated. Since R has an implied inverse role (R^{-}), the edge from x to y imposes an R^{-} edge from y

to x, which we refer to as an Implied Back Edge. We preserve this edge by a set of number restrictions $\{\geqslant 1\, R^-_{yx}, \leqslant 1\, R^-_{yx}\}$ related to node y. Therefore, the nature of inverse roles is captured in the form of numerical restrictions which will be handled by the algebraic component.

The tableau-based reasoning is based on a standard tableau for $\mathcal{ALC}$ [BS01] modified and extended to work with an algebraic reasoning component. Algebraic reasoning is based on the assumption that domain elements consist of a set of individuals divided into subsets depending on their role-filler membership. QNRs represents number restrictions on their corresponding sets, that is, QNRs represent at-least and at-most restrictions on the number of corresponding sets of role-fillers.

The hybrid algorithm is designed for ABox satisfiability therefore the constructed model will be a *completion forest*. Note that we refer to the tree model as a tree with specific characteristics, such that a node in a lower level (near to the leaf) may have an edge to its creator node.

Definition 13 (Completion forest)**.** *The algorithm generates a model consisting of a set of arbitrarily connected individuals in $I_{\mathcal{A}}$ as the roots of completion trees. Ignoring the connections between roots, the created model is a forest $\mathcal{F} = (V, E, \mathcal{L}, \mathcal{L}_E, \mathcal{L}_I)$ for a $\mathcal{SHIQ}$ ABox $\mathcal{A}$. Every node $x \in V$ is labeled by $\mathcal{L}(x) \subseteq clos(A)$ and $\mathcal{L}_E(x)$ as a set of inequations of the form $\sum_{i \in \mathbb{N}} v_i \bowtie n$ with $n \in \mathbb{N}$ and $\bowtie \in \{\geqslant, \leqslant\}$ and variables $v_i \in \mathcal{V}$. Each edge $(x, y) \in E$ is labeled by the set $\mathcal{L}(x, y) \subseteq RN$. For each node x, $\mathcal{L}_I(x)$ is defined to keep an implied back edge for x equivalent to $Inv(\mathcal{L}(y, x))$, where y is the parent of x (see Def. 15). For nodes with no parents, $\mathcal{L}_I$ will be the empty set.*

4.1 Pre-processing

Before applying the completion rules, the algorithm modifies the input ontology in two steps. One step would be *re-writing ABox assertions* in order to capture numerical restrictions in terms of number restrictions and the other one, *re-writing QNRs to NRs*.

4.1.1 Re-writing ABox assertions

ABox role assertions are translated into number restrictions since they actually impose a numerical restriction on a node. The assertion $(a, b) : R$ will be replaced by $b : (\geqslant 1\, R_{ab}) \sqcap (\leqslant 1\, R_a b)$, and $\{R_{ab} \sqsubseteq R\}$ since the assertion $(a, b) : R$ means there exists one and only one R_{ab}-filler for b which is c. Since the hybrid algorithm needs to consider all the numerical restrictions before creating an arithmetic solution and generating the successors for a node, it is necessary to consider the ABox assertion as well.

4.1.2 Re-writing QNRs to NRs

Inspired by [OK99], we use a satisfiability-preserving rewriting to replace QNRs ($\mathcal{Q}$) with unqualified ones ($\mathcal{N}$). This rewriting uses a new role-set difference operator $\forall(R \setminus R').C$ for which $(\forall(R \setminus R').C)^{\mathcal{I}} = \{x \,|\, \forall y : (x, y) \in R^{\mathcal{I}} \setminus R'^{\mathcal{I}} \text{ implies } y \in C^{\mathcal{I}}\}$. We have $(\geqslant nR)^{\mathcal{I}} = (\geqslant nR.\top)^{\mathcal{I}}$ and $(\leqslant nR)^{\mathcal{I}} = (\leqslant nR.\top)^{\mathcal{I}}$. After this transformation, the new language is called $\mathcal{SHIN}_{\setminus}$, because of the number restrictions which are not qualified anymore and a new role-set difference operator, $\forall(R \setminus R')$. Considering $\dot{\neg} C$ as the standard negation normal form (NNF) of $\neg C$, we define a recursive function unQ which rewrites $\mathcal{SHIQ}$ concept descriptions and assertions into $\mathcal{SHIN}^{\setminus}$.

Definition 14 (unq)**.** *Let R' be a new role in RN with $\mathcal{R} := \mathcal{R} \cup \{R' \sqsubseteq R\}$ for each transformation. unQ rewrites the axioms as follows:*

$unQ(C) := C$ *if* $C \in N_C$

$unQ(\neg C) := \neg C$ *if* $C \in N_C$*, otherwise* $unQ(\dot{\neg} C)$

$unQ(\forall R.C) := \forall R.unQ(C)$

$unQ(C \sqcap D) := unQ(C) \sqcap unQ(D)$

$unQ(C \sqcup D) := unQ(C) \sqcup unQ(D)$

$unQ(\geqslant nR.C) := \geqslant nR' \sqcap \forall R'.unQ(C)$

$unQ(\leqslant nR.C) := \leqslant nR' \sqcap \forall(R \setminus R').unQ(\dot{\neg} C)$

$unQ(a : C) := a : unq(C)$

$unQ((a, b) : R) := (a, b) : R$

$unQ(a \neq b) := a \neq b$

Note that this rewriting generates a unique new role for each QNR. For instance, if there exists an axiom $D \sqsubseteq \geqslant nR.C \sqcap \leqslant mR.C \sqcap \geqslant kR.D$ in TBox $\mathcal{T}$ w.r.t. role hierarchy $\mathcal{R}$ after the application of unQ, we have $D \sqsubseteq \geqslant nR_1 \sqcap \forall R_1.C \sqcap \leqslant mR_2 \sqcap \forall R \setminus R_2.\neg C \sqcap \geqslant kR_3 \sqcap \forall R_3.D$ and $\{R_1 \sqsubseteq R, R_2 \sqsubseteq R, R_3 \sqsubseteq R\} \subseteq \mathcal{R}$.

According to [OK99], $\geqslant nR.C$ can be converted to $\exists R' : (R' \sqsubseteq R) \in \mathcal{R} \wedge \geqslant nR' \wedge \forall R'.C$ and $\leqslant nR.C$ to $\exists R' : (R' \sqsubseteq R) \in \mathcal{R} \wedge \leqslant nR' \wedge \forall R'.C \wedge \forall(R \setminus R').\neg C$. Then the negated form of $\neg(\geqslant nR.C)$ according to its converted form is $\forall R' \sqsubseteq R :\leqslant (n-1)R' \sqcup \exists R'.\neg C$ which is not expressible in $\mathcal{SHIQ}$. In order to prevent such transformation, the unQ must be applied to the NNF of its input. The axiom of the form $\exists R.C$ is represented as a cardinality restriction of the form $\geqslant 1R.C$.

As shown, the translation of $\leqslant nR.C$ includes one more $\forall$ expression than the one we defined in unq function. [FH10b] proved that this issue dose not violates the correctness of our algorithm.

$\mathcal{SHIN}^{\setminus}$ is not closed under negation due to the fact that $unQ(\leqslant nR.C)$ itself creates a negation which must be in NNF before further application of unQ. In order to avoid the whole negating problem for the concept descriptions generated by $unQ(\leqslant nR.C)$ and $unQ(\geqslant nR.C)$, the calculus makes sure that the application of unQ starts from the inner-most part of an axiom, therefore such concept descriptions will never be negated.

Similar to [HST00b], the algorithm propagates TBox axioms through all the individuals, by defining $C_{\mathcal{T}} := \bigsqcap_{C_i \sqsubseteq D_i \in \mathcal{T}} unQ(\dot{\neg} C_i \sqcup D_i)$ and U as a new transitive role in the role hierarchy $\mathcal{R}$. A TBox $\mathcal{T}$ is consistent w.r.t $\mathcal{R}$ *iff* the concept $C_{\mathcal{T}} \sqcap \forall U.C_{\mathcal{T}}$ is satisfiable w.r.t $\mathcal{R}_U := \mathcal{R} \cup \{R \sqsubseteq U | R \in N_R\}$. Hence, all the axioms in TBox $\mathcal{T}$ will be applied to all the individuals.

Since the algorithm deals with inverse roles, and between two individuals there may exist a pair of directed edges, it is necessary to distinguish between the level of nodes in the constructed tree. To this end, a unique precedence is assigned to each individual.

Definition 15 (Precedence)**.** *Due to the existence of inverse roles for each pair of individuals x, y, $R \in \mathcal{L}(\langle x, y\rangle)$ imposes $Inv(R) \in \mathcal{L}(\langle y, x\rangle)$. A global counter PR keeps the number of nodes, and each time a new node x is created, the value of PR is increased by one and $PR_x = PR$. Hence, all nodes are ranked with a PR. A successor of x with the lowest PR is called parent (or parent successor) of x and others are called its children. Accordingly, a node x has lower precedence than a node y if x has lower rank compare to y. Also, each node has a unique rank and no two nodes have the same rank.*

For reasons of simplicity, to each role in the existing number restrictions, a set will be assigned which contains a specific type of sub-role called *proper sub-role*.

Definition 16 (Proper Sub-Role)**.** *A proper sub-role $\Re(R)$ for role R is defined as $\Re(R) = \{R_i \,|\, (R \in N_R \cup Inv(R)) \wedge R_i \sqsubseteq R\}$. This makes specializing the edges between nodes possible. Therefore, in our algorithm, when $\alpha(v)$ is assigned to $\mathcal{L}(\langle x, y\rangle)$ a new proper sub-role S_i will be created for each role $S \in \alpha(v)$, where $\Re(S) = \Re(S) \cup \{S_i\}$, and S_i will be assigned to the edge label. A role in $\Re(S)$ cannot have any proper sub-role. Only roles that occur in number restrictions can have proper sub-roles. Since these proper sub-roles do not appear in the logical label of nodes, they do not violate the correctness of our algorithm.*

A blocked node is defined according to the pair-wise blocking technique presented in section 3.1.1.

Definition 17 (Blocked Node)**.** *Node y is blocked by node x, also called witness, if $\mathcal{L}(x) = \mathcal{L}(y)$ and for their successors y', x', $\mathcal{L}(y') = \mathcal{L}(x')$ and $\mathcal{L}(x, x') = \mathcal{L}(y, y')$. Moreover, unreachable nodes which were discarded from the forest (due to the application of the $reset$-Rule or $reset_{IBE}$-Rule) are called blocked. In order to detect blocked nodes, each role that is a proper sub-role of R is considered equivalent to R.*

4.1.3 Atomic Decomposition

[OK99] proposed a so-called atomic decomposition for reasoning about sets, which was later on applied to DLs for reasoning about role fillers. Using atomic decomposition, all possible disjoint subsets of a role filler are considered such that $|A + B| = |A| + |B|$ for two subsets A, B and $|\cdot|$ denotes the cardinality of a set. For instance, if there is an ABox $\mathcal{A}$, such that $\mathcal{A} = \{a : (\leqslant 3hasComputer \sqcap \geqslant 5hasPC \sqcap \leqslant 4hasMac)\}$, using atomic decomposition, we get the following disjoint partitions:

c	=	(computer, not Mac, not PC)
p	=	(PC, not Mac, not computer)
m	=	(Mac, not computer, not PC)
cp	=	(computer, PC, not Mac)
cm	=	(computer, Mac, not PC)
mp	=	(Mac, PC, not computer)
cmp	=	(computer, Mac, PC)

Due to these seven disjoint subsets, the implied cardinalities for the individual a can be translated to the following inequations:

$$|c| + |cp| + |cm| + |cmp| \leqslant 3$$

$$|p| + |cp| + |mp| + |cmp| \geqslant 5$$

$$|m| + |cm| + |mp| + |cmp| \leqslant 4$$

4.2 Arithmetic Reasoning for $\mathcal{SHIN}\backslash$

The hybrid algorithm benefits from an algebraic reasoning component. The atomic decomposition defined in Def. 4.1.3 is the first phase in this method. In contrast to [Fad11], the hybrid algorithm proposed in this work uses a local atomic decomposition. Due to the global nature of *nominals* ($\mathcal{O}$), [Fad11] provides a global atomic decomposition as a pre-processing step, while our hybrid algorithm provides a local atomic decomposition for each individual.

In the following, we introduce several definitions and show an example to explain arithmetic reasoning more precisely.

Definition 18 (ξ)**.** *Let $\mathcal{V}^R = \{v \in \mathcal{V} \,|\, R \in \alpha(v)\}$ be the set of all variables which are related to role R. The function ξ maps number restrictions to inequations such that $\xi(R, \bowtie, n) := (\sum_{v_i \in \mathcal{V}^R} v_i) \bowtie n$. (Assume that $\alpha(v)$ is a set of roles which are mapped to v, in Def. 20 we will define it more precisely.)*

Definition 19 (Distinct partitions)**.** *R_x is defined as the set of related roles for x such that $R_x = \{S \,|\, \{\xi(S, \geqslant, n), \xi(S, \leqslant, m)\} \cap \mathcal{L}_E(x) \neq \emptyset\}$. A partitioning $\mathcal{P}_x$ is defined as $\mathcal{P}_x = \bigcup_{P \subseteq R_x} \{P\} \setminus \{\emptyset\}$. For a partition $P_x \in \mathcal{P}_x$, $P_x^{\mathcal{I}} = (\bigcap_{S \in P_x} Fil^{\mathcal{I}}(x, S)) \setminus (\bigcup_{S \in (R_x \setminus P_x)} Fil^{\mathcal{I}}(x, S))$ with $Fil^{\mathcal{I}}(x, S) = \{y^{\mathcal{I}} \,|\, y \in Fil(x, S)\}$. The definition clearly demonstrates that the fillers of x related to the roles of partition P_x are not the fillers of the roles in $R_x \setminus P$ (other partitions). Therefore, by definition the fillers of x associated with the partitions in $\mathcal{P}_x$ are mutually disjoint w.r.t. the interpretation $\mathcal{I}$.*

The arithmetic solution is defined using the function $\sigma : \mathcal{V} \rightarrow \mathbb{N}$ mapping each variable in $\mathcal{V}$ to a non-negative integer. Let $\mathcal{V}_x$ be the set of all variables assigned to node x such that $\mathcal{V}_x = \{v_i \in \mathcal{V} \,|\, v_i \text{ occurs in } \mathcal{L}_E(x)\}$, a solution Ω for node x is $\Omega(x) := \{\sigma(x) = n \,|\, n \in \mathbb{N}, v \in \mathcal{V}_x\}$. The lp-solver uses an objective function to determine whether to minimize the solution or maximize it. We minimize the solution in order to keep the size of the forest small.

Definition 20 (Variables)**.** *Assuming a set of variables $\mathcal{V}$ and a mapping $\alpha : \mathcal{V} \leftrightarrow \mathcal{P}_x$ for a node x, a unique variable $v \in \mathcal{V}$ is associated to a partition $P_x \in \mathcal{P}x$ such that $\alpha(v) = P_x$.*

Definition 21 (Node Cardinality)**.** *The cardinality associated with proxy nodes is defined by the mapping $card : V \rightarrow \mathbb{N}$.*

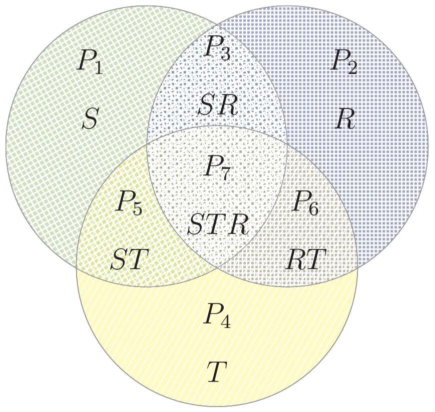
Due to the nature of inverse roles , a node can be a successor of two nodes (see Fig. 9 in which y is successor of x and z). As a result, a filler of a set of roles in partition P_x, can be a filler of a set of roles in partition P_y. Therefore, partitions and variables are defined locally for a corresponding node.

Since the hybrid algorithm required to have all numerical restrictions as a set of inequations, following functions are defined to transform the NRs to the inequations and/or modify the variable.

Function ξ transforms the NRs to inequations and put them to the $\mathcal{L}_E$ of a corresponding node. It is used in the $\geqslant$-Rule and $\leqslant$-Rule as shown in Fig. 13 and will be explained in Sec. 4.3. Function ζ and ς also add new inequations to the $\mathcal{L}_E$ of a node and modify the variables. The ζ and ς are respectively used in IBE-Rule and $reser_{IBE}$-Rule as shown in Fig. 13. The examples in Sec. 4.5.1 and Sec. 4.5.2 are explaining the task of these two functions in more details.

Definition 22 (ζ)**.** *For a set of roles RO and $k \in \mathbb{N}$, the function $\zeta(RO, k)$ maps number restrictions to inequations via the function ξ for each $R_j \in RO$. $\zeta(RO, k)$ would return a set of inequations such that $\zeta(RO, k) = \{\xi(R_j, \geqslant, k) \mid R_j \in RO\} \cup \{\xi(R_j, \leqslant, k) \mid R_j \in RO\}$. For $v \in \mathcal{V}^{R_j}$ if $R_j \in \alpha(v) \wedge \alpha(v) \nsubseteq RO$ then $v \leqslant 0$ is returned too.*

Definition 23 (ς)**.** *For a set of roles RO and $k \in \mathbb{N}$, the function $\varsigma(RO, k)$ maps number restrictions to inequations via the function ξ for each $R_j \in RO$. $\varsigma(RO, k)$ would return a set of inequations such that $\varsigma(RO, k) = \{\xi(R_j, \geqslant, k) \mid R_j \in RO\} \cup \{\xi(R_j, \leqslant, k) \mid R_j \in RO\}$. For $v \in \mathcal{V}^{R_j}$ if $RO = \alpha(v)$ then $v = k$ is returned.*



$$\alpha(v_{001}) = p_1, \alpha(v_{010}) = p_2, \alpha(v_{100}) = p_4,$$
$$\alpha(v_{011}) = p_3, \alpha(v_{110}) = p_6, \alpha(v_{101}) = p_5,$$
$$\alpha(v_{111}) = p_7$$

$$v_{001} + v_{011} + v_{101} + v_{111} \leqslant 2$$
$$v_{010} + v_{011} + v_{110} + v_{111} \geqslant 1$$
$$v_{100} + v_{110} + v_{101} + v_{111} \leqslant 2$$
$$v_{100} + v_{110} + v_{101} + v_{111} \geqslant 3$$

Figure 12: Atomic Decomposition

The following example depicts the process of finding an arithmetic solution in more details. Let $\mathcal{L}(x) = \{\leqslant 2S, \geqslant 1R, \leqslant 2T, \geqslant 3T\}$ be the label of node x. Applying the atomic decomposition for related roles $R_x = \{S, R, T\}$ results in seven disjoint partitions such that:

$$\mathcal{P}_x = \{p_1,\, p_2,\, p_3,\, p_4,\, p_5,\, p_6,\, p_7\} \text{ where}$$
$$p_1 = \{S\},\, p_2 = \{R\},\, p_4 = \{T\},\, p_3 = \{S, R\},$$
$$p_5 = \{S, T\},\, p_6 = \{R, T\},\, p_7 = \{R, S, T\}$$

as demonstrated in Fig. 12. In order to simplify the mapping between variables and partitions, each digit of the binary coding of a variable index represents a specific role in R_x. Therefore, in this example the first bit from right represents S, the second R, and the last T. Since $|R_x| = 3$, the number of variables in $\mathcal{V}_x$ becomes $2^3 - 1$. The mapping of variables and the resulting inequations in $\mathcal{L}_E(x)$ are shown in Fig. 12.

4.3 Completion Rules for $\mathcal{SHIN}_\backslash$

The ABox completion rules for $\mathcal{SHIN}_\backslash$ are shown in Fig. 13, listed in decreasing priority from top to bottom. Rules in the same cell have the same priority. Rules with lower priorities cannot be applied to a node x, which is not blocked, if there is any rule with a higher priority still applicable to it.

Therefore the rules are applied according to the following priorities:

1. $reset$-Rule, $reset_{IBE}$-Rule
2. $merge$-Rule
3. $\sqcap$-Rule, $\sqcup$-Rule, $\forall$-Rule, $\forall_{\backslash}$-Rule, $\forall_{+}$-Rule, ch-Rule
4. $\geqslant$-Rule, $\leqslant$-Rule, IBE-Rule
5. BE-Rule
6. RE-Rule
7. fil-Rule

Among the completion rules in Fig. 13, $\sqcap$-Rule, $\sqcup$-Rule, $\forall$-Rule, $\forall_{+}$-Rule are the same as in the standard tableau. The $merge$-Rule, $\forall_{\backslash}$-Rule, ch-Rule, $\geqslant$-Rule, $\leqslant$-Rule are similar to the one in [FH10b].

$\geqslant$-Rule and $\leqslant$-Rule: all number restrictions from $\mathcal{L}(x)$ are collected via these two rules. ξ translates them to inequations according to the proper atomic decompositions and adds them to $\mathcal{L}_E(x)$.

In contrast to [FH10b] these two rules are not the only source of generating inequations and consequently extending $\mathcal{L}_E(x)$. In addition, $\mathcal{L}_E(x)$ is modified by the IBE-Rule, $reset$-Rule and $reset_{IBE}$-Rule. we will explain these rules in the following.

Note that the idea of using atomic decomposition, inequation generation, and finally an arithmetic solution is to create role fillers for a corresponding node according to all the stabilized information gathered from lower priority rules. The issue in our case is that a node may have a role filler, i.e., an element of a partition, prior to computing the atomic decomposition and generating a solution. Therefore, this partition should be taken into

$reset$-Rule	**if** $\{(\leq nR), (\geq nR)\} \cap \mathcal{L}(x) \neq \emptyset$ and $\forall v \in V_x : R \notin \alpha(v)$
	then set $\mathcal{L}_E(x) := \emptyset$ and
	for every successor y of x set $\mathcal{L}(\langle x, y\rangle) := \emptyset$ and,
	if y in not parent of x set $\mathcal{L}(\langle y, x\rangle) := \emptyset$

$reset_{IBE}$-Rule	**if** $Inv(R) \in \mathcal{L}(\langle y, x\rangle)$ but $R \notin \mathcal{L}(\langle x, y\rangle)$
	then set $\mathcal{L}_E(x) := \mathcal{L}_E(x) \cup \{\zeta(\mathcal{L}(\langle x, y\rangle), card(y))\}$ and,
	for every successor y of x set $\mathcal{L}(\langle x, y\rangle) := \emptyset$ and,
	if y is not parent of x set $\mathcal{L}(\langle y, x\rangle) := \emptyset$

$merge$-Rule	**if** there exist root nodes z_a, z_b, z_c for $a, b, c \in I_{\mathcal{A}}$ such that $R' \sqsubseteq_* R_{ab}, R' \in \mathcal{L}(\langle z_a, z_c\rangle)$ **then** merge the node z_b, z_c and their labels and, replace every occurrence of z_b in the completion graph by z_c
$\sqcap$-Rule	**if** $(C_1 \sqcap C_2) \in \mathcal{L}(x)$ and $\{C_1, C_2\} \nsubseteq \mathcal{L}(x)$ **then** set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C_1, C_2\}$
$\sqcup$-Rule	**if** $(C_1 \sqcup C_2) \in \mathcal{L}(x)$ and $\{C_1, C_2\} \cap \mathcal{L}(x) = \emptyset$ **then** set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{X\}$ for some $X \in \{C_1, C_2\}$
$\forall$-Rule	**if** $\forall S.C \in \mathcal{L}(x)$ and there is an S-neighbour y of x with $C \notin \mathcal{L}(y)$ **then** set $\mathcal{L}(y) = \mathcal{L}(y) \cup \{C\}$
$\forall_{\setminus}$-Rule	**if** $\forall R \setminus S.C \in \mathcal{L}(x)$ and there is an R-neighbour y of x with $C \notin \mathcal{L}(y)$ and y is not S-neighbour of x **then** set $\mathcal{L}(y) = \mathcal{L}(y) \cup \{C\}$
$\forall_+$-Rule	**if** $\forall S.C \in \mathcal{L}(x)$ and there is some R with $Trans(R)$ and $R \sqsubseteq_* S$ and there is R-neighbour y of x with $\forall R.C \notin \mathcal{L}(y)$ **then** set $\mathcal{L}(y) = \mathcal{L}(y) \cup \{\forall R.C\}$
ch-Rule	**if** there occurs v in $\mathcal{L}_E(x)$ with $\{v \geq 1, v \leq 0\} \cap \mathcal{L}_E(x) = \emptyset$ **then** set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{X\}$ for some $X \in \{v \geq 1, v \leq 0\}$
$\geq$-Rule	**if** $(\geq nR) \in \mathcal{L}(x)$ and $\xi(R, \geq, n) \notin \mathcal{L}_E(x)$ **then** set $\mathcal{L}_E(x) = \mathcal{L}_E(x) \cup \{\xi(R, \geq, n)\}$
$\leq$-Rule	**if** $(\leq nR) \in \mathcal{L}(x)$ and $\xi(R, \leq, n) \notin \mathcal{L}_E(x)$ **then** set $\mathcal{L}_E(x) = \mathcal{L}_E(x) \cup \{\xi(R, \leq, n)\}$
IBE-Rule	**if** $\mathcal{L}_I(x) \neq \emptyset$ and $\{\varsigma(\mathcal{L}_I(x), 1)\} \cap \mathcal{L}_E(x) = \emptyset$ **then** set $\mathcal{L}_E(x) = \mathcal{L}_E(x) \cup \{\varsigma(\mathcal{L}_I(x), 1)\}$
BE-Rule	**if** there exists v occurring in $\mathcal{L}_E(x)$ such that $\sigma(v) = 1$, $R \in \alpha(v)$, $R \in \mathcal{L}_I(x)$ and y is parent of x with $\mathcal{L}(\langle x, y\rangle) = \emptyset$ **then** set $\mathcal{L}(\langle x, y\rangle) := \alpha(v)$
RE-Rule	**if** there exists v occurring in $\mathcal{L}_E(z_a)$ such that $\sigma(v) = 1$, z_a, z_b root nodes, $R_{ab} \in \alpha(v)$ with $x, b \in I_A$ and $\mathcal{L}(\langle z_a, z_b\rangle) = \emptyset$ **then** set $\mathcal{L}(\langle z_a, z_b\rangle) := \alpha(v), \mathcal{L}_I(\langle z_b) :=$ Inv$(\alpha(v))$
fil-Rule	**if** there exists v occurring in $\mathcal{L}_E(x)$ such that $\sigma(v) = n$ with $n > 0$, x is not blocked and $\neg\exists y : \mathcal{L}(\langle x, y\rangle) = \alpha(v)$ **then** create a new node y and set $\mathcal{L}(\langle x, y\rangle) := \alpha(v)$, $\mathcal{L}_I(y) :=$ Inv$(\alpha(v))$ and $card(y) = n$

Figure 13: The complete tableaux expansion rules for $\mathcal{SHIQ}$-ABox

account when generating solutions. As a result, in order to consider this edge, we translate it into a set of inequations, through the IBE-***Rule***, and ensure that the possible solutions include this back edge.

IBE***-Rule***: this rule considers the implied back edge in $\mathcal{L}_E$ and determines potential variables that can represent the IBE through elimination of the non-related variables. Assume that for a node x a successor y has been created with $\mathcal{L}(\langle x, y\rangle)$. This implies a back edge for y with a label $Inv(\mathcal{L}(\langle x, y\rangle))$. This back edge is considered as a set of NRs of the form $\geqslant 1R_i, \leqslant 1R_i$ where $R_i \in Inv(\mathcal{L}(\langle x, y\rangle))$. The IBE-Rule transforms the implied back edge into a set of inequations in $\mathcal{L}_E(x)$ of the form $(\sum_{v_j \in \mathcal{V}^{R_i}} v_j) \geqslant 1$ and $(\sum_{v_j \in \mathcal{V}^{R_i}} v_j) \leqslant 1$ using function ς. Since the inequations representing the back edge are restricted to the value one, only one common variable v_k in these inequations will be $\sigma(v_k) = 1$. In addition, ς ensures that the potential variables for IBE include all the roles in $\mathcal{L}_I(y)$ (see Def. 23). See the example in section 4.5.1 for considering IBE which explains the function ς (Fig. 16 and Fig. 17).

$reset_{IBE}$***-Rule***: this rule extends $\mathcal{L}_E$ as follows. If for a node y and its parent node x, $\mathcal{L}(x, y) \neq Inv(\mathcal{L}(y, x))$, then it implies that a new role should be considered in $\mathcal{L}_E(x)$ of the parent node due to the restrictions of its child. Therefore, the $reset_{IBE}$-Rule fires for x where ζ extends $\mathcal{L}_E(x)$ to consider $Inv(\mathcal{L}(y, x))$ and ensures that the specific variable representing this implied forward edge (IFE) is included in the solution (see Def. 22). See the examples in section 4.5.2 for considering IFE which explains the function ζ (Fig. 18 and Fig. 19).

Function ζ and ς are defined, as shown respectively in Def. 22 and Def. 23, in order to transform number restrictions to inequations for a set of roles, and modify the boundary of variables in order to impose specific answer. The difference between these two function is due to the approach of initializing variables. ζ assigns a specific value greater than zero to the variable representing IFE and make it as an answer as shown in Fig. 18 and Fig. 19, while ς sets some unrelated variables to zero in order to limit the answer to the certain variables, potential variables for IBE as shown in Fig. 16 and Fig. 17.

$reset$-***Rule***: if a new number restriction with a new role R is added to the logical label of a node x, all its children that are not root nodes with lower precedence than x are discarded from the tree and marked as blocked which makes them unreachable. In addition $\mathcal{L}_E(x) = \emptyset$ and for a successor y, $\mathcal{L}(\langle x, y\rangle) = \emptyset$ and if it is not the parent of x, then $\mathcal{L}(\langle y, x\rangle) = \emptyset$.

BE-***Rule***: this rule fills the label of the back edge to its parent due to the solution of the inequations solver. If a variable v in a solution exists such that $\sigma(v) = 1$ and roles in $\mathcal{L}_I(y)$ are in $\alpha(v)$, then the variable represents the back edge and the BE-Rule fires. Checking for one of the roles of $\mathcal{L}_I$ is sufficient due to the fact than only one variable represents the back edge in an answer. The IBE-Rule ensures this issue by function ζ before generating a solution.

We adjust the edges between a pair of nodes to satisfy the nature of the inverse roles between them. The interactions between IBE-Rule, BE-Rule, and $reset_{IBE}$-Rule maintain this characteristic. Fig. 14 demonstrates the interaction of these rules to find a proper model.

RE-***Rule***: this rule sets the edge between two root nodes. For nodes $a, b \in I_{\mathcal{A}}$, $(a, b) : R$ is considered as $a : \geqslant 1R_{ab}, \leqslant 1R_{ab}$ and $b : \geqslant 1Inv(R_{ab}), \leqslant 1Inv(R_{ab})$, therefore, in a solution for node a a variable with the value of 1, $\sigma(v) = 1$, $R_{ab} \in \alpha(v)$ represents this edge. The RE-Rule fires and fills the edge label, $\mathcal{L}(\langle a, b\rangle) = \alpha(v)$.

$merge$-***Rule***: the $merge$-Rule merges root nodes. Assume three root nodes $a, b, c \in I_{\mathcal{A}}$ where b, c are respectively R-successor and S-successor of a. These assertions will be translated such that we have $a : \geqslant 1R_{ab}, \leqslant 1R_{ab}, \geqslant 1S_{ac}, \leqslant 1S_{ac}$. If there exists a variable v in an arithmetic solution of node a with $R_{ab}, S_{ac} \in \alpha(V)$, it means that c and b need to be merged. The $merge$-Rule merges b and c and w.l.o.g replaces every occurrence of b with c and all outgoing/incoming edges of b become outgoing/incoming edges of c.

ch***-Rule***: this rule is necessary to ensure the completeness of the algorithm. The partitions provided by the atomic decomposition technique represent all the possible combinations of successors of a corresponding node. If a partition is logically not satisfiable, the corresponding variable should be set to zero. If it is indeed satisfiable, only the inequations' restrictions may influence the number of successors in this partition. Besides, the arithmetic reasoner does not have any information about the satisfiability of a concept representing the partitions. Therefore, in order to organize the search space with respect to semantic branching and to ensure completeness, the algorithm needs to distinguish between these two cases: $v \leqslant 0, v \geqslant 1$. The ch-Rule is similar to the choose-rule in the standard tableau in the sense that it considers two branches for each partition.

fil***-Rule***: fil-Rule has the lowest priority among the completion rules. This rule is the only one that generates new nodes. Since this rule generates new nodes based on the arithmetic solution that satisfies all the inequations, there is no need to merge the generated nodes later.

It is worth mentioning that the algorithm preserves the role hierarchies when initializing variables. If for a node x there exists a variable $v \in \mathcal{L}_E(x)$ where $R \in \alpha(v)$ and $S \notin \alpha(v)$ and $R \sqsubseteq_* S$, then $\sigma(v) = 0$. Therefore the variables that violates the role hierarchy are set to zero.

4.4 A Scenario for Application of Completion Rules

Fig. 14 demonstrates the process of adjusting the edges between a pair of individuals through the completion rules. Assume again $E \sqsubseteq C \sqcap \geqslant 2R \sqcap \forall R.(\geqslant 1R^-.C \sqcap \leqslant 1R^-.C)$. After pre-processing we have $E \sqsubseteq C \sqcap \geqslant 2R \sqcap \forall R.(\geqslant 1R'^- \sqcap \forall R'^-.C \sqcap \leqslant 1R''^- \sqcap \forall R^- \setminus R''^-.\neg C)$ with $\{R'^- \sqsubseteq R^-, R''^- \sqsubseteq R^-\} \subseteq \mathcal{R}$. The algorithm starts with $\mathcal{A} = \{x : E\}$. After all number restrictions have been collected, variables have been initialized, and the arithmetic solution has been generated for x, the fil-Rule creates y with $card(y) = 2$.

Fig. 14 shows the process when the back edge is created for y. The cardinality for x is 1 and it is 2 for each of the individuals y, z, and t. The $*$ denotes the application of additional rules, from bottom to top, prior to the application of the current rule. The steps of this process are explained in Fig .15.

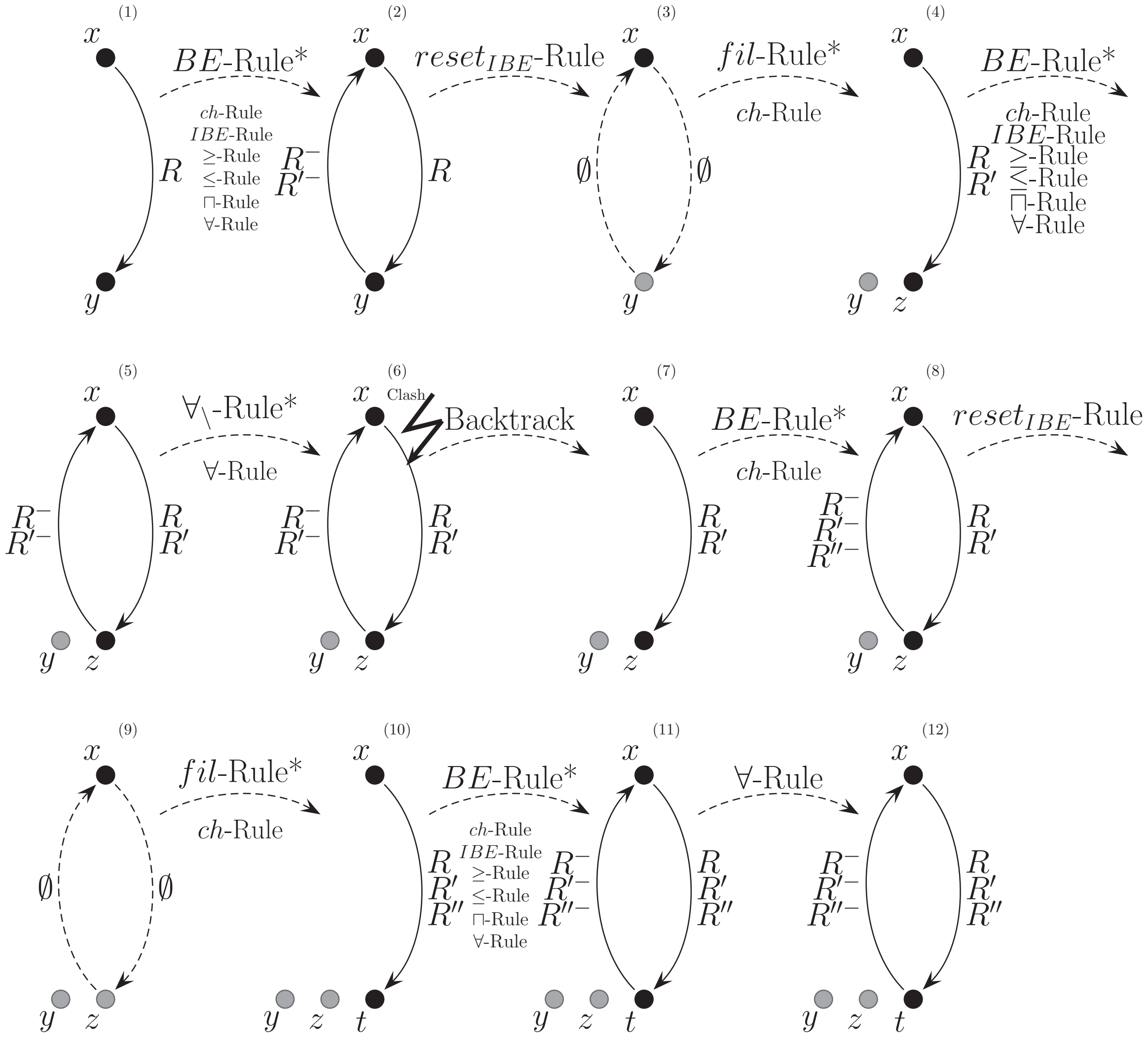


Figure 14: Interaction of completion rules for $C \sqcap \geqslant 2R \sqcap \forall R.(\geqslant 1R^{-'} \sqcap \forall R'^-.C \sqcap \leqslant 1R''^- \sqcap \forall R^- \setminus R''^-.\neg C)$

Note that in the hybrid algorithm the proper sub-roles are used in order to assign roles to the edge labels. This is necessary in order to specify the back edges and also to consider new roles in the $reset_{IBE}$-Rule.

$\longmapsto$(1)	The algorithm starts with $\mathcal{A} = \{x : E\}$. Then the $\sqcap$-Rule puts all the expressions in the label of x, the $\geqslant$-Rule and $\leqslant$-Rule collect the number restrictions from the $\mathcal{L}(x)$ and the arithmetic reasoning part finds a solution with $\Omega(x) = \{\sigma(v) = 2\}$ where $\alpha(v) = \{R\}$. Therefore the fil-Rule fires and generates an R-successor y with $\mathcal{L}(\langle x, y\rangle) = \{R\}$ and a cardinality of 2. Consequently $\mathcal{L}_I(\langle y, x\rangle) = \{R^-\}$.
(1)$\longrightarrow$(2)	The $\forall$-Rule adds expression $(\geqslant 1R'^- \sqcap \forall R'^-.C \sqcap \leqslant 1R''^- \sqcap \forall R^- \setminus R''^-.\neg C)$ to $\mathcal{L}(y)$ and the $\sqcap$-Rule adds all the disjuncts to $\mathcal{L}(y)$. Then, all the NRs will be collected via the $\geqslant$-Rule, the $\leqslant$-Rule, and the IBE-Rule and added to $\mathcal{L}_E(x)$. Afterwards, the ch-Rule fires to modify the variables and according to a generated answer from the arithmetic reasoner ($\Omega(x) = \{\sigma(v) = 1\}$ where $\alpha(v) = \{R^-, R'^-\}$), the BE-Rule sets $\mathcal{L}(\langle y, x\rangle) = \{R^-, R'^-\}$.
(2)$\longrightarrow$(3)	Since R'^- exists in $\mathcal{L}(\langle y, x\rangle)$ but Inv$(R'^-) = R'$ dose not exist in $\mathcal{L}(\langle x, y\rangle)$, the $reset_{IBE}$-Rule fires to consider R' in $\mathcal{L}_E(x)$. The node y will be discarded and the labels of the edges become empty.
(3)$\longrightarrow$(4)	The ch-Rule modifies the variables in $\mathcal{L}_E(x)$ and a new answer is generated for x. Then, according to the answer ($\Omega(x) = \{\sigma(v) = 2\}$ where $\alpha(v) = \{R, R'\}$), the fil-Rule creates a new R, R'-successor, z, with a cardinality of 2.
(4)$\longrightarrow$(5)	Similar to the (1)$\longrightarrow$(2), at last the BE-Rule fires and sets $\mathcal{L}(\langle y, x\rangle) = \{R^-, R'^-\}$ according to an answer based on $\mathcal{L}_E(y)$.
(5)$\longrightarrow$(6)	The $\forall_\setminus$ fires and adds $\neg C$ to the label of x. Since both $\neg C$ and C exist in $\mathcal{L}(x)$, a clash occurs.
(6)$\longrightarrow$(7)	Because of the clash the algorithm backtracks to select another choice for the answer regarding the cause of the clash.
(7)$\dashrightarrow$(11)	These steps are similar to the steps in (1)$\dashrightarrow$(5).
(11)$\longrightarrow$(12)	After application of the $\forall$-Rule no more rules are applicable and the algorithm terminates and returns *satisfiable* as the answer.

Figure 15: The explanation of steps for Fig. 14

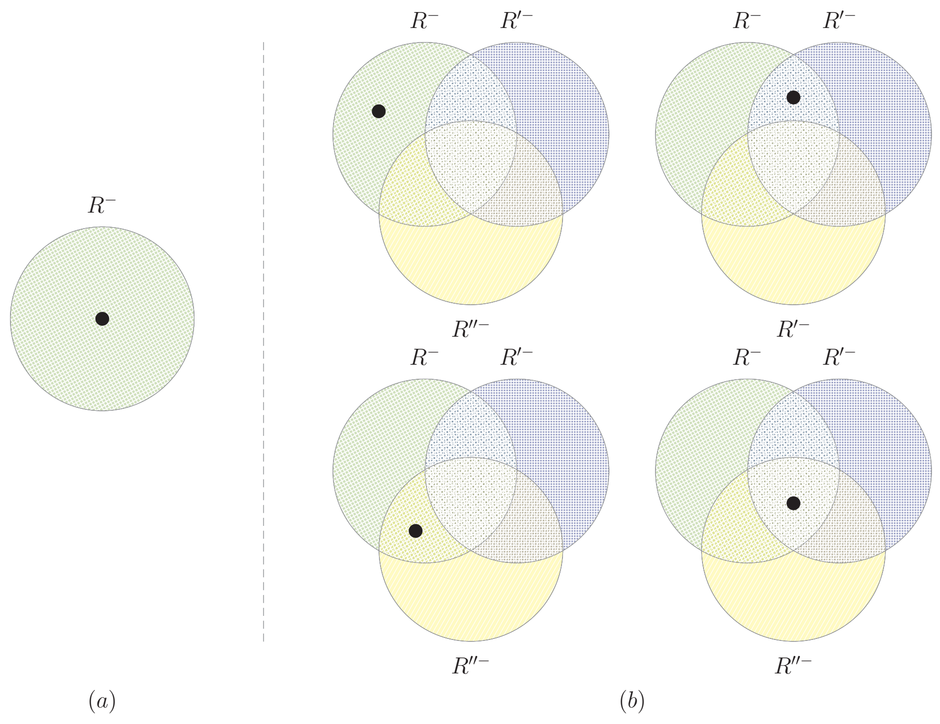


Figure 16: Considering IBE in the partitions.

4.5 Adjusting Partitions

The main idea of using algebraic reasoning is to capture all numerical restrictions for a node *at once*. By the phrase at once, we want to put an emphasis on the fact that the process of atomic decomposition and assigning variables are performed for each node only once, after all restrictions for a corresponding node are collected. The works presented in [FH10b] for DL $\mathcal{SHQ}$ is based on this argument. However, in the presence of inverse roles this argument no longer holds. There is always the possibility that a new QNR is added to the label of a node for which the arithmetic reasoning has been applied before and the related partitions have been build for it. In such a case the new QNR must be considered in partitions and variables. Therefore, the atomic decomposition is performed for the node again. Moreover since inverse roles are preserved as numerical restrictions, extra variable boundaries should be enforced to impose some specific answers to the arithmetic reasoner. In order to explain these cases more precisely some examples are demonstrated in the

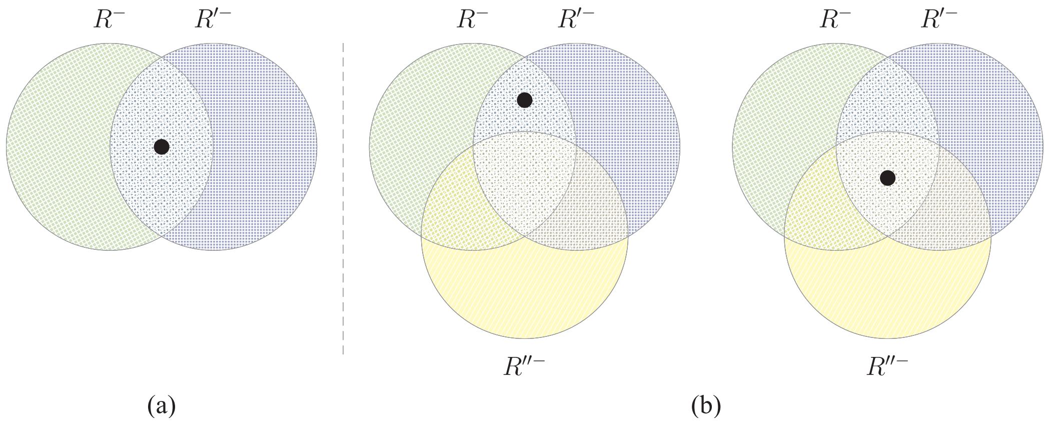


Figure 17: Considering IBE in the partitions.

following.

4.5.1 Considering Implied Back Edges

Similar to the example in Fig. 14, assume a concept expression $E \sqsubseteq C \sqcap \geqslant 2R \sqcap \forall R.(\geqslant 1R^-.C \sqcap \leqslant 1R^-.C)$. After pre-processing we have $E \sqsubseteq C \sqcap \geqslant 2R \sqcap \forall R.(\geqslant 1R'^- \sqcap \forall R'^-.C \sqcap \leqslant 1R''^- \sqcap \forall R^- \setminus R''^-.\neg C)$ with $\{R'^- \sqsubseteq R^-, R''^- \sqsubseteq R^-\} \subseteq \mathcal{R}$, Inv$(R'^-) = R'$, and Inv$(R''^-) = R''$. A model for this concept is generated based on an individual x with an R-successor y (see Fig. 14). This R-successor implies an IBE for y, which is the R^--successor x and is represented as a set of inequations of the form $\{\geqslant 1\,R^-, \leqslant 1R^-\}$, and as shown in Fig. 16(a) the only answer to satisfy this set is represented by a dot in R. By collecting all number restrictions for y a set of inequations would be $\{\geqslant 1\,R^-, \leqslant 1R^-, \geqslant 1R'^-, \leqslant 1R''^-\}$. Accordingly the three roles R^-, R'^-, R''^- should be considered in the atomic decomposition as demonstrated in Fig. 16(b).

In order to impose the IBE in answers for node y, the algorithm determines the potential partitions for the IBE (via the function ζ in the IBE-Rule). As shown in Fig. 16(b), there would be four potential partitions to be considered for the IBE. The only condition for a potential partition is that it must contain all the roles occurring in the IBE (see Def. 23 for the function ς). Note that since the values of number restrictions for IBE are 1, only one

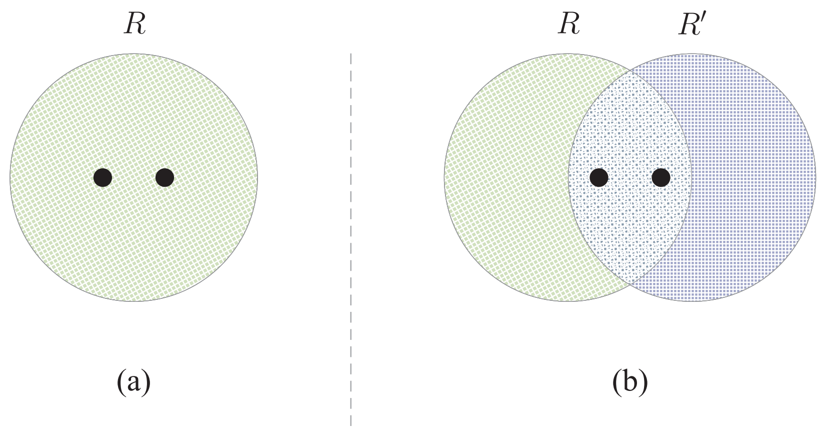


Figure 18: Considering IFE in the partitions.

partition could be selected as an answer for the IBE. This processing is performed by the IBE-Rule in the step between (1) and (2) in Fig. 14.

Now consider the case in which there are more than one role in an IBE (the application of the IBE-Rule between (4) and (5) or (7) and (8)). R^-, R'^- are the roles in the IBE for node y. Fig. 17(a) shows that the partition which is the intersection of R^-, R'^- represents the IBE which contains an answer. Now considering all QNRs, the potential partitions for IBE would be the two cases that are demonstrated in Fig. 17(b). Both potential IBE partitions contain all the roles occurring in the IBE. The whole process will be executed via the IBE-Rule, in which the function ς (see Def. 23) would generate the inequations and modify the variables.

4.5.2 Considering Implied Forward Edges

Assume that in previous examples shown in Fig. 16 and Fig. 17, a partition from a set of potential IBE partitions was chosen as an answer, which contains at least one role more than the ones occurring in the IBE of y, such as R^-, R'^-, see model (2) in Fig. 14. In such a case the extra role Inv$(R^{-'}) = R'$ should be considered for node x in an *implied forward edge* (IFE). Therefore, $\{\geqslant 2R', \leqslant 2R'\}$ will be added to the number restrictions for x via the $reset_{IBE}$-Rule and the function ζ (see Def. 22). Note that the values of number

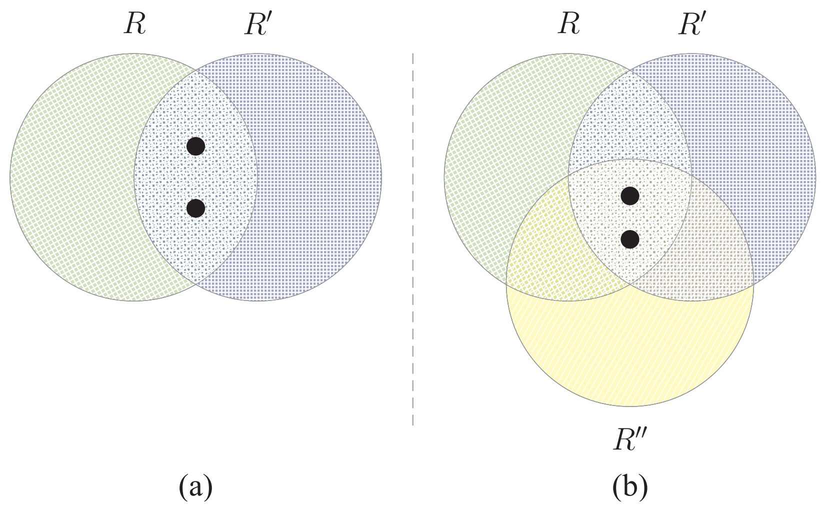


Figure 19: Considering IFE in the partitions.

restrictions are 2 since the cardinality of y was 2, and the point of adding such NRs is to recreate a proxy individual similar to y with the proper incoming and outgoing edges (see the steps from (2) to (4) in Fig. 14).

Fig. 18(a) demonstrates the partitioning before adding the new IFE with R' for node x (it shows the partitioning for node x in Fig. 14 graph (2)). Two dots in Fig. 18(a) represent the node proxy y with cardinality of two Fig. 14 graph (2). After having the IFE, the related partition which includes all the roles in IEF, $\{R,\ R'\}$, will be selected as shown in Fig. 18(b). It shows the partitions for node x in Fig. 14, graph (4). The $reset_{IBE}$-Rule performs this process via function ζ which modifies the variables to impose the IFE in an answer.

Fig. 19(a) shows the partitioning for node x in Fig. 14, graph (8) before the application of the $reset_{IBE}$-Rule and Fig. 19(b) demonstrates the partitioning for x after the execution of the function ζ through the $reset_{IBE}$-Rule and considering IFE as shown in Fig. 14, graph (10).

4.5.3 Propagation of new QNRs

Whenever new QNRs are propagated back to a node for which arithmetic reasoning has been applied before, the atomic decomposition will be run considering new QNRs and all the arithmetic reasoning will be applied again.

4.6 Correctness of the Hybrid Algorithm

In the following, we will show termination, soundness and completeness of the presented hybrid calculus for $\mathcal{SHIQ}$ ABox consistency.

Lemma 1. *For each $\mathcal{SHIQ}$-ABox and a role hierarchy $\mathcal{R}$ the hybrid algorithm terminates.*

Proof. Let $n = |clos(A)|$ and m be the number of different number restrictions after the pre-processing step. Then, the length of a concept expression in a label of a node is at most m and the maximum number of roles which are included in the atomic decomposition is denoted as m. The following facts result in the termination of the algorithm:

1. The $merge$-Rule is the only rule in this algorithm which removes a node (specifically root nodes) from the forest. Considering the finite number of root nodes, which is equal to the number of individuals in the ABox, and the fact that the algorithm never generates root nodes, one can conclude that the $merge$-Rule cannot lead to any loops of generating and deleting a particular root node. The maximum number of times that it can be fired for a node is equal to the number of the root nodes.

2. The fil-Rule is the only rule in charge of creating nodes except for the root nodes. Let V_x be the set of variables assigned to a node x, $\mathcal{V}_x = \{v \in \mathcal{V} \mid v \text{ occurs in } \mathcal{L}_E(x)\}$. For a node x, the fil-Rule generates at most $|\mathcal{V}_x| = k$ successors y_i, $1 \leq i \leq k$, based on the solution of the lp-solver. Since the maximum number of roles is m then there would be at-most $2^m - 1$ possible combinations for variables. Consequently, the out degree of the forest is bounded by 2^m, that is $|\mathcal{V}_x| \leq 2^m$.

3. The $reset$-Rule, $reset_{IBE}$-Rule, and $merge$-Rule are the only rules that modify $\mathcal{L}(\langle x, y\rangle)$ and set it to the empty set. Since adding a new number restriction invokes the $reset$-Rule, the maximum number of times the $reset$-Rule can fire for a node is bounded to the number of all number restrictions after pre-processing, m. The $reset_{IBE}$-Rule fires whenever a new role occurs in $\mathcal{L}(y, x)$ such that its inverse is not included in $\mathcal{L}(x, y)$. In other words, the nature of the inverse roles has been violated. The number of variables in $\mathcal{L}_E(x)$ represents the number of different possible successors for a node x according to the number restrictions. Each of these successors has a back edge to node x. One or more roles may occur in the labels of these back edges $\mathcal{L}(y, x)$ while their inverse roles have not occurred yet in $\mathcal{L}(x, y)$, therefore, the $reset_{IBE}$-Rule for x is invoked. These roles only occur due to the number restrictions in $\mathcal{L}(y_i)$. Since m is the number of all number restrictions after pre-processing and $|\mathcal{V}_x| = 2^m - 1 = k$, then the number of times the $reset_{IBE}$-Rule can be fired for x is bounded by $2^k * (m - 1)$. This holds due to the fact that in the worst case only one role occurs in $\mathcal{L}(x, y)$ and the $(m - 1)$ remaining roles invoke $rese_{IBE}$-Rule $m - 1$ times.

   The fil-Rule, BE-Rule and RE-Rule fire to fill the edge labels due to the existence of (possibly new) solutions. The number of times these three rules fire for a node x depends on the number of possible answers for x and the number of times the $reset_{IBE}$-Rule and $reset$-Rule fire for x. Since the execution of these two roles results in the application of the fil-Rule and/or BE-Rule and/or RE-Rule, the number of times they fire together for x is similarly limited to $2^k * m$. Together with the fact that the fil-Rule fires a finite number of the times, one can conclude that the number of times these nodes are generated and reset is finite.

4. Nodes can be labeled with at most $|clos(A)| = n$ logical labels where edges can be labeled with at most the number of all roles after pre-processing, m. Therefore there are 2^{2mn} possible label combinations for two nodes and an edge. Considering the

pair-wise blocking condition, a pair of nodes and an edge with the same label cannot be repeated in a path p from the root. In such a case the successor of the second pair is blocked and is not expanded any more, limiting the length of a path from a root to 2^{2mn}.

5. lp-solver always terminates for a finite set of inequations.

$\square$

Lemma 2. *For a set of inequations in $\mathcal{L}_E(x)$ the arithmetic reasoner generates a solution, if there exists any, that satisfies all the inequations.*

The arithmetic reasoner (lp-solver) uses the Simplex method which is a method to solve problems in linear programming. For more details of this method, we refer the interested reader to [CLRS01].

Lemma 3. *The pre-processing step of the Hybrid algorithm preserves the semantics of ABox assertions such as $(a, b) : R$ and $a \neq b$.*

Proof. The Hybrid algorithm transforms all ABox assertions of the form $(a, b) : R$ to $a :\leqslant 1R_{ab} \sqcap \geqslant 1R_{ab}$ with $b : \top$ and $R_{ab} \sqsubseteq R$. Therefore, the algorithm collects all number restrictions through the $\leqslant$-Rule, $\geqslant$-Rule and IBE-Rule ,including $\leqslant 1R_{ab} \sqcap \geqslant 1R_{ab}$, and generates related inequations for the corresponding node and passes them to the lp-solver. According to Lemma 2, the correct answer will be generated for some $v \in \mathcal{V}_a$ with $R_{ab} \in \alpha(v)$, $\sigma(v) = 1$. Consequently, the RE-Rule fires and satisfies $(a, b) : R$, hence R_{ab} is added to $\mathcal{L}(\langle a, b \rangle)$. Since $R_{ab} \sqsubseteq R$ and the role hierarchy is considered in the variable initialization, $(a, b) : R$ holds. Moreover, it is obvious that $\sigma(v) = 1$ with $R_{ab} \in \alpha(v)$ and $\leqslant 1R_{ab}$ is satisfied because for all other variables $v' \neq v$ with $R_{ab} \in \alpha(v')$, we have $\sigma(v') = 0$.

The only way that the assertion of the form $a \neq b$ may be violated is the application of the $merge$-Rule. Assume that $a \neq b$ is violated and $a^{\mathcal{I}} = b^{\mathcal{I}}$ and the $merge$-Rule had been applied. Then, there are $a, b, c \in V$ such that w.l.o.g. $R_{ca}, R_{cb} \subseteq \mathcal{L}(\langle c, b\rangle)$ which means $(c, a) : R, (c, b) : R \in \mathcal{A}$. In this case in the process of the variable initialization, a variable $v \in \mathcal{L}_E(c)$ with $R_{ca}, R_{cb} \in \alpha(v)$ is assigned to zero. After pre-processing and generating the proper role hierarchy $R_{ca}, R_{cb} \subseteq \mathcal{L}(\langle c, b\rangle)$ holds. Together with the fact that based on atomic decomposition all possible combinations are generated, one can conclude that there exists a variable v which represents a merge of a and b. Since $v \leq 0 \in \mathcal{L}_E(c)$ the arithmetic reasoner can never find a solution that allows the merging of a and b. $\square$

In order to simplify the proof of soundness and completeness of our algorithm we define a tableau for a $\mathcal{SHIQ}$-ABox $\mathcal{A}$, which is an abstraction of a model of $\mathcal{A}$ but is still similar to completion graphs.

Let $\mathbf{I}_{\mathcal{A}}$ be the set of all individuals in ABox $\mathcal{A}$ and $\mathbf{R}_{\mathcal{A}}$ be the set of all role names occurring in $\mathcal{A}$ and the role hierarchy $\mathcal{R}$ together with their inverses. We define T as follow:

Definition 24 (Tableau)**.** *$T = (\mathbf{S}, \mathcal{L}, \mathcal{E}, \mathcal{J})$ is a tableau for $\mathcal{A}$ with respect to $\mathcal{R}$ iff*

- $\mathbf{S}$ *is a non-empty set of objects representing individuals,*
- $\mathcal{L} : \mathbf{S} \rightarrow 2^{clos(\mathcal{A})}$ *maps each object of* $\mathbf{S}$ *to a set of concepts,*
- $\mathcal{E} : \mathbf{R}_{\mathcal{A}} \rightarrow 2^{\mathbf{S}\times\mathbf{S}}$ *maps each role to a set of pairs of individuals,*
- $\mathcal{J} : \mathbf{I}_{\mathcal{A}} \rightarrow \mathbf{S}$ *maps each individuals occurring in* $\mathcal{A}$ *to objects in* $\mathbf{S}$.

In addition, for every $s, t \in \mathbf{S}$, $C, C_1, C_2 \in clos(\mathcal{A})$, $R, S \in \mathbf{R}_{\mathcal{A}}$, T satisfies the following properties:

P1. if $C \in \mathcal{L}(s)$ then $\neg C \notin \mathcal{L}(s)$.

P2. if $C_1 \sqcap C_2 \in \mathcal{L}(s)$ then $C_1 \in \mathcal{L}(s)$ and $C_2 \in \mathcal{L}(s)$.

P3. if $C_1 \sqcup C_2 \in \mathcal{L}(s)$ then $C_1 \in \mathcal{L}(s)$ or $C_2 \in \mathcal{L}(s)$.

P4. if $\forall R.C \in \mathcal{L}(s)$ and $\langle s, t\rangle \in \mathcal{E}(R)$ then $C \in \mathcal{L}(t)$.

P5. if $\forall R.C \in \mathcal{L}(s)$ and there exists some $S \sqsubseteq_* R$ with $S \in N_{RT}$ and $\langle s,t\rangle \in \mathcal{E}(S)$ then $\forall S.C \in \mathcal{L}(t)$.

P6. if $\forall R\backslash S.C \in \mathcal{L}(s)$ and $\langle s,t\rangle \in \mathcal{E}(R)$ but $\langle s,t\rangle \notin \mathcal{E}(S)$ then $C \in \mathcal{L}(t)$.

P7. $\langle s,t\rangle \in \mathcal{E}(R)$ iff $\langle t,s\rangle \in \mathcal{E}(Inv(R))$.

P8. if $\langle s,t\rangle \in \mathcal{E}(R)$ and $R \sqsubseteq S$ then $\langle s,t\rangle \in \mathcal{E}(S)$.

P9. if $\geqslant nR \in \mathcal{L}(s)$ then $|S^T(s)| \geqslant n$.

P10. if $\leqslant nR \in \mathcal{L}(s)$ then $|S^T(s)| \leqslant n$.

P11. if $a : C \in \mathcal{A}$, then $C \in \mathcal{L}(\mathcal{J}(a))$.

P12. if $(a,b) : R \in \mathcal{A}$, then $\langle \mathcal{J}(a), \mathcal{J}(b)\rangle \in \mathcal{E}(R)$.

P13. if $a \dot{\neq} b \in \mathcal{A}$, then $\mathcal{J}(a) \neq \mathcal{J}(b)$.

Since after pre-processing QNRs are transformed to unqualified ones we have $S^T(s) = \{t \in \mathbf{S} \mid \langle s,t\rangle \in \mathcal{E}(S)\}$.

Lemma 4 (Completeness)**.** *If expansion rules can be applied to a $\mathcal{SHIQ}$ ABox $\mathcal{A}$ and a role hierarchy $\mathcal{R}$ in such a way that results in a complete and clash-free completion forest, then $\mathcal{A}$ has a tableau w.r.t $\mathcal{R}$.*

Proof. Given that the DL $\mathcal{SHIQ}$ does not have the finite model property, to prove the soundness of the algorithm, the fact that a blocked tree may be a part of an infinite model must be considered. Inspired by [HST00b], we use a path construction to deal with infinite models and to facilitate our proof.

A tableau $T = (\mathbf{S}, \mathcal{L}, \mathcal{E}, \mathcal{J})$ from a complete and clash-free forest $\mathcal{F}$ is defined by mapping the nodes in $\mathcal{F}$ to elements in T as follows. An individual in $\mathbf{S}$ represents a path in $\mathcal{F}$ from a root node to a node which is not blocked via non-root nodes.

A path p is a series of pairs of nodes from $\mathcal{F}$ in the form of $[(x_0, x'_0), \ldots, (x_n, x'_n)]$, for which $\mathbf{Tail}(p) = x_n$ and $\mathbf{Tail}'(p) = x'_n$ hold. $[p|(x_{n+1}, x'_{n+1})]$ represents a path $[(x_0, x'_0), \ldots, (x_n, x'_n), (x_{n+1}, x'_{n+1})]$. Therefore, the set of all paths $\mathbf{Paths}(\mathcal{F})$ is defined as follows:

- For all root nodes x_0^i in $\mathcal{F}$, $[(x_0^i, x_0^i)] \in \mathbf{Paths}(\mathcal{F})$

- For a path $p \in \mathbf{Paths}(\mathcal{F})$ and a node $z \in \mathcal{F}$:

  1. if z is a successor of $\mathbf{Tail}(p)$ and z is neither blocked nor a root node, then $[p|(z,z)] \in \mathcal{F}$, or
  2. if there exist some $y \in \mathcal{F}$ where y is successor of $\mathbf{Tail}(p)$ and z blocks y, then $[p|(z,y)] \in \mathcal{F}$ (z is the witness of y).

Consequently, for every $p \in \mathbf{Paths}(\mathcal{F})$, the following facts hold:

- $\mathbf{Tail}(p)$ is not blocked.
- $\mathbf{Tail}(p) = \mathbf{Tail}'(p)$ iff $\mathbf{Tail}'(p)$ is not blocked.
- $\mathcal{L}(\mathbf{Tail}(p)) = \mathcal{L}(\mathbf{Tail}'(p))$.

According to the fact that a root node is neither blocked nor blocking any other nodes, it is always placed at the head of a path, in the first place. For a path $p = [q|(x,y)]$ we refer to q as the back-tail of p with $\mathbf{backTail}(p) = q$.

Assume a completion forest $\mathcal{F} = \{V, E, \mathcal{L}, \mathcal{L}_E, \mathcal{L}_I\}$, a tableau $T = (\mathbf{S}, \mathcal{L}^T, \mathcal{E}, \mathcal{J})$ is defined as follows:

$$\begin{aligned}
\mathcal{S} &= \{p_1, \ldots, p_m |\, \mathbf{backTail}(p_1) = \cdots = \mathbf{backTail}(p_m) = \mathbf{backTail}(p), \\
&\quad\; p \in \mathbf{Paths}, card(p) = m\} \\
\mathcal{L}(p_i) &= \mathcal{L}(\mathbf{Tail}(p)) \\
\mathcal{E}(R) &= \{\langle p, [p|(x,x')]\rangle \in \mathbf{S} \times \mathbf{S} \,|\, x' \text{ is an } \mathbf{R}\text{-successor of } \mathbf{Tail}(p)\} \cup \\
&\quad\; \{\langle [q|(x,x')], q\rangle \in \mathbf{S} \times \mathbf{S} \,|\, x' \text{ is an } \mathbf{Inv}(\mathbf{R})\text{-successor of } \mathbf{Tail}(q)\} \cup \\
&\quad\; \{\langle [(x,x)], [(y,y)]\rangle \in \mathbf{S} \times \mathbf{S} \,|\, x, y \text{ are root nodes and } y \text{ is an } (\mathbf{R})\text{-neighbour of } x\} \\
\mathcal{J}(a_i) &= \begin{cases} [(x_0^i, x_0^i)] \text{ if } x_0^i \text{ is a root node in } \mathcal{F} \text{ corresponding to an individual } a_i \in I_{\mathcal{A}} \\ \text{with } \mathcal{L}(x_0^i) \neq \emptyset \\ [(x_0^j, x_0^j)] \text{ if } x_0^j \text{ is a root node in } \mathcal{F} \text{ corresponding to an individual } a_j \in I_{\mathcal{A}} \\ \text{with } \mathcal{L}(x_0^i) = \emptyset, \mathcal{L}(x_0^j) \neq \emptyset \text{ and } x_0^i \doteq x_0^j \end{cases}
\end{aligned}$$

Now we prove that T satisfies all the properties of a tableau in Def. 24 as follows:

- Due to the fact that $\mathcal{F}$ is clash free, **P1** holds for T.

- Assume $C_1 \sqcap C_2 \in \mathcal{L}(p)$ then $C_1 \sqcap C_2 \in \mathcal{L}(\mathbf{Tail}(p))$ and for $\mathbf{Tail}(p) = x$, $C_1 \sqcap C_2 \in \mathcal{L}(x)$. In such a case, $\sqcap$-rule fires and adds C_1 and C_2 to the label of x. Since $\mathcal{F}$ is complete, **P2** holds for T and likewise **P3**.

- For **P4**, consider $p, q \in \mathbf{S}$, $\forall R.C \in \mathcal{L}(p)$ and $\langle p, q\rangle \in \mathcal{E}(R)$. According to the definition of $\mathcal{E}(R)$:

  - If $q = [p|(x, x')]$ then x' is R-successor of $\mathbf{Tail}(P)$ and since $\mathcal{F}$ is complete, the invocation of $\forall$-Rule yields $C \in \mathcal{L}(x') = \mathcal{L}(x) = \mathcal{L}(q)$.
  - If $p = [q|(x, x')]$ then x' is $\mathbf{Inv}(R)$-Successor of $\mathbf{Tail}(q)$ and since $\mathcal{F}$ is complete, $C \in \mathcal{L}(q) = \mathcal{L}(\mathbf{Tail}(p))$.
  - If $p = [(x, x)]$ and $q = [(y, y)]$ for two root nodes x and y, then y is $\mathbf{R}$-neighbour of x and since $\mathcal{F}$ is complete $C \in \mathcal{L}(y)$ and consequently $C \in \mathcal{L}(p)$

  For similar reasons the $\forall_{+}$-Rule ensures **P5** for T.

- Let $\forall R \setminus S.C \in \mathcal{L}(p)$, $\langle p, q\rangle \in \mathcal{E}(R)$ but $\langle p, q\rangle \notin \mathcal{E}(S)$. Assume that **P6** does not satisfy T, then $C \notin \mathcal{L}(q)$. If $q = [p|(x, x')]$ then x' is R-successor of $\mathbf{Tail}(\mathbf{P})$ and the $\forall_{\setminus}$-Rule is applicable, therefore $C \in \mathcal{L}(x') = \mathcal{L}(x) = \mathcal{L}(q)$ which is contradiction to our assumption. Likewise for other types of paths **P6** holds for T.

- **P7** holds for T because of the symmetric definition of the mapping $\mathcal{E}$.

- **P8** is taken into account when initializing the variables. If $R \in \alpha(v)$ and $S \notin \alpha(v)$ then $v \leq 0$. Moreover, the role hierarchy $\sqsubseteq_*$ is captured in Def. 9, hence the $\forall$-Rule, $\forall_{+}$-Rule and $\forall\setminus$-Rule deal with role hierarchy correctly.

- Since all numerical restrictions are handled by the arithmetic reasoning part, the hybrid algorithm treats both **P9** and **10** in the same way. The $\leqslant$-Rule and $\geqslant$-Rule collect all number restrictions from the logical label of a node after all other rules with higher priority have been applied. Note that the assertion $(x, z) : R$ for root nodes x, z is preserved in the form of $\leqslant 1R_{xz}, \geqslant 1R_{xz} \in \mathcal{L}(x)$ in the pre-processing phase. Therefore, the $\leqslant$-Rule and $\geqslant$-Rule capture them. In order to consider the implied back edge's roles, the IBE-Rule fires and imposes that edge to $\mathcal{L}_E(x)$. Similarly, the $reset_{IBE}$-Rule may extend $\mathcal{L}_E(x)$. Then proper partitions are created and the arithmetic reasoner creates a solution which satisfies all the inequations in $\mathcal{L}_E(x)$.

Assume that $\leqslant nR \in \mathcal{L}(p_i)$, therefore $\leqslant nR \in \mathcal{L}(\mathbf{Tail}(p_i))$ and since $\mathbf{Tail}(p_i) = x$ then $\leqslant nR \in \mathcal{L}(x)$ for a corresponding node in $\mathcal{F}$. Due to the atomic decomposition and consequently proper partitions and variables for x, the $\leqslant$-Rule adds $\Sigma v_i \leqslant n$ to $\mathcal{L}_E(x)$ with $R \in \alpha(v_i)$ and $1 \leqslant i \leqslant k$. The arithmetic reasoner generates a solution $\Omega_j(x)$ which satisfies all the inequations in $\mathcal{L}_E(x)$. Therefore, if $R \in \alpha(v_i^j)$ and $\sigma(v_i^j) \geqslant 1$ then we have $\sum_{1 \leqslant i \leqslant k} \sigma(v_i^j) \leqslant n$. $k \leqslant n$ holds because of the fact that the arithmetic reasoner satisfies the inequation $\Sigma v_i \leqslant n$, hence the number of variables that are greater than zero are less than or equal to n. This is due to the fact that in the worst case there would be n variables $v_i = 1$ with the sum equal to n. For each $\sigma(v_i^j) = n_i$, we distinguish three cases:

- Due to the precondition of the BE-Rule, if $\sigma(v_i^j) = 1$ and v_i^j represents the back edge to the parent of x, the BE-Rule fills the label of the back edge with $\alpha(v_i^j)$. This R-successor will be mapped to a path q of the form $p = [q|(x, x')]$.
- According to the precondition of the RE-Rule if $\sigma(v_i^j) = 1$ and v_i^j represents an edge to a root node (which is not parent of x due to the priority of the BE-Rule), then the RE-Rule fills the label of the edge with $\alpha(v_i^j)$ which is an R-successor of x and is mapped to a path q of the form $q = [(x_0^j, x_0^j)]$.
- Otherwise, the fil-Rule creates an R-successor y_i for x with cardinality n_i. Therefore y_i will be mapped to n_i paths of the form $q = [p_i|(y_i, y_i')]$ in tableau T. y_i is an R-successor of $\mathbf{Tail}(p_i)$ with $p_i \in \mathbf{S}$.

Assuming that **P10** is not satisfied, then $\leqslant n(R) \in \mathcal{L}(p_i)$ but $\sharp R^T(p_i) > n$. Hence $\sum \sigma(v_i^j) > n$ leads to the conclusion that the arithmetic reasoner does not satisfy the inequations and the generated solution violates the inequations and that is a contradiction to Lemma 2. Therefore **P10** holds for T and likewise **P9**.

- **P11**, **P12**, and **P13** hold because of Lemma 3. **P10** satisfies T, since the cardinality of each individual x_a where $a \in I_{\mathcal{A}}$ is set to one and $\mathcal{L}(x_a) = \{C \mid (a : C) \in C\}$

□

Lemma 5. *The application of the $reset$-Rule or $reset_{IBE}$-Rule for a node x will not result in a loss of information.*

Proof. Assume that a new number restriction $(\bowtie nR)$ is added to the logical label of node x for which a solution was generated before. This invokes the $reset$-Rule which sets $\mathcal{L}_E(x)$ to the empty set, but $\mathcal{L}(x)$ remains unchanged. Due to the type of number restriction, the $\leqslant$-Rule or $\geqslant$-Rule fires and collects number restrictions from $\mathcal{L}(x)$, and generates new partitions and variables. Hence the $reset$-Rule never eliminates anything from the logical label of a node. In other words, by applying the $reset$-Rule more constraints are added to $\mathcal{L}_E(x)$. Similarly the $reset_{IBE}$-Rule ensures that newly acquired number restrictions are taken into consideration by extending $\mathcal{L}_E(x)$. Note that the IBE-Rule may also extend $\mathcal{L}_E(x)$ with inequations representing a back edge of x. Since the forest extension is based on the logical labels of nodes and the resetting rules never modify them, they maintain the information in the forest. □

4.6.1 Completeness

For completeness we have to show that if there exists a model (tableau) for the given ABox $\mathcal{A}$, the completion rules can fire in such a way that they find this model. To this end, we first prove some lemmas.

Lemma 6. *If a non-negative integer solution $\Omega(x)$ based on the set of inequations $\mathcal{L}_E(x)$ with defined variable boundaries causes a logical clash, all other non-negative integer solutions also trigger the same clash.*

To be more clear, this lemma highlights the necessity and importance of the ch-Rule in assigning $\geqslant 0$ or $\leqslant 0$ to the variables in $\mathcal{L}_E$.

Proof. Let $\Omega(x)$ be a solution with $\{\sigma(v_1) = m_1, \sigma(v_2) = m_2, \ldots, \sigma(v_n) = m_n\}$ according to the set of inequations and their variable boundaries assigned by the ch-Rule. Therefore, we have $v_i \geqslant 1$ and all the other variables are zero. Having the same variable constraints the arithmetic reasoner may generate another solution $\Omega'(x)$ with $\{\sigma(v_1) = p_1, \sigma(v_2) = p_2, \ldots, \sigma(v_n) = p_n\}$. For both solutions we know that $v_i \geqslant 1$ but only the value of m_i may be different from p_i. Consider three cases as follows.

- If the BE-Rule is applicable, due to its preconditions $\sigma(v) = 1$ represents a back edge to the parent of x, p, which contains $R_{xp} \in \alpha(v)$. When initializing the variables, we consider that if a variable represents a back edge to the parent of x it must at least include the inverse of all roles in the label of the implied back edge to guarantee that the characteristic of inverse roles holds. Unless a unique variable in different solution represents this implied back edge and a clash occurred via this edge (according to the expression $\forall$ or $\forall_{\backslash}$) before, the same clash still occurs. Therefore the results dose not changes unless the ch-Rule selects another potential variables as an answer for implied back edge. Note that value of a variable representing the implied back edge in each solution is one, $\sigma(v) = 1$.

- If the RE-Rule is applicable, then z_x and its successor z_y are root nodes. Therefore no matter what the solution is, there would be a role R_{xy} in the solution that represents this edge and since $\sigma(v) = \sigma'(v) = 1$, the assigned value $\alpha(v)$ of $\mathcal{L}_E(x, y)$ depends on the partitioning and not on the particular solution $\Omega(x)$ or $\Omega'(x)$.

- If the fil-Rule is applicable, then a successor y of x is created and similar to the previous case the assigned value $\alpha(v)$ of $\mathcal{L}(x, y)$ depends on the partitioning and not on the particular $\Omega(x)$ or $\Omega'(x)$.

In all three cases, if a clash occurred due to a role in a partition, changing the value of a non-zero variable corresponding to that partition still leads to the same clash, since the role exists in the partition. Hence solutions with similar the same but different values for the variables do not change the result. In order to catch (possibly) different results, different partitions should be selected.

□

Lemma 7. *Let $\mathcal{A}$ be a $\mathcal{SHIQ}$-ABox and $\mathcal{R}$ a role hierarchy. If $\mathcal{A}$ has a tableau w.r.t $\mathcal{R}$, then completion rules can be applied to $\mathcal{A}$ in such a way that they yield a complete and clash-free completion forest.*

Proof. Let $T = \{\mathbf{S}, \mathcal{L}^T, \mathcal{E}, \mathcal{J}\}$ be a tableau for $\mathcal{A}$ then we claim that the algorithm creates a forest $\mathcal{F} = \{V, E, \mathcal{L}, \mathcal{L}_E, \mathcal{L}_I\}$ from which T can be derived. Now we show that for a node x in $\mathcal{F}$ the application of each completion rule results in a forest which still can be mapped to T. To accomplish this purpose, we consider each completion rule as follows:

- The $\sqcap$-Rule: if $C_1 \sqcap C_2 \in \mathcal{L}(x)$ then $C_1 \sqcap C_2 \in \mathcal{L}^T(x)$. In this condition the $\sqcap$-Rule applies and adds $C_1, C_2 \in \mathcal{L}(x)$ which implies $C_1, C_2 \in \mathcal{L}^T(x)$ due to **P2**. Other rules such as $\sqcup$-Rule, $\forall$-Rule, $\forall_+$-Rule and $\forall_\backslash$-Rule are similar to the $\sqcap$-Rule according to the fact that all of them are defined on the corresponding properties in T.

- The ch-Rule steers the arithmetic reasoner to obtain the proper solution by modifying the boundary of variables. Assume that $t_1, t_2, ..., t_n$ represent successors of s in T. We define a set $CL_{t_i} = \{R | R \in N_R, \langle x, t_i \rangle \in \mathcal{E}(R)\}$ for each successor t_i of s. Such a set represents an edge label between two individuals x and y in the forest $\mathcal{F}$ that are mapped to s and t_i. The successors may have the same CL. These successors are intuitively clustered in the groups of elements with the same CL. For a set of equal CLs we define a variable with $\alpha(v) = CL$. The ch-Rule must impose $v \geqslant 1$ so that we have T as the mapping of F. Due to the **P9** and **P10** of T the $\leqslant nR$ and $\geqslant mR$ holds for T. Therefore the arithmetic reasoner will find a non-negative integer solution based on these variables. Although a solution may vary for a set of variables with defined boundaries, according to Lemma 6 the forest will end up with the same result (clash or not) for any solution with the same variable constraints.

- The $merge$-Rule may only merge root nodes. Let b and c be the R-successors of a with $a, b, c \in I_{\mathcal{A}}$ and according to an at-most restriction $a : (\leqslant nR)$, b and c must to be merged. Due to the fact that T is a tableau $\leqslant nR \in \mathcal{L}^T(\mathcal{J}(a))$ imposes that $\mathcal{J}(b) = \mathcal{J}(c)$. Besides, for the root nodes x_a, x_b, x_c which represent respectively a, b, c, the merge-Rule merges x_b and x_c based on a solution for $\mathcal{L}_E(x_a)$. Therefore x_b and x_c from $\mathcal{F}$ are mapped to the same element in T which does not violate the structure of T.

- The fil-Rule generates successors for a node x based on a solution which the arithmetic reasoner created for $\mathcal{L}_E(x)$. Due to the priority of the fil-Rule, all number restrictions in $\mathcal{L}(x)$ are collected and also the effect of the back edge (inverse of the incoming edge for x) is taken into consideration by the function ξ in the IBE-Rule. Together with the fact that the ch-Rule modifies the boundary for each variable, one has to conclude that the solution satisfies T. Note that each node x with $card(x) = m > 1$ will be represents by m elements in **S**.

- The $\leqslant$-Rule, $\geqslant$-Rule only set $\mathcal{L}_E(x)$ according to the number restrictions included in $\mathcal{L}(x)$. Therefore these rules never violate the mapping from $\mathcal{F}$ to T.

- The IBE-Rule extends $\mathcal{L}_E(x)$ with the implied back edge from x to its parent. Since **P8** holds for T there is a back edge from s to its parent p which is the only successor of x with a lower precedence in **S**, that must be considered in $\mathcal{L}_E(x)$. Therefore, the IBE-Rule adds all the roles in the implied back edge as a set of inequations to $\mathcal{L}_E(x)$. The function ζ of this rule also changes the variable constraint to ensure the existence of back edge in the solution for x. See Fig. 13 for an explanation of the IBE-Rule and for more details on the function ζ see Def. 22 and the examples in Sec. 4.5.1. Since this rule only modifies $\mathcal{L}_E(x)$, it will not affect the mapping $\mathcal{F}$ to T.

- The $reset$-Rule and $reset_{IBE}$-Rule remove the label of outgoing edges for a node x and modify $\mathcal{L}_E(x)$. According to the fact that later the BE-Rule, RE-Rule, and fil-Rule with lower priority fill the label of outgoing edges, the mapping from $\mathcal{F}$ to T is not violated.

- The BE-Rule fills the label of a back edge for node x to its parent node p. Since the implied back edge is considered in $\mathcal{L}_E(x)$ and due to the fact that the IBE-Rule with a higher priority represents this IBE in the set of inequations, and the ch-Rule sets the variable constraints and the arithmetic reasoner satisfies all the ineqations with a non-negative solution, there will be a variable $v_{be} = 1$ that represents the back edge (there exists only one such edge for each node in $\mathcal{F}$). Therefore, the BE-Rule only sets the edge label to $\alpha(v_{be})$, which does not violate the mapping.

- The RE-Rule fills the edge label between two root nodes. Like in the previous case it satisfies the mapping from $\mathcal{F}$ to T.

Hence the $\mathcal{F}$ is a complete forest and also clash free due to the following facts:

- $\mathcal{F}$ cannot contain a node x with $\{C, \neg C\} \in \mathcal{L}(x)$ since $\mathcal{L}(x) = \mathcal{L}^T(s)$ and hence **P1** would be violated.

- $\mathcal{F}$ cannot contain a node x for which $\mathcal{L}_E(x)$ is unsolvable. In such a case there should exist a number restriction in the form of $(\leqslant nR)$ or $(\geqslant mR)$ in $\mathcal{L}(x)$ and consequently $\mathcal{L}^T(s)$ which cannot be satisfied. This violates **P9** and/or **P10** of a tableau.

$\square$

4.7 Summary

In this chapter the hybrid algorithm was introduced. Since the algorithm benefits from algebraic reasoning, the atomic decomposition technique and the arithmetic reasoning was explained. Then, the completion rules were introduced and termination, soundness and completeness of the algorithm were proved. The main difference with [FH10b], in the partitioning technique, was shown in Section 4.4 and a couple of examples are presented in order to explain the algorithm and partitioning technique.

Chapter 5

Practical Reasoning

In this section, we discuss the practical aspect of our work by explaining the complexity of our algorithm in section 5.1 and introducing optimization techniques that our reasoner benefits from in section 5.2. Then we will briefly explain our implemented reasoner in section 5.3.

5.1 Complexity

After a pre-processing step (see section 4.1) and transforming all QNRs to unqualified NRs, $(p+q)$ new number restrictions, and consequently new roles, are introduced in the form of $\{\leqslant n_1R_1, \leqslant n_2R_2, \ldots, \leqslant n_pR_p, \geqslant m_1S_1, \geqslant m_2S_2, \ldots, \geqslant m_qS_q\}$. In the worst case, considering inverse roles, this will result in $2(p+q) = k$ roles.

The search space of the hybrid algorithm depends on the number of variables in $\mathcal{L}_E$. Since there are k roles, the number of possible partitions and their related variables is bounded by $|V_x| = 2^k - 1$. The ch-Rule creates two branches for each variable: $v \geqslant 1$ or $v \leqslant 1$. Consequently, 2^k cases will be examined by the arithmetic reasoner and the complexity of the algorithm is a double exponential function of $2(p+q)$. Moreover, the Simplex method which is used in the hybrid algorithm is NP in the worst case. However, [Pap81] shows that if the number of variables is bounded, then the Simplex method is P in worst case.

The implemented lp solver minimizes the sum of the variables that occur in the inequations. In addition, we use several heuristics that can eliminate branches in the search space, therefore, avoiding unnecessary invocations of the ch-Rule. These heuristics dramatically improve the average complexity of the hybrid algorithm over the worst case of 2^{2^k}.

5.2 Optimization Techniques

In most cases, in order to utilize these theoretical algorithms in practice, optimization techniques are required. Due to the complexity of the algorithm, achieving a good performance may seem infeasible. However, experiments with early DL systems such as KRIS, and lately with state of the art DL systems have shown that applying suitable (even simple) optimization techniques could lead to a significant improvement in the empirical evaluation of a DL system. The optimization techniques dramatically decrease the size of the search space by pruning many branches. For instance, [HT00] uses axiom absorption for TBox services in order to improve the reasoner by preventing several unnecessary steps. In the following, the optimization techniques used in the hybrid algorithm are explained. The worst case complexity of the hybrid algebraic tableau-based satisfiability algorithm has been shown in Section.5.1 as being double exponential. The theoretical complexity result is not surprising because the satisiability problem of expressive DLs is usually inevitably at-least exponential. However, the algorithm might be considered theoretically inefficient because the complexity of the satisfiability algorithm (double exponential) came out greater than the complexity of the satisfiability problem itself (single exponential). Such a gap between the complexity of an algorithm and that of the problem might be due to the fact that the algorithm was designed in such a way to facilitate proofs of its soundness and completeness without much consideration to its worst case complexity or practical implementation.

One may find it discouraging to consider the practical implication of an algorithm with a high theoretical worst case complexity. However, a high worst-case complexity is not uncommon with practical DL systems. For example, the hyper-tableau satisfiability algorithms [MSH07], [MSH09] designed to handle general concept inclusion axioms (GCIs) more efficiently with the DLs $\mathcal{SHIQ}$ and $\mathcal{SHOIQ}$ share a double exponential worst case complexity, whereas satisfiability with $\mathcal{SHIQ}$ is EXPTIME-complete and that with $\mathcal{SHOIQ}$ is EXPTIME-complete. Moreover, the algebraic tableau reasoning algorithm [FH10b] and [Fad11] designed for the DL $\mathcal{SHQ}$ and $\mathcal{SHOQ}$ run in the worst case in double exponential time whereas satisfiability with $\mathcal{SHQ}$ and $\mathcal{SHOQ}$ is EXPTIME-complete. On the other hand, systems based on optimized implementations of these algorithms demonstrate significant performance improvement showing their practical impact in solving specialized problems. So far, no better way has been reported in solving QNRs other than through algebraic reasoning. Also, Hermit is the first reasoner able to classify ontologies which had previously been proven too complex for any available reasoner to handle.

Various optimization techniques were developed, targeting various reasoning services. *Absorption* [HT00] and *lazy unfolding* [BHN$^+$92] address classification and subsumption as TBox services. These optimization techniques facilitate subsumption testing by avoiding unnecessary steps in TBox reasoning.

The main feature of our hybrid algorithm is to address the performance issues regarding reasoning with QCRs independently from the reasoning service. In other words, by means of hybrid reasoning, we want to improve reasoning at the concept satisfiability level which definitely has an impact on TBox and ABox reasoning.

At the concept satisfiability level, the major source of inefficiency is due to the high nondeterminism in completion rules such as the $\sqcup$-Rule in Fig. 3 and the $choose$-Rule in Fig. 8 that create several branches in the search space. In order to remain complete, an algorithm needs to explore all of these branches. Optimization techniques mainly try to reduce the size of the search space by pruning the non-related branches. Also, heuristics can help the algorithm to guess which branches to explore first. When the algorithm is more informed, it would explore branches that have a lower probability of failing.

Considering a practical implementation for the hybrid algebraic algorithm, another source of inefficiency is partitioning which determines the number of variables for which the non-deterministic ch-Rule can fire. Although the hybrid algorithm is worst-case double-exponential and the large number of variables seems to be hopelessly inefficient, there are some heuristics and specialized optimization techniques which handle those inefficiency sources and make the calculus feasible to use. These techniques are discussed in the following.

5.2.1 Variable Initialization

In case of ordering non-deterministic expansions for a concept expression that includes a disjunction $(C \sqcup D \sqcup E \sqcup \dots)$, there are two possible technique to explore all possible models: *syntactic branching*, and *semantic branching*. In syntactic branching, one non-deterministically chooses an unexpanded disjunction $(C \sqcup D \sqcup E \sqcup \dots)$ in the label $\mathcal{L}(x)$ of a node x and add each of the disjuncts in $(C \sqcup D \sqcup E \sqcup \dots)$ to $\mathcal{L}(x)$. Therefore, due to the different cases, the algorithm might need to explore the various completion graphs before it terminates. Moreover, completion graphs corresponding to each of the disjuncts are not disjoint and exploring them non-deterministically can result in the recurrence of an unsatisfiable disjunct in more than one graph which is a major cause of inefficiency.

In contrast, in order to improve this inefficiency, semantic branching opens two branches, based on a single unexpanded disjunct C in $\mathcal{L}(x)$, one model C and the other $\neg C$ (added to $\mathcal{L}(x)$). The $choose$-Rule, provides this semantic branching for the QNRs in the tableau algorithm (see Fig. 8).

The hybrid algorithm explores the branches for QNRs, via the ch-Rule (see Fig. 13) for variables. Then there would be $v \geqslant 1$ in one branch and $v \leqslant 0$ in the other branch. In contrast to concept branching provided by the $choose$-Rule, in *variable branching* the existence of the variables that are less or equal zero can be ignored, since the arithmetic reasoner only considers the variables that are greater or equal to one.

The arithmetic reasoner starts with the default value of zero for all the variables and later sets some to be more than zero to satisfy inequations, according to given at-least restrictions. Therefore, by setting the default value to $v \leqslant 0$ for every variable, the algorithm does not need to invoke the ch-Rule $|V_x|$ times before starting to find a solution for the inequations.

Since the ch-Rule is invoked $2^{|V_x|}$ times in the worst case to check the variables for $v \geqslant 1$ or $v \leqslant 0$, default zero setting of variables prevents unnecessary invocations of the ch-Rule.

After setting all the variables to zero, the algorithm must decide to set some variables greater than zero in order to satisfy the at-least restrictions and find an arithmetic solution. The order in which the algorithm chooses these variables can help the arithmetic reasoner find the solution faster.

- *Don't care* variables: The boundary value of some of the variables can be determined from the beginning according to the occurrence of the variables in at-most or/and at-least restrictions. For example, if a variable occurs in an at-least restriction but not in an at-most restriction, then it does not have any arithmetic restrictions, other than logical restrictions which later on will be processed by the algorithm. Such variables are called *don't care* variables [FH10b] and according to the arithmetic limitations, any non-negative integer value can be assigned to these variables. Hence, the algorithm let them exist in all of the inequations unless they trigger a logical clash.

- *Satisfying* variables: The *satisfying* variables are defined as the set of variables that occur in an at-least restriction and are not *don't care* variables. Since they occur in an at-least restriction, assigning them to be greater or equal to one, the arithmetic reasoner is guided to a solution. Whenever a node that is created based on v causes a clash, by means of dependency-directed backtracking (explained in the following), it will be set to zero $v \leqslant 0$ and removed from the *satisfying* variables set. When the satisfying variables set becomes empty the algorithm concludes that the set of qualified number restrictions in $\mathcal{L}(x)$ is unsatisfiable.

- *Non-IBE* variables: In order to avoid unnecessary application of $reset_{IBE}$-Rule additional heuristics are applied. For a node x in $\mathcal{L}(x)$, if a role occurs in an at-least restrictions but not in any at-most restriction and it is not a sub-role of any role R in concept of the form $\forall R \setminus S.C \in \mathcal{L}(x)$, then it cannot be in a variable that represents a back edge for node x.

Note that the number of variables that can be decided to be greater than zero in an inequation is bounded by the number occurring in its corresponding numerical restriction. For example, in the inequation $v_1 + v_2 + \cdots + v_i \leqslant 5$, although we have 100 variables in the inequation, not more than five v_i can be greater or equal than one at the same time.

In addition, the $reset_{IBE}$-Rule specifically determines the value of a single variable,*IFE*[1], (see Def. 22 and section 4.5.2). Similarly, IBE-Rule modifies the value of a set of variables to zero, add them to the *non-IBE* variables, and enforces a set of variables to be the potential choices for the answer, named *IBE* variables (see Def. 23 and section 4.5.1). These rules also reduces the solution space by setting variables to zero.

It is worth to mention that one of the interesting characteristics of the variables is the way of encoding. As it was explained in section 4.2, the indices of the variables are encoded in binary format in order to simplify the process of retrieving the role names related to them. On the other hand, there is no need to assign any memory space for them unless they have a value greater than zero based on an arithmetic solution.

5.2.2 Dependency Directed Backtracking (DDB)

DDB techniques find sources of logical clashes and then consider the cause of the clash in setting the boundaries of variables in new solutions. This results in the algorithm pruning

[1]*Implied Forward Edge*

the branches which lead to the same clashes. [Hor02] introduces DDB and demonstrates how this method significantly improves the performance of the FaCT system to deal much more effectively with QNRs.

DDB adapted to the hybrid algorithm, similar to [FH10b], also remarks a considerable improvement in algebraic reasoning. In the hybrid algorithm, whenever a logical clash occurred for a successor y of x, one can conclude that the corresponding variable v_y for the partition that includes y must be zero. Therefore, all branches with $v_y \geqslant 1$ could be ignored. This *Simple DDB* can exponentially decrease the size of the search space by pruning half of the branches each time the algorithm detects a clash. For example, in the general case of $\mathcal{L}(x)$, by pruning all the branches where $v_y \geqslant 1$, $2^{|V_x|-1} = 2^{2^{p+q}-1}$ branches w.r.t. the ch-Rule which is half of the branches.

A more sophisticated DDB technique, *Complex DDB*, if the algorithm encounters a clash because of $\{A, \neg A\} \in \mathcal{L}(x)$, then the source for propagation of these two concepts could be the roles in $\forall R.A$ and/or $\forall (S \setminus T).\neg A$. In this case, the variables which contain all these roles ($R_i \in \alpha(v) \wedge (S_j \in \alpha(v) \wedge T_j \notin \alpha(v))$) are set to zero. It is shown in [FH10b] that with complex DDB, the number of branches will be reduced from $2^{|V_x|}$ to $2^{3/4|V_x|}$. These techniques eliminate many branches in the search space and consequently improves the average complexity of the algorithm.

Benefiting from all these heuristics and optimization techniques, the atomic decomposition will be a method to organize the search space and at the same time by means of numerical reasoning and proxy individuals remain independent from the value of numbers.

As it was shown in Lemma 6 in Chapter 4.6, for a set of inequations with the same boundaries for the variables all the solutions lead to the same situation, if a clash occurs due to a solution for a set of inequations, without changing the boundaries of variables all other answers would result in the same clash. In other words, the algorithm will create successors with the same logical labels but different cardinalities based on these different solutions. Since all the solutions minimize the sum of variables and satisfy all the numerical restrictions, they do not make any logical differences (as long as the set of zero-value variables is the same). Moreover, backtracking within arithmetic reasoning is not trivial due to the fact that the cause of an arithmetic clash cannot be easily traced back. In other words, the whole set of numerical restrictions together causes the clash. In the same sense as in a standard tableau algorithm, if all the possible merging arrangements end up with a clash, one can only conclude that the corresponding numerical restrictions are not satisfiable together.

5.3 Prototype Reasoner

This chapter represents the implemented prototype reasoner based on the hybrid algorithm presented in Chapter 4 and benefits from optimization techniques defined in Chapter 5.2. The architecture and the main modules of the reasoner will be explained.

In order to evaluate our hybrid algorithm, a prototype reasoner is implemented. We used the prototype reasoner from [FH10b] and extended it to handle inverse roles. The reasoner decides the satisfiability of $\mathcal{ALCHIQ}$ concepts. $\mathcal{ALCHIQ}$ is equivalent to DL $\mathcal{SHIQ}$ without transitive roles. Since, the interaction of transitive roles with qualified number restrictions results in undecidability [HST00a], only simple roles N_{RS} can have number restrictions except $\geqslant 1R.C$. To this end, it is not usual to consider an unrestricted combination of transitive roles and QNR. Therefore, in order to have a minimal work prototype, the reasoner is implemented without transitive roles.

The main focus of this implementation is to demonstrate how the hybrid reasoner exhibits stable behavior while other reasoners become dramatically slow as the number of qualified number restrictions and the complexity of combinations of them w.r.t. a role hierarchy and presence of inverse roles increases. The reasoner is implemented in Java using

Web Ontology Language (OWL) API[2] typically used to represent Semantic Web ontologies.

The system consists of two main modules, a logical module and an arithmetic module (lp-solver). The logical module, as the main part of the system, performs the completion rules and calls the arithmetic module when needed. After a call the arithmetic module gets all the number restrictions, applies atomic decomposition, generates variables according to the partitions, creates the proper set of inequations, applies the ch-Rule and finds the answer for the set. These interactions continue until a clash occurs and no more branches remain meaning the concept is not satisfiable, or it reaches the point that no more rules are applicable meaning the concept is satisfiable. Then the output of the reasoner would be either a complete and clash-free completion graph, if the input concept expression is satisfiable, or otherwise it returns *unsatisfiable*. Note that since the input of the reasoner is not an ABox, the algorithm constructs a completion graph rather than a completion forest.

As a pre-processing step, the logical module transforms the input concept expression according to the function unQ defined in Def. 14. It also provides the arithmetic module with a set of unqualified number restrictions (NRs). The arithmetic module either returns an arithmetic clash or a non-negative integer solution based on which the logical module generates the new successors for an individual, or modifies the edge to the existing successor (parent). In the following sections we describe applications of both modules in more details.

5.3.1 Logical Module

The logical module is the main part of the hybrid reasoner which includes the pre-processing component, clash strategy component, some auxiliary components. It also performs a set of expansion rules (except ch-Rule) and interacts with arithmetic modules according to the

[2]http://owlapi.sourceforge.net/

QNRs. The input ontology file (.owl[3] file) is processed with the pre-processing component and all qualified number restrictions are replaced with equisatisfiable unqualified ones which are also transformed to NNF. As explained in Chapter 4, the DL $\mathcal{SHIN}^{\setminus}$ is not closed under negation. Hence, the reasoner never negates a concept expression which is a direct or indirect output of the pre-processing component.

Reasoner Completion Rule	**Algorithm Completion Rules**
ResetRule()	$reset$-Rule
ResetIBERule()	$reset_{IBE}$-Rule
AndRule()	$\sqcap$-Rule
ForAllRule()	$\forall$-Rule
ForAllSubtractRule()	$\forall_{\setminus}$-Rule
Or-Rule()	$\sqcup$-Rule
BERule()	BE-Rule
BuildResultInSiblingRule()	backtracking
CollectAndInitiateArithmeticRule()	$\geqslant$-Rule, $\leqslant$-Rule, IBE-Rule, ch-Rule

Figure 20: The completion Rules and their represented functions.

The output of the pre-processing step is the input of the expansion rules component, based on which the completion graph is generated. The algorithm constructs a tree of *states*. Each state is a completion graph and represents the state of the constructed model. The application of each deterministic completion rule results in one new state and the application of non-deterministic rules leads to more than one state. For instance, if the $\forall$-Rule is applied on a node x in $state_1$ based on $\forall R.C \in \mathcal{L}(x)$, then $state_2$ includes the completion graph of $state_1$ with C added to the label $\mathcal{L}(y)$ where y is R-successor of x. Therefore, $state_2$ would be added as a child of $state_1$ in the tree. If the reasoner fires the $\sqcup$-Rule, based on $C_1 \sqcup \cdots \sqcup C_n \in \mathcal{L}(x)$, then n $state_i$ will be generated as children of $state_1$ and for each state, one of the conjuncts C_i will be added to $\mathcal{L}(x)$.

Each state, contains a unique completion graph with all related information of the corresponding graph such as individuals, cardinalities, labels. Recording all the states as a tree data structure, simplifies to find the source of creation of each state and consequently supports backtracking on states.

[3]http://www.w3schools.com/rdf/rdf-owl.asp

A set of completion rules represented in Fig. 20 are implemented in the reasoner to consider the concept satisfiability of a $\mathcal{SHIN}_\backslash$ TBox. The RE-Rule, and $merge$-Rule are not included since they are build in order to deal with the individuals in the ABox.

```
Expansion Rules Strategy

canApply(State $s$) {
if ($s$ contains an individual for which $this$-Rule is applicable) :
  return $true$
else
  return $false$
end if
}

apply(State $s$) {
$newState \longleftarrow$ Copy($s$)
$newState.parent \longleftarrow s$
for all $individual$ in $s$ such that $this$-Rule is applicable do :
  $newInd \longleftarrow$ apply $this$-Rule on $individual$
  replace $individual$ with $newInd$ in $newState$
end for
return $newState$
```

Figure 21: Expansion Rules Strategy

As it is demonstrated in Fig. 21, all individuals in the current state are checked to see if any of them has the required conditions (see the rules condition in Fig. 13), for the application of a rule. If a rule is applicable, it modifies a copy of a current state and puts it as a new state and child of the current state. An applicable rule is applied on all the individuals that satisfy the preconditions of the rule. The *Rule Expansion Strategy* ensures that the rules are applied according to their priorities, from top to bottom as listed in Fig. 20, to every state that is not closed. If no rule is applicable to a state, it will be closed. If all of the states are clashed and closed the input concept expression is unsatisfiable. Whenever a rule is applicable to a state, the logical module ensures that no rule with higher priority is applicable to it.

Using backtracking, the reasoner does not find all the solutions in one try. In other words, if a generated solution is end up in a clash, the algorithm backtracks to the choice point and generates a new solution considering the cause of the clash (modifying variables). There are two rules specified to handle these tasks (see Fig. 20): $CollectAndInitiateArithmeticRule()$ and $BuildResultInSiblingRule()$. The $CollectAndInitiateArithmeticRule()$, collects all number restrictions for each individuals in a $state_1$ and calls the arithmetic reasoner, with a set of inequations as input. Afterwards, it generates new successors based on the given solution (if there exists any) in the $state_2$.

$BuildResultInSiblingRule()$ is applicable to a clashed closed state, which is a clashed state that is not expandable w.r.t the OrRule. When the rule finds a clashed individual y in a $state_2$, it retrieves the parent of $state_2$, $state_1$, and calls the arithmetic module, sets the variable represented y to zero and gets a new solution.

All other rules shown in Fig. 20 are implemented similar to their related completion rule in the tableau (see Fig. 13).

Clash Strategy Component is implemented in order to detect the clashes. It triggers a clash for an individual x, either when for a concept name A, $\{A, \neg A\} \subseteq \mathcal{L}(x)$ or an arithmetic clash is detected in the arithmetic component (i.e. there exists no solution).

Blocking Strategy Component is called after each rule fired. This component searches the completion graph in the current state to check if blocking is needed. Also, it rechecks the blocked nodes and the witnesses and if the blocking condition is violated, it unblocks the nodes.

5.3.2 Arithmetic Reasoner

One of the major and also time consuming function of the hybrid reasoner, is the arithmetic module, which generates arithmetic solutions. Also, the ch-Rule is implemented in the arithmetic modules. The arithmetic module benefits from some heuristics, which improve the process of finding a non-negative solution.

For n NRs with roles $R_1, R_2, \ldots, R_n$, the arithmetic reasoner generates $2^n - 1$ variables. The encoding is similar to the one explained in Section.4.1.3 for atomic decomposition.

$freedom(v)$	**Category**
2	v must be zero according to logical reasons
0	v must be zero according to the ch-Rule choice
1	v must be greater or equal to 1 according to the ch-Rule choice
-1	v is a $don'tcare$ variable and can have any value

Figure 22: Categorizing variables.

The ith digit of the binary coding of m represents R_i where $R_i \in \alpha(v_m)$ For instance, with $n = 5$, we have $\alpha(v_5) = \{R_1, R_3\}$.

All the variables are classified according to the values they can take. In order to perform this classification, the notion of $freedom$ of variables is defined. For each variable v, $freedom(v)$ express the value assigned to the related classification. The classification of variables is demonstrated in Fig. 22.

```
Algorithm $findDontCare()$

for $i = 1$ to $n$ :
  if $NR[i]$ is an at-least restriction :
    for all $v_j$ $j = 1$ to $2^n - 1$ such that its $i$th digit in binary coding is equal to $1$ :
        if ($k$th digit of $j$) $= 1$ AND $NR[j]$ is not an at-most restriction :
          $freedom(v_j) = -1$
        end if
        if $freedom(v_j) \neq -1$ AND $freedom(v_j) \neq -2$ :
          add $v_j$ to $satisfyingVariablesList$
        end if
    end for
  end if
end for
```

Figure 23: Finding the *don't care* variables (see section 5.2)

Note that the assigned value to the freedom of an individuals may change handling unless it is equal to 2. The techniques discussed in section 5.2 regarding categorizing variables are all implemented in order to guide the arithmetic reasoner to find a solution. The heuristic mentioned in section 5.2 was also implemented according to the explanation.

```
Algorithm $fixRoleHierarchy()$

for $i = 1$ to $n$ :
  for $j = 1$ to $n$ :
    if $R_j \sqsubseteq R_i$ AND $i \neq j$ :
      for all $v$ such that $R_i$ related to $v$ AND $R_j$ not related to $v$ :
        $freedom(v) = 2$
      end for
    end if
  end for
end for
```

Figure 24: Applying role hierarchy.

The two function ζ (described in Def. 22) and ς (described in Def. 23) which respectively are used in the $reset_{IBE}$-Rule and IBE-Rule (Fig. 13), also modify the variables. They are implemented according to their definitions. Fig. 25(a) shows the algorithm for the function ς and Fig. 25(b) shows the algorithm for the function ζ.

The heuristic which is mentioned in section 5.2 is implemented as shown in Fig. 26. It prevents unnecessary applications of the $reset_IBE$-Rule.

After finalizing the satisfying variables list, the main function starts the application of the ch-Rule. The branching function starts letting the satisfying variables to have the freedom of 1 (i.e., being greater or equal 1). If there exist k variables in the satisfying variables list, in order to be complete, the algorithm must try all the 2^k cases regarding the freedom of the variables. Benefiting from backtracking, the algorithm returns to the logical module the first non-negative integer solution it finds. If the found solution logically fails, at least for one variable v, $freedom(v) = 1$ changes to $freedom(v) = 2$ which later will result in a different solution and the algorithm cannot compute the same solution again and falling in a cycle. If branching does not return any solution, the arithmetic module returns an arithmetic clash.

The integer programming or the inequation-solver component gets a set of linear inequations as input. The goal function is always set to minimize the sum of all the variables,

```
Algorithm $findIBE(roleSETRS)$

$findIBE(roleSETRS)\{$
for all $v_i$: $i = 1$ to the number of all variables:
  for all $R_j \in RS$: $j = 1$ to $|RO|$:
    if $R_j$ is related to $v_i$ :
      for all $R_k \in RS$: $k = 1$ to $|RO|$:
        if $R_k$ is not related to $v_i$ AND $j \neq k$ :
          $freedom(v_i) = 2$
          break
        end if
        add $v_i$ to $potentialIBEVList$
      end for
    end if
  end for
end for
$\}$
```

(a)

```
Algorithm $findIFE(ROLESRO, k)$

$findIFE(roleSETRS)\{$
for all $v_i$ : $i = 1$ to the number of all variables
  for all $R_j \in RO$: $j = 1$ to $|RO|$
    if $R_j$ is not related to $v_i$ AND $i \neq j$ :
      break
    end if
    add $v_i$ to $potentialIFEVList$
  end for
end for
for all $v_i$ : $i = 1$ to $|potentialIFEVList|$
  for all $R_j$: $j = 1$ to the number of all roles
    if $R_j$ is related to $v_i$ AND $R_j \notin RO$ :
      remove $v_i$ from $potentialIFEVList$
    end if
  end for
end for
$v \longleftarrow$ the first element of $potentialIFEVList$
$v = k$
$\}$
```

(b)

Figure 25: (a) Algorithm for finding potential IBE variables. (b) Algorithm for finding potential IFE variables.

while all of the variables are greater or equal zero. The set of constraints imposed by the freedom of the variables will also be part of the input in form of inequations. In other words, if $freedom(v) = 1$ for a variable, then $v \geqslant 1$ as a part of the input. Notice that in the cases where $freedom(v) = 0$ or $freedom(v) = 2$, v never appears in the set of input inequations. The integer programming component is composed of a linear programming algorithm according to the Simplex method presented in [CLRS01] and branch-and-bound to obtain integer solutions when the linear solution contains fractional values.

One of the major issue with this choice of language was the representation of float numbers. In other words, floating point numbers as a result of linear programming cannot be represented precisely. Therefore, sometimes rounding errors can result in a wrong solution. Especially when having a large number of variables, the sum of the errors may exceed 1 and may result in a wrong answer. This problem can be solved when representing fractional numbers by two integers: numerator and denominator. Unfortunately, these integers grow dramatically fast and use dynamic memory. Also, objects as float numbers can become

Heuristic 1

```
$findnotIBE()$ {
if $R$ occurs in at-least restriction AND
  $R$ does not occur in at-most restriction
  for all $forAllSubE_i$ in a label:
    if $isInFirstSubtract(R, mathitforAllSubE_i)$:
    for all $v_j$ in $IBEVariableList$
      if $R$ is related to $v_j$
          $freedom(v_l) = 2$
       end if
    end for
end if
}
```

```
$isInFirstSubtract(R, forAllSubE)$ {
$S \longleftarrow$ firstSubtract of $forAllSubE$
if $isSubproperyOf(R, S)$:
  return $true$
else
  return $false$
end if
}
```

Figure 26: A heuristic to prevent unnecessary application of $reset_{IBE}$. The firstSubtract refers to a role like S_i such that S_i is equal or sub-rule of S in an expression $\forall(S \setminus L)$.

very expensive in terms of time and memory. This is expected to be unlikely and basically never happened in any of the sample ontologies we used.

5.4 Summary

In this chapter we discussed the implemented prototype reasoner and also the optimization techniques provided for the reasoner were explained. In the next chapter we will evaluate our algorithm by comparing it to other reasoners.

Chapter 6

Evaluation

In this chapter we evaluate the practical performance of the hybrid reasoner. To this end, we compare the hybrid reasoner with state of the art reasoners such as, Pellet, FaCT++, and Hermit. In addition, the improvement by the optimization techniques represented in Chapter 5.2 is measured.

6.1 Choosing Benchmarks

The main goal of implementing the hybrid reasoner is to address the improvement on the effects of inverse roles and QNRs using algebraic reasoning in one hand, and show the effect of optimization techniques on the other hand. To this end, we focus our evaluation on concept expressions containing QNRs and inverse roles. For all roles in the test cases their inverse roles are also considered, therefore, the impact of inverse always is demonstrated. In order to meet these objectives, we designed various test cases that include pattern of problems encountered in the presence of QNRs and inverse roles and evaluated the hybrid reasoner with TBox consistency tests.

In order to study the behavior of the hybrid reasoner, we developed a set of synthetic benchmarks and identify the following parameters that may affect the complexity of reasoning:

- The value of the numbers, (m, n), occurring in QNRs, $(\geqslant mR.C, \leqslant nR.C)$, in presence of inverse roles.
- The back propagation of QNRs via inverse roles.
- The number of QNRs (at-least vs at-most).

The algebraic method has been adopted and studied previously in [HTM01, FH10b] to handle QNRs. Also [Fad11] benefits from the algebraic method to deal with QNRs and nominals and for both cases it shows a significant improvement on reasoning time. There are no well known ontologies addressing specifically the interaction between inverse roles and QNRs, therefore we synthesize test cases to show the weakness of state of the art reasoners and shows how our reasoner can overcome this weakness.

6.2 Evaluating Benchmarks

We will show the performance of the hybrid reasoner by a set of benchmarks. The test cases and results of evaluating the reasoner will be discussed. The demonstrated numbers are calculated considering the average of 10 executions of each test case. We run test cases on a system with AMD 3.4GHz quad core CPU and 16 GBs of DDR3 RAM.

6.2.1 Inverse roles and the value of numbers in QNRs

Consider $(\geqslant nhasComputer.PCand \leqslant mhasComputer.)$, we refer to n, m as the value of QNRs, which may vary according to the modeling domain. For example if the given domain is a person then n and m will be relatively small like 1 or 2, but considering modeling a big computer company or computer company, the value can easily increase to the value of (more or less than) several hundred thousand.

Assume the concept $Test_{Sat}$, which is the same as the example in Fig. 14, such that:

$$Test_{Sat} \sqsubseteq C \sqcap \geqslant 2kR \sqcap \forall R.(\geqslant kR^{-}.C \sqcap \leqslant kR^{-}.C) \text{ where } k = 2^{i}, i \in \{0, \ldots, 10\}$$

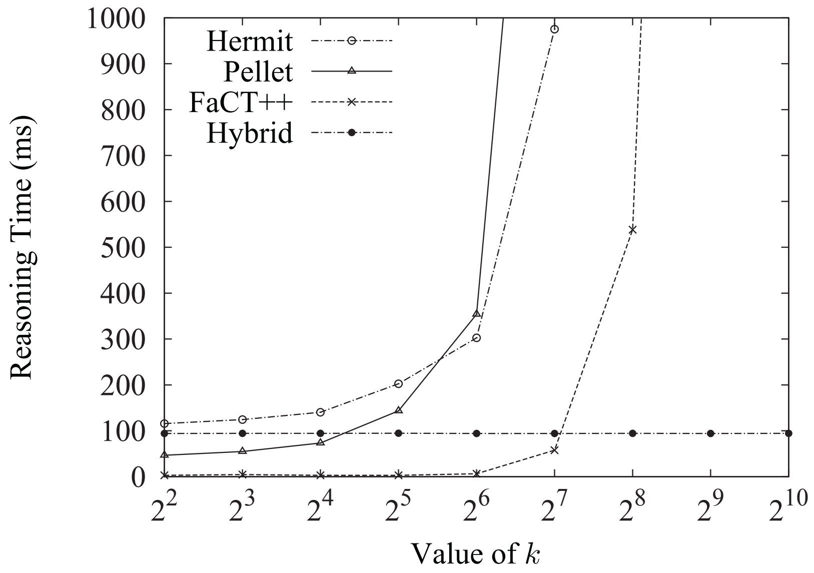


Figure 27: Exponential increase of k

Fig. 27 compares the reasoning time of our hybrid reasoner to Pellet, FaCT++, and Hermit. As demonstrated in the diagram, all three reasoners determine the satisfiability of the concept in less than $100(ms)$ before reaching $k = 2^4$. While the hybrid reasoner is able to maintain this reasoning time (constantly under a $100(ms)$), Pellet, Hermit, and FaCT++ exhibit exponential growth in the reasoning time for values higher than $k = 2^4$, $k = 2^6$, and $k = 2^7$ respectively. For example, for $k = 2^9$ Pellet's reasoning time is more than $18(mins)$ and for $k = 2^{10}$ it was not able to execute in a timely manner. This example demonstrates the independent behavior of our hybrid calculus in the presence of high number value of QNRs interacting with inverse roles.

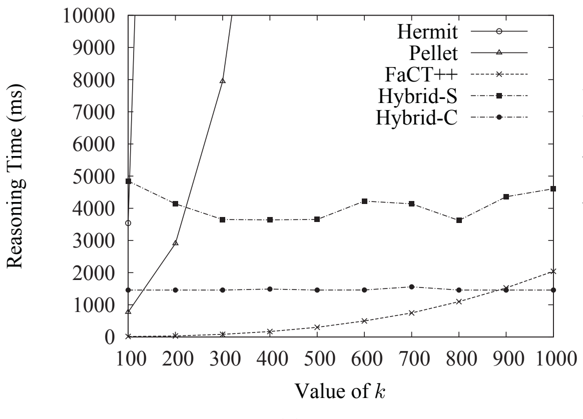
6.2.2 Back propagation of QNR

Assume the unsatisfiable concept $Test_{Unsat}$ such that:

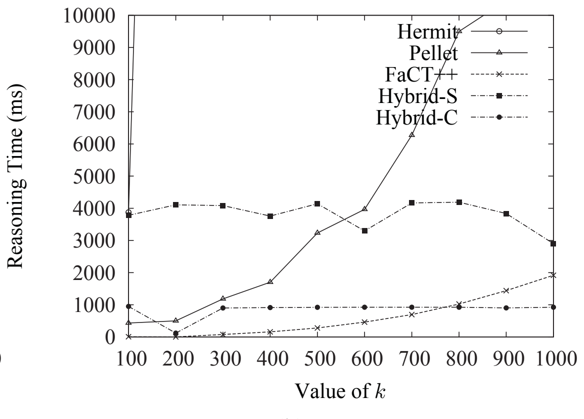
$$Test \sqsubseteq \geqslant 8S.A \sqcap \leqslant 9S.A \sqcap \forall S.(\geqslant kR.C \sqcap \leqslant 6T.D \sqcap \geqslant 5T.D) \sqcap$$

$$\forall S.\forall R.(\geqslant 2T.D \sqcap \leqslant 3T.D) \sqcap \forall S.\forall R.\forall T.\forall T^{-}.\forall R^{-}.P)$$

$$\text{with } \{R \sqsubseteq M\} \in \mathcal{R} \text{ and } k \in \{100, \ldots, 1000\}$$



(a)



(b)

Figure 28: (a) Linear increase of k for unsatisfiable concept expression. (b) Linear increase of k for satisfiable concept expression.

Fig. 28(a) demonstrates the reasoning time for satisfiability test of $Test$, where $P = (\leqslant (k-2)M.(C \sqcup D)$. In this example $\leqslant (k-2)M.(C \sqcup D)$ propagates back and makes the concept unsatisfiable. Fig. 28(b) shows the reasoning time for satisfiability test of $Test$ where $P = (\leqslant (k+1)M.(C \sqcup D)$. In this example $\leqslant (k+1)M.(C \sqcup D)$ propagates back and makes the concept satisfiable. As expected the Hybrid reasoner remains stable while the execution times of the other reasoners depend on the values in number restrictions.

As shown in Fig. 28(a) and Fig. 28(b), Hermit's behavior is the worst among all the reasoners. Hermit and Pellet show a rapid exponential growth in their reasoning times as a function of k. FaCT++ solves the problem in a more reasonable time, however, it demonstrates its dependency on the value of k as its runtime increases.

6.2.3 Backtracking

In order to analyze the impact of backtracking we focus on the hybrid algorithm in a separate graph. Fig. 29 shows the behavior of the hybrid algorithm using complex and simple dependency directed backtracking techniques regarding the previous test case, $Test_{Unsat}$. Complex backtracking outperforms simple backtracking techniques since it prunes more

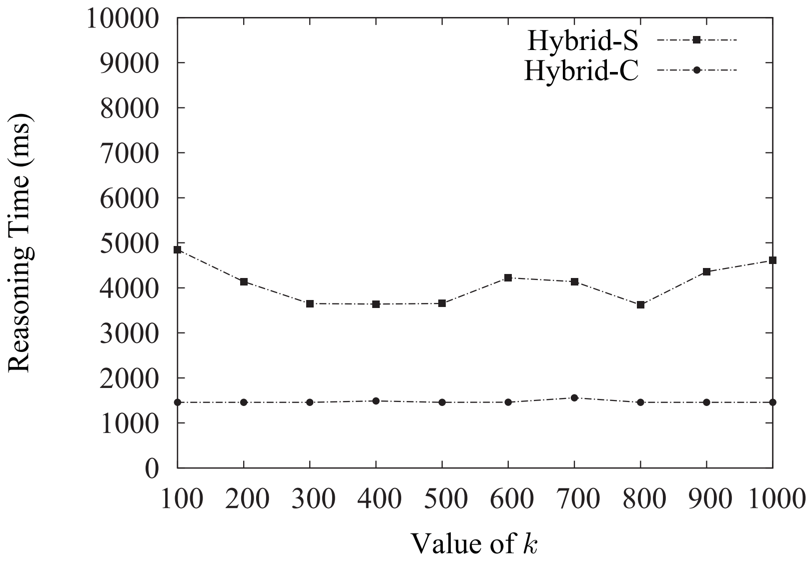


Figure 29: Complex backtracking vs simple backtracking

branches that lead to the same sort of clashes. The number of logical clashes and backtracks for this case are shown in Table 1.

Hybrid-S	**Logical Clash**	8	9	9	9	8	8	9	8
	Back-Track	7	8	8	8	7	7	8	7
Hybrid-C	**Logical Clash**	4	5	5	5	4	4	5	4
	Back-Track	3	4	4	4	3	3	4	3

Table 1: Number of logical clashes and backtracks

The number of backtracks presented in Table 1, highlights the improvement that using the complex backtracking method results in a smaller number of backtracking steps and consequently the algorithm terminates in less time compared to simple backtracking. According to the explanation given in Chapter 5 for these two techniques, simple backtracking only set one variables to zero, while complex backtracking sets all the variable that are causing the same clash to zero and obviously the complex technique can prune more branches from the search space. In addition to improving the performance of the reasoner the optimization techniques used in our hybrid reasoner can make its performance more stable.

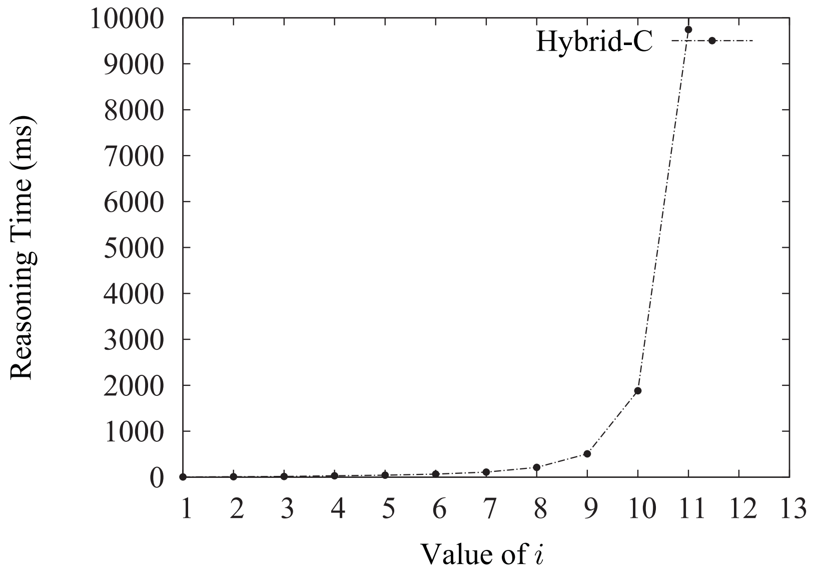


Figure 30: Increasing at-least QNR

6.2.4 The number of QNRs

There is no doubt that the arithmetic reasoning has a significant improvement for the cases with large values in number restrictions. However, this component can incur a higher processing cost. The run time to find an answer for a set of inequations depends on the number of variables occurring in the inequations and the number of inequations. As mentioned in section 4.1.3 and Section 4.2, if there exists m NRs then there would be $2^m - 1$ partitions for all possible combinations of the NRs, and consequently $2^m - 1$ variables. Therefore by increasing the number of NRs, the number of variables exponentially increases. Since the ch-Rule should consider two branches for each variable as $v \geqslant 1$ or $v \leqslant 1$, the number of time that the ch-Rule is applied dramatically increases. In fact the worst-case complexity of the hybrid algorithm is characterized by a double-exponential function of the number of cardinality restrictions.

Assume a concept $Test$ such that:

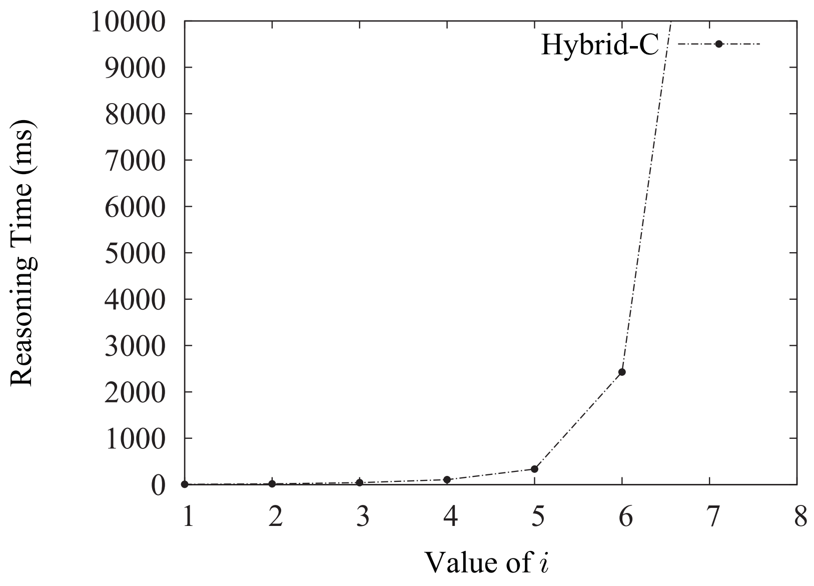


Figure 31: Increasing at-most QNR

$$Test \sqsubseteq \geqslant 1S.A \sqcap \forall S. \geqslant 1R.B \sqcap \forall S.\forall R.\forall S^{-}.P$$

$$\text{where } P = \{\sqcap \bowtie 1M_i.C_i \,|\, 1 \leqslant i \leqslant k\}$$

The diagrams in Fig. 30 and Fig. 31 show the effect of increasing the number of at-least and at-most restrictions in reasoning for the concept $Test$. Fig. 30, shows the reasoning time for concept $Test$ where $\bowtie$ is replaced with $\geqslant$ and $k = 14$. Fig. 31, shows the reasoning time for concept $Test$ where $\bowtie$ is replaced with $\leqslant$ and $k = 8$. In a model for the concept $Test$, the concept expression P is propagated back and will be added to the label of a node which already has $\geqslant 1R.B$, therefore, we have $(k + 1)$ QNRs. Since for each node which has a parent, an IBE will be considered as a set of two inequations and consequently two more QNRs will be added to the $\mathcal{L}_E$ of the node. At last we have $(k + 1) + 2$ QNRs. Accordingly there would be $2^{(k+2)} - 1$ variables to be considered in the arithmetic reasoner. A large number of roles in the QNRs may affect the performance of the hybrid calculus since it increases the number of variables and the size of the search space. Moreover, since the arithmetic always searches for a minimal solution, significantly affects the complexity of the reasoning even when no at-most restriction exists Comparing these two diagrams shows that by increasing the number of at-most NRs the reasoning time for the arithmetic reasoner increases faster than for at-least restrictions. The reason is the heuristic that we explained in section 5.2. By means of this heuristic (see Fig. 26), if a role occurs in an at-least restriction and not in any at-most restriction and has pre-conditions that are mentioned in section 5.2, then the potential variables for IBE which contain this role are set to zero. Therefore, the number of variables in search space is decreased.

6.3 Summary

In this chapter the implemented prototype reasoner was compared with state of the art reasoners. The effectiveness of reasoner in handling large values of number restrictions in presence of QNRs and consequently back propagation of QNRs are shown. Also the inefficiency of algorithm in dealing with large number of QNRs and consequently large number of variables was demonstrated.

Chapter 7

Conclusion and Future Work

The presented hybrid calculus in this thesis decides the satisfiability of $\mathcal{SHIQ}$ concepts. The implemented hybrid reasoner demonstrates the improvement on reasoning time featuring QNRs and inverse roles. Utilizing algebraic reasoning and applying optimization techniques, the hybrid calculus can be a good solution in case of large numbers for qualified number restrictions. In addition $\mathcal{SHIQ}$ can be further extended with nominals to a more expressive DL $\mathcal{SHOIQ}$.

7.1 Conclusion

Our presented approach uses algebraic reasoning which makes the reasoning time independent from the value of number restrictions. While other reasoner's reasoning time exponentially increases due to the exponential increase of cardinality value, our approach shows a stable behavior for some test cases. In absence of inverse roles the hybrid algorithm collects all the numerical restrictions before expanding the completion graph. Therefore it will generate a more informed solution with a good chance of survival. Therefore there is no need to merge as in standard tableau algorithms. The latter issue holds in the case of inverse roles. This is due to the fact that we still consider this edge as a numerical restriction before finding an answer for a node and generating new individuals. Therefore we never merge the proxy individuals, but if a modification is needed the algorithm recompute the atomic decomposition (see section 4.4). In case of back propagation of QNRs, our algorithm resets the arithmetic label of a corresponding node, discards its children and recalculates the atomic decomposition and finds a new answer.

In addition, benefiting from the atomic decomposition (see section 4.1.3), the search space can be modeled as a set of variables. This well structured search space, simplifies the process of backtracking in case of a clash. In other words, information is well presented as a set of variables and tracking become much easier. However, these variables are the source of inefficiency, since the algorithm introduces an exponential number of variables. As shown in Fig. 30 and Fig. 31 a large number of QNRs results in a large number of variables and the search to find a model (via ch-Rule) can become expensive.

The simplex method uses an objective method, which determine whether to minimize or maximize a solution [CLRS01]. As mentioned before, in the hybrid algorithm we minimize a solution in order to have a smaller completion graph. This issue can be considered as a benefit, since the size of a graph becomes smaller but, since it searches for an optimum solution, it can cost more processing time. Hence, the usefulness of the algorithm depends on the domain of the problems.

Considering all benefits and costs of the hybrid algorithm, for problems with large values for QNRs, and the presence of inverse roles the hybrid algorithm makes a significant improvement. According to the test cases demonstrated in Chapter 6, there exists cases such that the hybrid algorithm is necessary compute an answer in a short time. Obviously the whole argument depends on the application's domain and how it is used.

7.2 Future Work

One of the challenging extension for DL languages, is the combination of *Nominals* ($\mathcal{O}$), and *Inverse roles* ($\mathcal{I}$). The interaction between $\mathcal{O}$ and $\mathcal{I}$ results in higher level of complexity. Note that, by only adding $\mathcal{O}$ to the DL $\mathcal{ALCO}$ with the complexity of PSPACE-complete, leads to the DL $\mathcal{ALCOI}$ with the complexity of EXPTIME-complete. Moreover, if the DL $\mathcal{SHOQ}$ with complexity of EXPTIME-complete will be extended with $\mathcal{I}$, the new expressive DL $\mathcal{SHOICQ}$ has the complexity of NEXPTIME-complete. This is obviously highlights that the combination of the two constructors, $\mathcal{O}$ and $\mathcal{I}$, provides a significant increase in the complexity of the corresponding DL language. Therefore, this combination, would make a challenging topic, which may interest the DL researchers to overcome the problem of high complexity in practical reasoning for an expressive DL language. [Fad11] proposes a hybrid algorithm for the DL $\mathcal{SHOQ}$ using algebraic reasoning. Due to the nature of nominals, [Fad11] uses global partitioning in which by benefiting from several optimization techniques partitions are calculated on demand. In contrast, our hybrid algorithm performs local partitioning to avoid the calculating partitions as a pre-processing step and unnecessary partitions. An approach based on both $\mathcal{SHIQ}$ and $\mathcal{SHOQ}$ could be developed that benefits from optimizations presented in each DL, supporting DL $\mathcal{SHOIQ}$.

Bibliography

[BBH96] Franz Baader, Martin Buchheit, and Bernhard Hollander. Cardinality restrictions on concepts. *Artificial Intelligence*, 88(1-2):195–213, 1996.

[BCM^{+}07] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneide. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2nd edition, 2007.

[BFT95] Paolo Bresciani, Enrico Franconi, and Sergio Tessaris. Implementing and testing expressive description logics: a preliminary report. In *Knowledge Retrieval, Use and Storage for Efficiency: Proceedings of the First International KRUSE Symposium*, pages 131–139, 1995.

[BH91] Franz Baader and Bernhard Hollunder. Kris: Knowledge representation and inference system. *SIGART Bulletin*, 2(3):8–14, 1991.

[BHLW03] Franz Baader, Jan Hladik, Carsten Lutz, and Frank Wolter. From tableaux to automata for description logics. In Moshe Vardi and Andrei Voronkov, editors, *Proceedings of the 10th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR 2003)*, volume 2850 of *Lecture Notes in Computer Science*, pages 1–32. Springer, 2003.

[BHN^{+}92] Franz Baader, Bernhard Hollunder, Bernhard Nebel, Hans-Jürgen Profitlich, and Enrico Franconi. An empirical analysis of optimization techniques for terminological representation systems, or making kris get a move on. In *KR*, pages 270–281, 1992.

[BHS07] Franz Baader, Ian Horrocks, and Ulrike Sattler. Description Logics. In Frank van Harmelen, Vladimir Lifschitz, and Bruce Porter, editors, *Handbook of Knowledge Representation*. Elsevier, 2007.

[BPS94] Alex Borgida and Peter F. Patel-Schneider. A semantics and complete algorithm for subsumption in the classic description logic. *Journal of Artificial Intelligence Research*, 1:277–308, 1994.

[BS01] Franz Baader and Ulrike Sattler. An overview of tableau algorithms for description logics. *Studia Logica*, 69(1):5–40, 2001.

[CLRS01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Second Edition*. The MIT Press, 2nd edition, September 2001.

[DM00] Francesco M. Donini and Fabio Massacci. Exptime tableaux for alc. *Artif. Intell.*, 124:87–138, November 2000.

[Fad11] Jocelyne Faddoul. *Reasoning Algebraically with Description Logics*. PhD thesis, Department of Computer Science and Software Engineering, Concordia University, 2011.

[FFHM08] Jocelyne Faddoul, Nasim Farsinia, Volker Haarslev, and Ralf Möller. A hybrid tableau algorithm for $\mathcal{ALCQ}$. In *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI 2008), Patras, Greece, July 21-25*, pages 725–726, 2008.

[FH10a] Jocelyne Faddoul and Volker Haarslev. Algebraic tableau reasoning for the description logic $\mathcal{SHOQ}$. *Journal of Applied Logic (Special issue on Hybrid Logic)*, 8(4):334–355, December 2010.

[FH10b] Nasim Farsiniamarj and Volker Haarslev. Practical reasoning with qualified number restrictions: A hybrid Abox calculus for the description logic $\mathcal{SHQ}$. *AI Commun.*, 23(2-3):205–240, 2010.

[HB91] Bernhard Hollunder and Franz Baader. Qualifying number restrictions in concept languages. In *KR'91*, pages 335–346, 1991.

[HM92] Joseph Y. Halpern and Yoram Moses. A guide to completeness and complexity for modal logics of knowledge and belief. *Artificial Intelligence*, 54:311–379, 1992.

[HM01a] Volker Haarslev and Ralf Möller. Optimizing reasoning in description logics with qualified number restrictions. In *Proceedings International Workshop on Description Logics (DL-2001), Stanford, USA, 1.-3. August*, pages 142–151, 2001.

[HM01b] Volker Haarslev and Ralf Möller. Racer system description. In *IJCAR*, pages 701–706, 2001.

[Hor02] Ian Horrocks. Backtracking and qualified number restrictions: Some preliminary results. In *Proc. of the 2002 Description Logic Workshop (DL 2002)*, volume 63 of *CEUR*, pages 99–106, 2002.

[Hor03] Iian Horrocks. Implementation and optimisation techniques. In Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors, *The Description Logic Handbook: Theory, Implementation, and Applications*, chapter 9, pages 306–346. Cambridge University Press, 2003.

[HS05] Ian Horrocks and Ulrike Sattler. A tableaux decision procedure for $\mathcal{SHOIQ}$. In *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005)*, pages 448–453, 2005.

[HST99] Ian Horrocks, Ulrike Sattler, and Stephan Tobies. A description logic with transitive and converse roles role hierarchies and qualifying number restrictions. Technical report, 1999.

[HST00a] Ian Horrocks, Ulrike Sattler, and Stephan Tobies. Practical reasoning for very expressive description logics. *Logic Journal of the IGPL*, 8:2000, 2000.

[HST00b] Ian Horrocks, Ulrike Sattler, and Stephan Tobies. Reasoning with individuals for the description logic $\mathcal{SHIQ}$. In *Proceedings of the 17th International Conference on Automated Deduction*, CADE-17, pages 482–496, London, UK, 2000. Springer-Verlag.

[HT00] Ian Horrocks and Stephan Tobies. Reasoning with axioms: Theory and practice. In *Proc. of the 7th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2000)*, pages 285–296, 2000.

[HTM01] Volker Haarslev, Martina Timmann, and Ralf Möller. Combining tableaux and algebraic methods for reasoning with qualified number restrictions. In *Proceedings of the International Workshop on Description Logics (DL'2001), Aug. 1-3, Stanford, USA*, pages 152–161, 2001.

[KM06] Yevgeny Kazakov and Boris Motik. A resolution-based decision procedure for shoiq. In *Proc. of the 3rd Int. Joint Conf. on Automated Reasoning (IJCAR 2006), volume 4130 of LNAI*, pages 662–667. Springer, 2006.

[Luk05] Alena Lukasová. Reasoning with semantic tableau binary trees in description logic. In *Description Logics*, 2005.

[MSH07] Boris Motik, Rob Shearer, and Ian Horrocks. Optimized reasoning in description logics using hypertableaux. In *Proc. of the 21st Int. Conf. on Automated Deduction (CADE-21)*, volume 4603 of *Lecture Notes in Artificial Intelligence*, pages 67–83. Springer, 2007.

[MSH09] Boris Motik, Rob Shearer, and Ian Horrocks. Hypertableau reasoning for description logics. *J. of Artificial Intelligence Research*, 36:165–228, 2009.

[OK99] Hans Jürgen Ohlbach and Jana Koehler. Modal logics, description logics and arithmetic reasoning. *Artif. Intell.*, 109:1–31, April 1999.

[Pap81] Christos H. Papadimitriou. On the complexity of integer programming. *J. ACM*, 28:765–768, October 1981.

[SPG$^+$07] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical owl-dl reasoner. *Web Semant.*, 5:51–53, June 2007.

[SSS91] Manfred Schmidt-Schau and Gert Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26, 1991.

[THPS07] Dmitry Tsarkov, Ian Horrocks, and Peter F. Patel-Schneider. Optimizing terminological reasoning for expressive description logics. *J. of Automated Reasoning*, 39(3):277–316, 2007.