A Tableau-based Algebraic Calculus for Description Logic $\mathcal{SHOIQ}$

Humaira Farid

A Thesis

in

The Department

of

Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy (Computer Science) at

Concordia University

Montréal, Québec, Canada

May 2024

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: **Humaira Farid**

Entitled: **A Tableau-based Algebraic Calculus for Description Logic** $\mathcal{SHOIQ}$

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

Doctor of Philosophy (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. Ciprian Alecsandru

______________________________ External Examiner
Dr. David Toman

______________________________ Examiner
Dr. Jeremy Clark

______________________________ Examiner
Dr. Lata Narayan

______________________________ Examiner
Dr. Hovhannes Harutyunyan

______________________________ Supervisor
Dr. Volker Haarslev

Approved by ______________________________
Dr. Leila Kosseim, Graduate Program Director

__________ 2024

____
Dr. Mourad Debbabi, Dean
Gina Cody School of Engineering and Computer Science

Abstract

A Tableau-based Algebraic Calculus for Description Logic $\mathcal{SHOIQ}$

Humaira Farid, Ph.D.

Concordia University, 2024

The growing demand for efficient knowledge representation and reasoning in the era of interconnected systems and extensive data collection motivates this thesis. Addressing the limitations of existing Description Logic (DL) reasoners, we focus on the challenges posed by qualified cardinality restrictions (QCRs), nominals, and inverse roles. These constructs, though crucial for expressive ontologies, often hinder computational efficiency in traditional reasoning approaches. Consequently, real-world ontologies either exclude them or employ very small numerical values. This motivates our exploration of a novel reasoning approach, employing algebraic methods, aiming to enhance DL reasoning with a focus on large numerical restrictions.

In this thesis, a novel algebraic tableau calculus for $\mathcal{SHOIQ}$ is presented for deciding ontology consistency. This hybrid approach integrates standard tableau-based reasoning with algebraic reasoning to handle a large number of nominals, QCRs, and their interaction with inverse roles. The algorithm extends the previously presented algebraic tableau algorithm for $\mathcal{SHOI}$ [24, 25]. Numerical restrictions imposed by nominals and qualified number restrictions are encoded into a set of linear inequalities. The knowledge about other axioms, such as universal restrictions, role hierarchy, subsumption and disjointness, is also embedded in order to get a more informed mapping of QCR satisfiability to feasibility. Column generation and branch-and-price algorithms are used to solve these inequalities. The feasibility test for the linear inequalities can be computed in polynomial time, as shown in [51]. Rigorous proofs ensure soundness, completeness, and termination of the reasoning procedure.

In practice, the proposed reasoning approach, implemented in the Cicada prototype, demonstrates its effectiveness against existing state-of-the-art reasoners. Empirical evaluations using synthetic and ORE 2014 datasets reveal Cicada's robust performance against increasing numerical values, showcasing its viability for handling expressive ontologies. Despite its more focused optimization techniques, Cicada outperforms other reasoners in certain scenarios, providing a promising avenue for practical applications requiring efficient DL reasoning.

Acknowledgement

Embarking on the doctoral journey was a formidable challenge, and the completion of this thesis is a testament to the unwavering support and guidance I received along the way.

I extend my deepest gratitude to my supervisor, Dr. Volker Haarslev, whose mentorship, encouragement, and cooperative spirit were instrumental in shaping this research. His dedication to my learning journey and expertise in Description Logics laid the foundation for my understanding and accomplishments.

I am profoundly thankful to my supervisory committee for their invaluable time, patience, and insightful remarks throughout the years. Their constructive criticism played a pivotal role in refining the research and pushing its boundaries.

A special tribute goes to my late father, Muhammad Farid Khan, whose initial joy at my pursuit of a Ph.D. was a driving force. Though he is not present today, his pride in my success remains a cherished motivation. I would also like to fondly remember my late mother, Hussan Jan. I deeply miss her, and I am confident that she would be overjoyed by this significant achievement of mine.

The unyielding love, support, and encouragement from my siblings, nephews, nieces, and friends were indispensable pillars of strength. A sincere acknowledgement goes to my brothers, Sajjad Ahmed Qureshi, Muhammad Rafiq Qureshi, and sisters Gulshan Farid, and Nargis Sajjad; their constant backing made this achievement possible.

My heartfelt appreciation goes to my husband, Khawar, for his steadfast support throughout every phase of this journey. This thesis would not have reached completion without his dedicated encouragement and assistance.

Last but not least, I express my gratitude for the joy and motivation my daughters, Minha and Esha, brought into my life. Their presence served as a welcomed diversion from the challenges of Ph.D. life.

To all those who contributed to this journey, your belief in me and your roles in shaping this endeavour are eternally appreciated.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

This chapter serves as an introduction to the research conducted within this thesis. It commences by outlining the motivational aspects behind the undertaken research. Following this, the chapter details our motivation to research, encapsulating the definition of the problem, the establishment of goals and objectives, and the devised approach to achieve these objectives.

1.1 Motivation

Description Logic (DL) serves as a formal language for representing knowledge about concepts, individuals, and their relationships. Originally designed to extend semantic networks with formal logical semantics, DL has found extensive application in the semantic web, with the Web Ontology Language (OWL) being based on description logics. Its usage extends to various application domains, including medical informatics. Modern description logic systems provide reasoning services, enabling automated inference of implicit knowledge from explicitly provided information. Designing high-performance reasoning algorithms has been a central concern for DL researchers.

In description logics, a knowledge base consists of two main components: Terminological Knowledge (Tbox), and Assertional Knowledge (Abox). The Tbox defines the vocabulary and characteristics of the domain. It contains axioms that specify relationships, concepts, and their properties. On the other hand, the Abox contains the asserted knowledge about the actual individuals in the domain. These assertions specify information about individuals in terms of concepts they belong to and relationships they have with other individuals through certain roles. Together, these components form a comprehensive knowledge base that can be used for reasoning and making inferences in the specified domain.

The fundamental inference services related to Tboxes include *concept satisfiability* and *subsumption*. Concept satisfiability is determined by checking whether a concept description denotes the empty set. A concept C is deemed satisfiable if there is at least one individual in the domain that serves as an instance of C. Subsumption testing involves querying whether all instances of the concept C (the subsumee) are also instances of the concept D (the subsumer). In essence, subsumption examines whether the first concept can consistently be considered a subset of the second one. For instance, the axiom $Doctor \sqsubseteq Human$ asserts that all doctors are human. A basic Abox reasoning is *instance checking* which involves determining if an individual x, where x is a named individual in the Abox, is an instance of concept C. These inference problems should be both decidable and preferably of low complexity to ensure a reasonable and predictable behaviour of a DL system.

There are two primary approaches for providing reasoning services in DL: tableau-based reasoning, and consequence-based reasoning. In tableau reasoning, a set of tableau rules is utilized to construct a data structure known as a *tableau*. The first tableau-based algorithm was proposed in [64] for DL $\mathcal{ALC}$. Subsequently, this approach has been extended to accommodate more expressive DLs. More expressive DLs may include additional features and constructs such as cardinality restrictions, nominals, or inverse roles. However, these extra features contribute to higher computational complexity in reasoning. To enhance reasoning performance, the expressivity of the DL language can be reduced. This reduction, however, should not compromise the DL's ability to express crucial notions in the application domain. In contrast, consequence-based reasoning algorithms, as proposed in [2] for the lightweight logic $\mathcal{EL}$, focus on deriving logical consequences of axioms in the ontology using inference rules. Consequence-based methods aim to be more efficient than tableau-based approaches, offering a different perspective on deriving entailments and reasoning about DL ontologies. The choice between these reasoning approaches often involves a trade-off between expressiveness and computational efficiency, and the selection depends on the specific requirements and characteristics of the DL and its intended application domain.

Cardinality restrictions in DL enhance the language's expressive power by enabling the specification of numerical constraints on relationships. There are two types of cardinality restrictions: (i) unqualified cardinality restrictions ($\mathcal{N}$), expressed in the forms ($\geq nR$) or ($\leq mR$), specify the least or the most number of allowed role successors for an individual; (ii) qualified cardinality restrictions ($\mathcal{Q}$), shown by ($\geq nR.C$) or ($\leq mR.C$), additionally describe the type of individuals that are counted by a given number restriction. For example, the concept representation of $Canada \sqsubseteq \geq 12 hasProvince.CA\_Province$ states a necessary condition that an instance of the $Canada$ concept must have at least 12 successors through the $hasProvince$ role, and these successors must specifically belong to the concept $CA\_Province$.

One of the key features of many description logics is support for nominals ($\mathcal{O}$). Nominals are special concept names and they must be interpreted as singleton sets. They allow in particular for a direct combination of knowledge about individuals (Abox assertions) with terminological knowledge (Tbox axioms). In practical ontology applications, nominals often serve as identifiers for entities such as countries, persons, flavors, and more. For instance, we can use nominals to represent 10 provinces of Canada, namely Ontario, Quebec, Nova Scotia, New Brunswick, Manitoba, British Columbia, Prince Edward Island, Saskatchewan, Newfoundland and Labrador, Alberta, as follows:

$$CA\_Province \equiv \{Ontario \sqcup ... \sqcup Alberta\}$$

Here $Ontario$, $Alberta$, and others are distinct nominals, each representing a specific province.

Qualified Cardinality Restrictions (QCRs) carry explicit numerical restrictions and their restrictions are local, whereas, nominals carry implicit global numerical restrictions. For example, the representation of the concept $CA\_Province$ as an enumeration of 10 distinct nominals expresses an implicit cardinality restriction; the concept $CA\_Province$ is constrained to have exactly 10 individuals. These restrictions are global because they affect the set of all individuals of $CA\_Province$ in the domain. While these global cardinality restrictions enhance the expressiveness of description logics, they also contribute to practical complexity in reasoning and ontology management.

Inverse role ($\mathcal{I}$) is another DL construct that adds more expressiveness to a language. Inverse Roles are used to represent converse relationships between individuals. For instance, $hasChild \equiv hasParent^{-}$ signifies that $hasChild$ is the inverse of $hasParent$. This construct facilitates the representation of bidirectional connections within a knowledge base.

In summary, DLs that incorporate QCRs, nominals, and inverse roles benefit from enhanced expressiveness. However, this increased expressiveness comes at the cost of higher computational complexity in reasoning tasks. As a result, there exists an inherent trade-off between the richness of language features and the efficiency of algorithms delivering inference services.

1.2 Problem Statement

The extension of $\mathcal{ALC}$ with nominals and QCRs introduces powerful capabilities to express arithmetic constraints on both concepts and roles. However, it is essential to recognize that nominals carry implicit global numerical restrictions that increase the reasoning complexity. Moreover, the interaction between QCRs, nominals and inverse roles leads to the loss of the tree model property and the finite model property. This results in a complexity increase from ExpTime to NExpTime. Moreover, it makes the design of reasoning calculi more complicated.

Most state-of-the-art reasoners, such as Konclude [72], Fact++ [74], HermiT [66], have implemented traditional tableau algorithms. These non-algebraic reasoners attempt to construct completion models in a highly nondeterministic way in order to handle numerical restrictions. For example, consider a small ontology having the following axioms:

$$CA\_Province \equiv \{Ontario \sqcup ... \sqcup Alberta\} \tag{1}$$

$$Canada \sqsubseteq \geq 12 hasProvince.CA\_Province \tag{2}$$

Axiom (1) defines the concept $CA\_Province$ by enumerating all 10 provinces of Canada as nominals and these 10 nominals are pairwise disjoint. Axiom (2) states a necessary condition for the concept $Canada$ that Canada must have at least 12 provinces. It is trivial to see that one cannot satisfy the $\geq 12 hasProvince.CA\_Province$ QCR because the cardinality of $CA\_Province$ is implicitly restricted to the 10 provinces listed as nominals. However, according to our experiments, most DL reasoners are unable to decide this inconsistency within a reasonable amount of time.

In the above scenario, the primary cause of inefficiency lies in the nondeterministic merging process. To determine the satisfiability of the concept $Canada$, a standard tableau algorithm generates 12 distinct yet anonymous instances of $CA\_Province$. It then proceeds to nondeterministically attempt to merge these instances with the 10 nominals representing the Canadian provinces. This merging process continues until all possible combinations are explored, and eventually, the unsatisfiability of $Canada$ is determined. This lack of consideration for numerical constraints can lead to significant performance degradation, impacting not only the size of the completion models but also introducing a substantial degree of non-determinism. The issue becomes more pronounced when dealing with a large number of nominals or when handling substantial numerical values, as illustrated in the following concept description:

$$Human \sqsubseteq \geq 600 hasMuscles(Skeletal \sqcup Smooth \sqcup Cardiac)$$

However, algebraic DL reasoners are considered more efficient in handling numerical restrictions [22, 26, 36, 75]. RacerPro [36] was the first highly optimized reasoner that combined tableau-based reasoning with algebraic reasoning [37]. Other tableau-based algebraic reasoner for $\mathcal{SHQ}$ [26], $\mathcal{SHIQ}$ [58], $\mathcal{SHOQ}$ [22, 23] are also proposed to handle QCRs and their interaction with inverse roles or nominals. These reasoners use an atomic decomposition technique to encode number restrictions into a set of linear inequalities. These inequalities are then solved by integer linear programming (ILP). These reasoners perform very efficiently in handling huge values in number restrictions. Nevertheless, their ILP algorithms are best-case exponential to the number of inequalities. For example, in the case of m inequalities they require 2^m variables in order to find the optimal solution. Therefore, it is not feasible to enumerate all variables for ILP with a huge number of variables. To overcome this problem, the column generation technique has been used [75, 77] which considers a small subset of variables. Integer programming has also been applied for finite model reasoning [59, 50, 60]. However, to the best of our knowledge, no algebraic calculus can handle DLs supporting nominals, QCRs and inverse roles simultaneously.

1.3 Research Objectives

The research presented in this thesis is centred around developing a reasoning approach for Description Logics (DLs) that effectively handles a substantial number of nominals, Qualified Cardinality Restrictions (QCRs), and their interactions with inverse roles. The chosen reasoning approach is hybrid, combining a standard tableau-based reasoning algorithm for DL with algebraic reasoning.

The primary objectives of adopting this reasoning approach can be summarized as follows:

1. **Expressive Language Support:** The thesis aims to provide reasoning support for DL languages that enable the expression of all elements within an application domain using available logical language constructs. Specifically, the focus is on entailments based on the combination of nominals ($\mathcal{O}$), inverse roles ($\mathcal{I}$), and qualified cardinality restrictions ($\mathcal{Q}$), achieving the expressivity of $\mathcal{SHOIQ}$, which is nearly the full expressivity of OWL utilized in Semantic Web applications.

2. **Hybrid Reasoning Approach:** While algebraic DL reasoners are acknowledged for their efficiency in managing numerical restrictions, there exists a challenge in finding a calculus

capable of handling DLs that support nominals, QCRs, and inverse roles simultaneously. Therefore, the proposed reasoning approach aims to integrate both tableau and algebraic reasoning to support these expressive DL constructs. The objective is to address the challenge of efficiently handling numerical restrictions using Integer Linear Programming (ILP). These restrictions are encoded as inequalities and solved using the branch-and-price technique. The proposed algorithm aspires to encode additional knowledge about axioms such as universal restrictions, role hierarchy, subsumption, and disjointness for more informed reasoning.

3. **Correctness Emphasis:** The reasoning procedure must ensure both soundness and completeness. Soundness implies that every "yes" answer in an inference test is valid, while completeness ensures that every "no" answer is also valid.

4. **Termination and Efficiency:** Developing sound and complete decision procedures for highly expressive DL languages is crucial. However, if these procedures do not terminate or fail to respond within a reasonable time, the entire system loses its utility. Therefore, the proposed system aims to incorporate a suite of optimization techniques to ensure efficiency without compromising correctness or termination.

1.4 Thesis Outline

The remainder of this thesis is organized as follows:

- Chapter 2 provides background knowledge and defines the formal syntax and semantics of Description Logics (DL).

- Chapter 3 offers a brief overview of existing approaches, discusses related systems developed for handling Qualified Cardinality Restrictions (QCRs), nominals, and inverse roles, and explores state-of-the-art optimization techniques.

- Chapter 4 presents the proposed Algebraic Tableau Calculus for SHOIQ, illustrating the generation of the inequality system using numerical restrictions and related information. It explains the column generation and branch-and-price technique for solving inequalities and provides proofs of soundness, completeness, and termination.

- Chapter 5 introduces the prototype reasoner, Cicada, implemented to demonstrate the practical applicability of the proposed hybrid approach.

- Chapter 6 conducts an empirical evaluation of Cicada against existing state-of-the-art reasoners. It describes the datasets and metrics used for system evaluation and results comparison.

- Chapter 7 concludes the thesis with a summary, and suggests future directions.

Chapter 2

Preliminaries

In this section, we provide some important definitions and introduce notations used later. The aim of this chapter is to give an overview of the structure and expressiveness of Description Logic. The syntax and semantics of concept descriptions in basic DL language $\mathcal{ALC}$ are discussed in section 2.1.1. Section 2.2 provides an overview of the expressive constructs of DL $\mathcal{SHOIQ}$. The different approaches for DL reasoning adopted by most state-of-the-art DL reasoners and the complexity of different DL languages are also discussed in section 2.3. Section 2.4 introduces the algebraic method that is also used for description logic reasoning.

2.1 Description Logic

Description Logic (DL) is a formal knowledge representation language that is used for modeling ontologies. Modern description logic systems provide reasoning services that can automatically infer implicit knowledge from explicitly expressed knowledge. Description logic mainly models three types of entities; *concepts* that represent sets of individuals, *roles* that represent binary relations between individuals of the domain, and *individuals* that are elements of the domain of reasoning.

Definition 1. (Concept) A concept represents a set of elements of the domain with similar characteristics. For example, Woman is the concept which represents all persons that are female. Similarly, the concept Child represents the set of individuals who are offsprings of a person. The concepts that cannot have any common elements are declared as *disjoint* concepts. For instance, the concepts Male and Female can be considered as disjoint concepts.

Definition 2. (Role) A role defines binary relationships between concepts. For example, hasChild is a role that defines the relationship between two concepts, Woman and Child.

Definition 3. (Individual) An individual is a named element of the domain. For instance, the concept Woman has an individual Mary (Mary : Woman), and the concept Child has an individual John (John : Child). Roles can be used to define the relationship between a pair of individuals, (e.g., Mary, John : hasChild).

These concepts, roles and individuals are used to represent the knowledge about the domain of interest. From this explicitly represented knowledge one can infer the implicit knowledge. For instance, the concept Mother is defined as a Woman who has at least one Child, and Mary is a Woman who hasChild John. By using the reasoning services one can easily infer that Mary is a Mother. Similarly, the concept Grandmother is defined as a Mother who has at least one Child who is a Person and that child also has at least one Child. Mary is a Mother, John is a Person, Peter is a Person, Mary hasChild John and John hasChild Peter. One can easily infer from the given knowledge that Mary is a Grandmother.

The expressiveness of a DL language depends on the set of constructors it provides for building complex concepts and roles. The DL $\mathcal{ALC}$ is the basic DL language that provides the smallest set of DL constructors (conjunction $\sqcap$, disjunction $\sqcup$, negation $\neg$, existential restriction $\exists$, universal restriction $\forall$). The formal syntax and semantics of the DL $\mathcal{ALC}$ are discussed in the following section.

The main focus throughout this thesis is on the DL $\mathcal{SHOIQ}$ which extents the DL $\mathcal{ALC}$ with more expressive constructs such as transitive roles ($\mathcal{S}$), roles hierarchies ($\mathcal{H}$), nominals ($\mathcal{O}$), inverse roles ($\mathcal{I}$), and qualified cardinality restrictions ($\mathcal{Q}$).

Table 2.1: Syntax and semantics of concept descriptions in $\mathcal{ALC}$.

Syntax	**Semantics**
A (concept)	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
$\top$ (universal or top concept)	$\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$
$\bot$ (empty or bottom concept)	$\bot^{\mathcal{I}} = \emptyset$
$\neg A$ (negation)	$\Delta^{\mathcal{I}} \setminus A^{\mathcal{I}}$
$C \sqcap D$ (conjunction)	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
$C \sqcup D$ (disjunction)	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
$\forall R.C$ (universal restriction)	$\{x \in \Delta^{\mathcal{I}} \mid \forall y : (x, y) \in R^{\mathcal{I}} \Rightarrow y \in C^{\mathcal{I}}\}$
$\exists R.C$ (qualified existential restriction)	$\{x \in \Delta^{\mathcal{I}} \mid \exists y \in \Delta^{\mathcal{I}} : (x, y) \in R^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\}$

2.1.1 Basic DL Language $\mathcal{ALC}$

The base DL language, called $\mathcal{AL}$, is introduced in [64]. $\mathcal{AL}$ contains atomic concepts, atomic negations, top and bottom concepts, conjunction for concept expressions, and unqualified existential restrictions.

$\mathcal{AL}$ is then extended to $\mathcal{ALC}$, which is one of the basic DL languages that is propositionally complete. The syntax and semantics of concept descriptions in $\mathcal{ALC}$ are shown in Table 2.1. $\mathcal{ALC}$ contains disjunction and negation for concept expressions, and qualified existential restrictions in addition to $\mathcal{AL}$.

For example, we use the following concept description to define the concept of "A woman that is married to an Engineer, and all of their children are either Lawyers or Artists":

$$\mathsf{Human} \sqcap \neg\mathsf{Male} \sqcap (\exists \mathsf{married}.\mathsf{Engineer}) \sqcap (\forall \mathsf{haschild}.(\mathsf{Lawyer} \sqcup \mathsf{Artist}))$$

The semantics of concept descriptions can be defined in terms of standard Tarski-style semantics based on an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\Delta^{\mathcal{I}}$ is a non-empty set of individuals called the domain of interpretation and $\cdot^{\mathcal{I}}$ is an interpretation function. An interpretation function $\cdot^{\mathcal{I}}$ associates each role name R with a binary relation $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ and each concept name C with a subset $C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$.

Definition 4. (Concept Inclusion Axiom) Concept inclusion axioms are expressed in the following forms:

- $C \sqsubseteq D$ (*Concept Subsumption Axiom*) where C is subsumed by D. For example, $\mathsf{Mother} \sqsubseteq \mathsf{Parent}$.
- $C \equiv D$ (*Concept Definition Axiom*) which is a substitute for $\{C \sqsubseteq D, D \sqsubseteq C\}$. If C is a concept name then this axiom is said to be a primitive definition. For example, $\mathsf{Woman} \equiv \mathsf{Human} \sqcap \mathsf{Female}$.

In the case where C is a complex class expression, a concept inclusion axiom is referred to as a General Concept Inclusion (GCI) axiom. For example, we use the following GCI to define a constraint that "only professors can teach to graduate students":

$$\exists \mathsf{teachesTo}.\mathsf{GradStudent} \sqsubseteq \mathsf{Professor}$$

Here, the left-hand side of the axiom is not a simple concept name.

Definition 5. (Tbox) A Tbox $\mathcal{T}$ is the *terminological knowledge* of the domain that contains a finite set of concept inclusion axioms of the form $C \sqsubseteq D$, and/or $C \equiv D$.

Definition 6. (Abox) An Abox $\mathcal{A}$ with respect to a Tbox $\mathcal{T}$ is a finite set of assertions of the form

- $a : C$ where a is an individual and $C \in N_C$ and N_C denotes a set of concepts (e.g., $\mathsf{Mary} : \mathsf{Woman}$),
- $(a, b) : R$ where $R \in N_R$ and N_R denotes a set of roles (e.g., $\mathsf{Mary}, \mathsf{John} : \mathsf{hasChild}$), and
- $a \neq b$ where a and b are individuals (e.g., $\mathsf{Mary} \neq \mathsf{John}$).

Definition 7. **($\mathcal{ALC}$ Knowledge Base)** A knowledge base $\mathcal{K}$ in description logics is composed of two parts: the *terminological knowledge* (Tbox $\mathcal{T}$) that describes the vocabulary and characteristics of the domain, and the *assertional knowledge* (Abox $\mathcal{A}$) that contains the asserted knowledge regarding the actual individuals of the domain, shown in Figure 2.1.

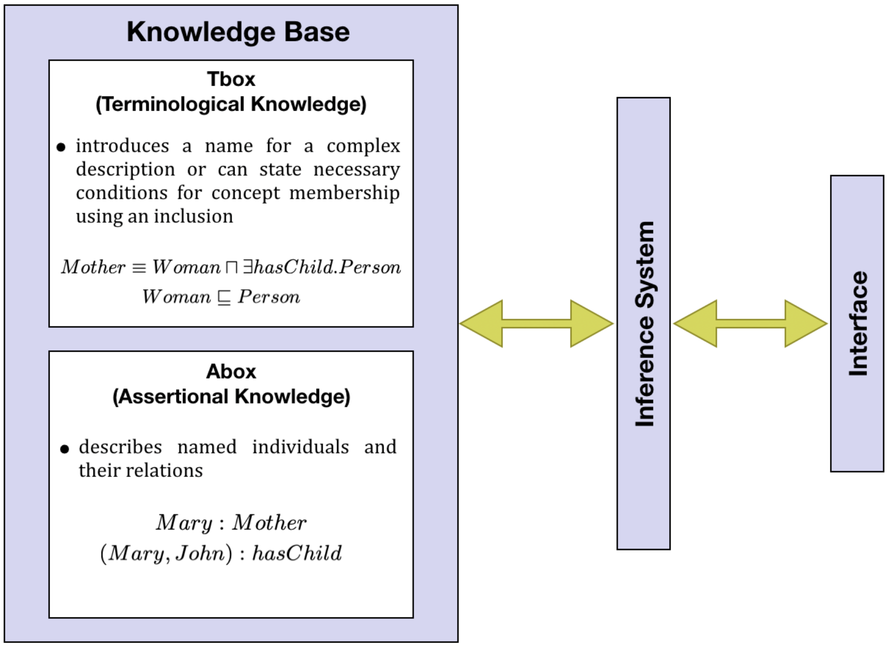

Knowledge Base
Tbox
(Terminological Knowledge)
• introduces a name for a complex description or can state necessary conditions for concept membership using an inclusion
$Mother \equiv Woman \sqcap \exists hasChild.Person$
$Woman \sqsubseteq Person$
Abox
(Assertional Knowledge)
• describes named individuals and their relations
$Mary : Mother$
$(Mary, John) : hasChild$
Inference System
Interface

Figure 2.1: Basic DL knowledge base consisting of a Tbox and an Abox

An interpretation $\mathcal{I}$ is said to be a model of the knowledge base $\mathcal{K}$ if $\mathcal{I}$ is a model of the Tbox $\mathcal{T}$ and the Abox $\mathcal{A}$. A knowledge base $\mathcal{K}$ is consistent if it has a model.

2.2 Description Logic $\mathcal{SHOIQ}$

$\mathcal{SHOIQ}$ extends the description logic $\mathcal{ALC}$ with transitive roles, role hierarchies, singleton concepts (nominals), inverse roles and qualified cardinality restrictions (QCRs).

Definition 8. **(R-filler)** Assume a and b are two individuals. b is called an R-filler of a if $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$ holds for a given role $R \in N_R$. The set of all R-fillers of a is defined as $Fil(a, R) = \{b \mid (a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}\}$.

Definition 9. (Role Hierarchy ($\mathcal{H}$)) A role hierarchy is used to define subrole and super-role relationship between the roles. For example, hasMother is a subrole of hasParent and hasParent is a super-role of hasMother.

A set of role inclusion axioms are used to express these role hierarchies.

Definition 10. (Role Inclusion Axiom (RIA)) A role inclusion axiom is expressed in the form $R \sqsubseteq S$ where $R, S \in N_R$ and R is called a subrole of S and S is a super-role of R. For example, the RIA $hasMother \sqsubseteq hasParent$ specifies that every filler of the hasMother role must be a filler of the hasParent role. The interpretation of an axiom $R \sqsubseteq S$ using the interpretation function $\cdot^{\mathcal{I}}$ is equal to $R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$ which means that for each pair of individuals if $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$ is true then $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in S^{\mathcal{I}}$ should also be true.

Definition 11. (Transitive Role ($\mathcal{S}$)) If a role name R is declared as transitive then

$$(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}} \wedge (b^{\mathcal{I}}, c^{\mathcal{I}}) \in R^{\mathcal{I}} \Rightarrow (a^{\mathcal{I}}, c^{\mathcal{I}}) \in R^{\mathcal{I}}$$

A set $N_{R^+} \subseteq N_R$ denotes the set of transitive roles. For example, hasAncestor $\in N_{R^+}$ implies that the ancestor of one's ancestor is also considered as his or her ancestor.

A role is called *simple* if it is neither transitive nor has a transitive subrole.

Definition 12. (Inverse Role ($\mathcal{I}$)) Inverse roles are used to represent converse relationships between individuals. For example, $hasChild \equiv hasParent^{-}$ means that $hasChild$ is the inverse of $hasParent$. For any role $R \in N_R$, if a role R^{-} is interpreted as the inverse of R, then the semantics of inverse roles by the interpretation function $\cdot^{\mathcal{I}}$ is represented as

$$(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}} \Leftrightarrow (b^{\mathcal{I}}, a^{\mathcal{I}}) \in (R^{-})^{\mathcal{I}}$$

Definition 13. (Nominals ($\mathcal{O}$)) Nominals are Abox named individuals that can be used within concept descriptions in Tbox. They must be interpreted as singleton sets and they allow one to express the notion of uniqueness and identity. For example, nominals can be used to model the planets or the countries, e.g., "Earth", "Moon", "Canada", "Pakistan", etc. A set $N_o \subseteq N_C$ denotes a set of nominals. Within a concept description, a nominal name is enclosed in curly brackets "{}" to distinguish it from a concept name. Nominals can be defined as the *hasValue* and the *oneOf* constructor.

- The *oneOf* constructor is used to enumerate the nominals to define a concept, e.g,

  $$\mathsf{Continent} \equiv \{\mathsf{Africa}, \mathsf{Antarctica}, \mathsf{Australia}, \mathsf{Asia}, \mathsf{Europe}, \mathsf{NorthAmerica}, \mathsf{SouthAmerica}\}$$

  where $\mathsf{Africa}, ..., \mathsf{SouthAmerica}$ are all nominals.

- In the *hasValue* constructor, nominals are used as a part of an existential restriction, e.g,

  $$\mathsf{AfricanCountry} \sqsubseteq \exists \mathsf{locatedIn}.\{\mathsf{Africa}\}$$

  where Africa is a nominal and a concept $\mathsf{AfricanCountry}$ is defined as "a country located in the continent of Africa".

The interpretation of the nominal o using the interpretation function $\cdot^{\mathcal{I}}$ is equal to $\sharp\{o\}^{\mathcal{I}} = 1$ which means that o is a singleton set. Here, $\sharp$ denotes set cardinality. Nominals carry implicit global numerical restrictions. For example, the concept $\mathsf{Continent}$ which is defined by enumerated 7 nominals can have exactly 7 instances.

Moreover, nominals also introduce non-determinism because $\{o_1, o_2, o_3\}$ denotes a disjunction of nominals. For example, the definition of $\mathsf{Continent}$ in axiom (3), which is using an enumeration of nominals, is equivalent to a disjunction of nominals in axiom (4).

$$\mathsf{Continent} \equiv \{\mathsf{Africa}, \mathsf{Antarctica}, \mathsf{Australia}, \mathsf{Asia}, \mathsf{Europe}, \mathsf{NorthAmerica}, \mathsf{SouthAmerica}\} \tag{3}$$

$$\mathsf{Continent} \equiv \{\mathsf{Africa}\} \sqcup \{\mathsf{Antarctica}\} \sqcup ... \sqcup \{\mathsf{NorthAmerica}\} \sqcup \{\mathsf{SouthAmerica}\} \tag{4}$$

Therefore, nominals increase the reasoning complexity. Furthermore, the presence of nominals leads to the loss of the tree model property (see Definition 23).

Definition 14. (Unqualified cardinality restrictions ($\mathcal{N}$)) Unqualified cardinality restrictions ($\mathcal{N}$), specify the least or the most number of allowed role fillers for an individual. For example, the concept inclusion axiom $Person \sqsubseteq (\leq 2hasParent) \sqcap (\geq 2hasParent)$ indicates that every individual that is a member of the concept Person has exactly two (distinct) parents.

Definition 15. (Qualified cardinality restrictions ($\mathcal{Q}$)) In qualified cardinality restrictions ($\mathcal{Q}$), the cardinality restriction, along with the restriction on the number of role fillers, also specifies the concept to which these fillers belong. For example, the assertion $a : (\leq 1hasParent.Male)$ states that a has at most one $hasParent$-filler that is also a member of the concept $Male$. Similarly, the assertion $a : (\geq 3hasChild.Female)$ states that a has at least three $hasChild$-fillers that are also members of the concept $Female$.

Let $N = N_C \cup N_o$ where N_C represents concept names and N_o nominals. The set of roles in $\mathcal{SHOIQ}$ is $N_R \cup \{R^- \mid R \in N_R\}$. A function Inv returns the inverse of a role such that $\mathsf{Inv}(R) = R^-$ if $R \in N_R$ and $\mathsf{Inv}(R) = S$ if $R = S^-$ and $S \in N_R$. We use $\top$ ($\bot$) as an abbreviation for $A \sqcup \neg A$ ($A \sqcap \neg A$) for some $A \in N_C$. We denote with $\sqsubseteq_*$ the transitive, reflexive closure of $\sqsubseteq$ over N_R. In the following $\sharp$ denotes set cardinality. Table 2.2 presents the syntax and semantics of QCRs, nominals and other expressive constructs of $\mathcal{SHOIQ}$.

Qualified cardinality restrictions (QCRs) carry explicit numerical restrictions and their restrictions are local, whereas, nominals carry implicit global numerical restrictions. Suppose an individual x is an instance of a concept C and $C \sqsubseteq \geq 3R.D$ where $C, D \in N_C$ and $R \in N_R$. It imposes that at least 3 individuals of D, say x_1, x_2 and x_3, must be R-fillers of x. These restrictions are local since they only affect the set of individuals that are R-fillers of x. On the other hand, if $C \sqsubseteq \{o_1, o_2, o_3\}$ (or $\{o_1, o_2, o_3\} \sqsubseteq C$), then o_1, o_2, o_3 impose a numerical restriction that there can be at most (or at least, provided $o_1, o_2, o_3 \in N_o$ are pairwise disjoint) three instances of C. These restrictions are global because they affect the set of all individuals of C in $\Delta^{\mathcal{I}}$. These cardinality

Table 2.2: Syntax and semantics of DL $\mathcal{SHOIQ}$

Construct	**Syntax**	**Semantics**
Nominals		
$\mathcal{O}$	$\{o\}$	$\sharp\{o\}^{\mathcal{I}} = 1$
Number Restrictions		
$\mathcal{N}$	$\geq nR.\top$ $\leq mR.\top$	$\{x \in \Delta^{\mathcal{I}} \mid \sharp\{y \in \Delta^{\mathcal{I}} \mid (x, y) \in R^{\mathcal{I}}\} \geq n\}$ $\{x \in \Delta^{\mathcal{I}} \mid \sharp\{y \in \Delta^{\mathcal{I}} \mid (x, y) \in R^{\mathcal{I}}\} \leq m\}$
Qualified Cardinality Restrictions		
$\mathcal{Q}$	$\geq nR.C$ $\leq mR.C$	$\{x \in \Delta^{\mathcal{I}} \mid \sharp\{y \in \Delta^{\mathcal{I}} \mid (x, y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\} \geq n\}$ $\{x \in \Delta^{\mathcal{I}} \mid \sharp\{y \in \Delta^{\mathcal{I}} \mid (x, y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\} \leq m\}$
Inverse Role		
$\mathcal{I}$	R^{-}	$(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}} \Leftrightarrow (b^{\mathcal{I}}, a^{\mathcal{I}}) \in (R^{-})^{\mathcal{I}}$
Role Hierarchy		
$\mathcal{H}$	$R \sqsubseteq S$	$R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$
Transitive Role		
$\mathcal{S}$	$R \in N_{R_+}$	$(a^{\mathcal{I}}, b^{\mathcal{I}}) : R^{\mathcal{I}} \wedge (b^{\mathcal{I}}, c^{\mathcal{I}}) : R^{\mathcal{I}} \Rightarrow (a^{\mathcal{I}}, c^{\mathcal{I}}) : R^{\mathcal{I}}$

restrictions significantly increase the expressiveness of a DL language, however, they also raise its practical complexity.

2.3 Description Logic Reasoning

One of the most interesting aspects of DLs is that they provide several reasoning services regarding domain knowledge. By using these services one can obtain some implicit knowledge about the domain. In order to check the logical consistency of the ontology, the reasoner performs two tasks:

1. the Tbox consistency test, which involves the satisfiability test of all concept names, and
2. the Abox consistency test, in which the reasoner verifies whether the Abox is consistent w.r.t. the Tbox by considering the Abox assertions.

The most basic services regarding Tboxes are *concept satisfiability,* which verifies the satisfiability of a concept, and *subsumption,* which checks subconcept-superconcept relationships.

Definition 16. (Satisfiability) A concept description C is said to be satisfiable by an interpretation $\mathcal{I}$ iff $C^{\mathcal{I}} \neq \emptyset$, i.e., there exists an individual $x \in \Delta^{\mathcal{I}}$ as an instance of C such that $x \in C^{\mathcal{I}}$.

Definition 17. (Subsumption) A concept description D subsumes concept description C ($C \sqsubseteq D$) iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for all interpretations $\mathcal{I}$, i.e., the first concept description is always interpreted as the subset of the second one.

For Aboxes, *instance checking* is a basic reasoning service which inquires if individual a is a member of concept C where a is a named individual in the Abox and the reasoner needs to consider the relevant Abox assertions as well as the Tbox.

In DLs, which support full negation, subsumption and satisfiability can be reduced to one another. A concept description C is subsumed by D, $C \sqsubseteq D$, iff $C \sqcap \neg D$ is unsatisfiable. Similarly, a concept C is unsatisfiable if the subsumption $C \sqsubseteq \bot$ is entailed.

A Tbox $\mathcal{T}$ and its associated role hierarchy $\mathcal{R}$ is satisfied by $\mathcal{I}$ (or consistent) if each GCI and RIA is satisfied by $\mathcal{I}$. Such an interpretation $\mathcal{I}$ is then called a model of $\mathcal{T}$. Due to the nominals, a concept assertion $a : C$ can be transformed into a concept inclusion $\{a\} \sqsubseteq C$ and a role assertion $(a, b) : R$ into $\{a\} \sqsubseteq \exists R.\{b\}$. Therefore, concept satisfiability and Abox consistency can be reduced to Tbox consistency by using nominals. For example, the concept C is satisfiable w.r.t. the Tbox $\mathcal{T}$ iff $\mathcal{T} \cup (\{a\} \sqsubseteq C)$, where $a \in N_o$, is satisfiable and the Abox $\mathcal{A}$ is consistent w.r.t. the Tbox $\mathcal{T}$ iff $\mathcal{T} \cup ((\{a\} \sqsubseteq \exists R.\{b\}) \sqcap (\{a\} \sqsubseteq C))$ is satisfiable. We use $\{o_1, \ldots, o_n\}$ as an abbreviation for $\{o_1\} \sqcup \cdots \sqcup \{o_n\}$ and may write $\{o\}$ as o if it is clear from the context that o is a nominal.

In order to prove concept satisfiability or test Tbox consistency, tableau algorithms try to construct a representation of a model by constructing a *completion graph*. A completion graph $G = (V, E, \mathcal{L})$ is a directed graph where V is a set of nodes representing individuals. Each node $x \in V$ is labelled with a set of concepts $\mathcal{L}(x)$, and each edge between x and y, $(x, y) \in E$ is labelled with a set of role names $\mathcal{L}(x, y)$.

A completion graph is usually used as a data structure to describe an abstraction of a model for a given knowledge base $\mathcal{K}$.

Definition 18. (R-successor, R-predecessor, R-neighbour) If a node y is a successor of a node x and $S \in \mathcal{L}(\langle x, y\rangle)$ with $S \sqsubseteq R$ where $S, R \in N_R$, then y is called an R-successor of a node x and x is called an $\mathsf{Inv}(R)$-predecessor of y. A node y is called an R-neighbour of x iff y is an R-successor or an R-predecessor of x.

Definition 19. (Decidability) A DL reasoning algorithm is said to be *decidable* if its soundness, completeness, and termination can be proved. *Soundness* ensures that something can be inferred only if it is entailed by a knowledge base $\mathcal{K}$, *completeness* ensures anything that is entailed by a knowledge base $\mathcal{K}$ can be inferred, and *termination* means that the algorithm always terminates.

There are two major approaches to provide reasoning services:

- Tableau-based reasoning
- Consequence-based reasoning

2.3.1 Tableau-based Reasoning

The most widely used approach to provide reasoning services is tableau reasoning. Tableau reasoning is composed of a set of tableau rules which are applied by a tableau algorithm which constructs a tableau. A tableau is a data structure first introduced in [64] for $\mathcal{ALC}$ which was later on extended for various other DLs. The tableau algorithm checks the satisfiability of a given concept C by attempting to construct a finite interpretation $\mathcal{I}$ which contains an element x for which $x \in C^{\mathcal{I}}$ [6]. For convenience, we assume that all concept descriptions are in negation normal form (NNF).

Definition 20. (Negation Normal Form (NNF)) A concept description is in negation normal form if the negation sign ($\neg$) only appears in front of concept names (atomic concepts). Concept descriptions can be transformed into an equivalent description in NNF in linear time for any $\mathcal{ALC}$-concept description [6]. We compute the NNF of concept expressions using

- de Morgan's rules (e.g., $\neg(C \sqcap D) \equiv \neg C \sqcup \neg D, \neg(C \sqcup D) \equiv \neg C \sqcap \neg D$),

$\sqcap$-Rule	**if** $(C \sqcap D) \in \mathcal{L}(x)$ and $\{C, D\} \nsubseteq \mathcal{L}(x)$ **then** set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C, D\}$
$\sqcup$-Rule	**if** $(C \sqcup D) \in \mathcal{L}(x)$ and $\{C, D\} \cap \mathcal{L}(x) = \emptyset$ **then** set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C\}$ or $\mathcal{L}(x) = \mathcal{L}(x) \cup \{D\}$
$\exists$-Rule	**if** $(\exists R.C) \in \mathcal{L}(x)$ and there is not a node y with $R \in \mathcal{L}(x, y)$, $C \in \mathcal{L}(y)$ **then** create a new node y and set $\mathcal{L}(y) = \mathcal{L}(y) \cup \{C\}$ and $\mathcal{L}(x, y) = \mathcal{L}(x, y) \cup \{R\}$
$\forall$-Rule	**if** $(\forall R.C) \in \mathcal{L}(x)$ and there exists a node y with $R \in \mathcal{L}(x, y)$ and $C \notin \mathcal{L}(y)$ **then** set $\mathcal{L}(y) = \mathcal{L}(y) \cup \{C\}$

Figure 2.2: The expansion rules for $\mathcal{ALC}$ tableau algorithm

- conversion rules of existential and universal restrictions (e.g., $\neg(\forall R.C) \equiv \exists R.(\neg C), \neg(\exists R.C) \equiv \forall R.(\neg C)$), and

- the usual rules for quantifiers (e.g., $\neg(\leq nR.C) \equiv \geq n+1R.C, \neg(\geq nR.C) \equiv \leq n-1R.C$).

The Expansion Rules for DL $\mathcal{ALC}$:

In order to prove concept satisfiability or test Tbox consistency, tableau algorithms try to construct a representation of a model by constructing a completion graph (see Definition **??**). A tableau algorithm converts all subsumption relations of the form $C \sqsubseteq D$ to their equivalent NNF($\neg C \sqcup D$) then reduces all the concept axioms in Tbox $\mathcal{T}$ to a single axiom $\top \sqsubseteq C_{\mathcal{T}}$ where $C_{\mathcal{T}}$ is the conjunction of NNF of all axioms occurring in $\mathcal{T}$. Then the algorithm applies the completion rules, shown in Figure 2.2, to the initial Abox $\mathcal{A} = \{a : C_{\mathcal{T}}\}$, where a is a fresh individual.

These rules decompose the concepts in node labels ($\mathcal{L}$) by either inferring new constraints for a given node or extending the completion graph according to these constraints. For example, if $C_1 \sqcap C_2 \in \mathcal{L}(x)$ then the $\sqcap$-rule adds both C_1 and C_2 to $\mathcal{L}(x)$, if either $C_1 \notin \mathcal{L}(x)$ or $C_2 \notin \mathcal{L}(x)$. However, the $\exists$-Rule extends the completion graph by adding a new node. For example, if $\exists R.C \in \mathcal{L}(x)$, and x does not yet have an R-successor with C in its label, then the $\exists$-rule generates a new R-successor node y of x with $\mathcal{L}(y) = \{C\}$.

The $\sqcup$-Rule is *nondeterministic* because a given Abox is transformed into more than one new Aboxes such that the original Abox is consistent if one of the new Aboxes is consistent. For

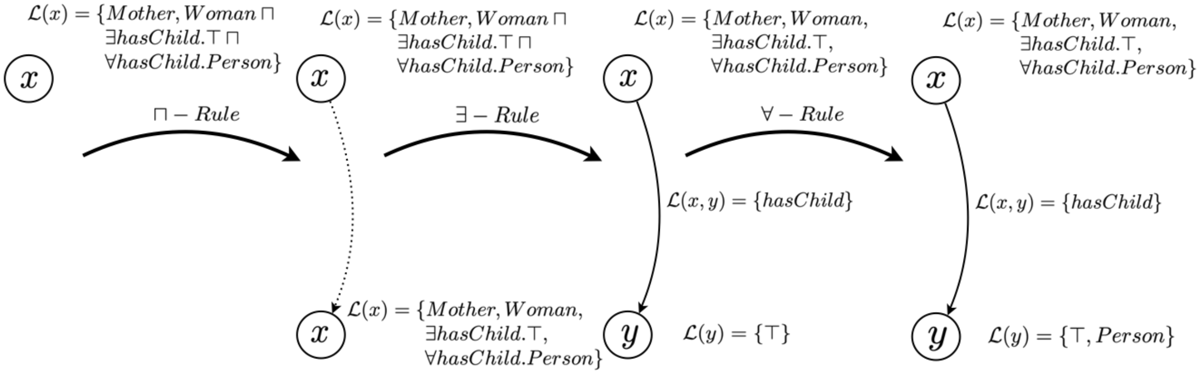

$\mathcal{L}(x) = \{Mother, Woman \sqcap \exists hasChild.\top \sqcap \forall hasChild.Person\}$
x
$\sqcap - Rule$
$\mathcal{L}(x) = \{Mother, Woman \sqcap \exists hasChild.\top \sqcap \forall hasChild.Person\}$
x
$\mathcal{L}(x) = \{Mother, Woman, \exists hasChild.\top, \forall hasChild.Person\}$
x
$\exists - Rule$
$\mathcal{L}(x) = \{Mother, Woman, \exists hasChild.\top, \forall hasChild.Person\}$
x
$\mathcal{L}(x,y) = \{hasChild\}$
y
$\mathcal{L}(y) = \{\top\}$
$\forall - Rule$
$\mathcal{L}(x) = \{Mother, Woman, \exists hasChild.\top, \forall hasChild.Person\}$
x
$\mathcal{L}(x,y) = \{hasChild\}$
y
$\mathcal{L}(y) = \{\top, Person\}$

Figure 2.3: The application of the tableau completion rules in order to test the concept satisfiability

example, if $C_1 \sqcup C_2 \in \mathcal{L}(x)$ and neither $C_1 \in \mathcal{L}(x)$ nor $C_2 \in \mathcal{L}(x)$, then it adds either C_1 or C_2 to $\mathcal{L}(x)$. In practice, the $\sqcup$-Rule is the main source of complexity in tableau algorithms, because it may be necessary to explore all possible choices of rule applications. The algorithm must backtrack if it reaches an obvious contradiction known as a clash (see Definition 21). If a concept is unsatisfiable then all possible expansions will lead to a clash. However, in the case where a concept is satisfiable, the algorithm finds at least one expansion that leads to a complete and clash-free completion graph.

Definition 21. (Clash) A node x has an obvious contradiction if there exists a concept expression C such that $\{C, \neg C\} \subseteq \mathcal{L}(x)$.

The step-by-step application of the completion rules for a satisfiability test of the concept Mother has been shown in Figure 2.3. The concept Mother is defined as follows:

$$\mathsf{Mother} \sqsubseteq \mathsf{Woman} \sqcap \exists \mathsf{hasChild}.\top \sqcap \forall \mathsf{hasChild}.\mathsf{Person}$$

An Abox is satisfiable if the algorithm produces a complete clash-free completion graph from the initial Abox. The algorithm terminates if

1. the Abox $\mathcal{A}$ contains a clash, or

2. none of the rules from Figure 2.2 is applicable to Abox $\mathcal{A}$, in this case $\mathcal{A}$ is called *complete*.

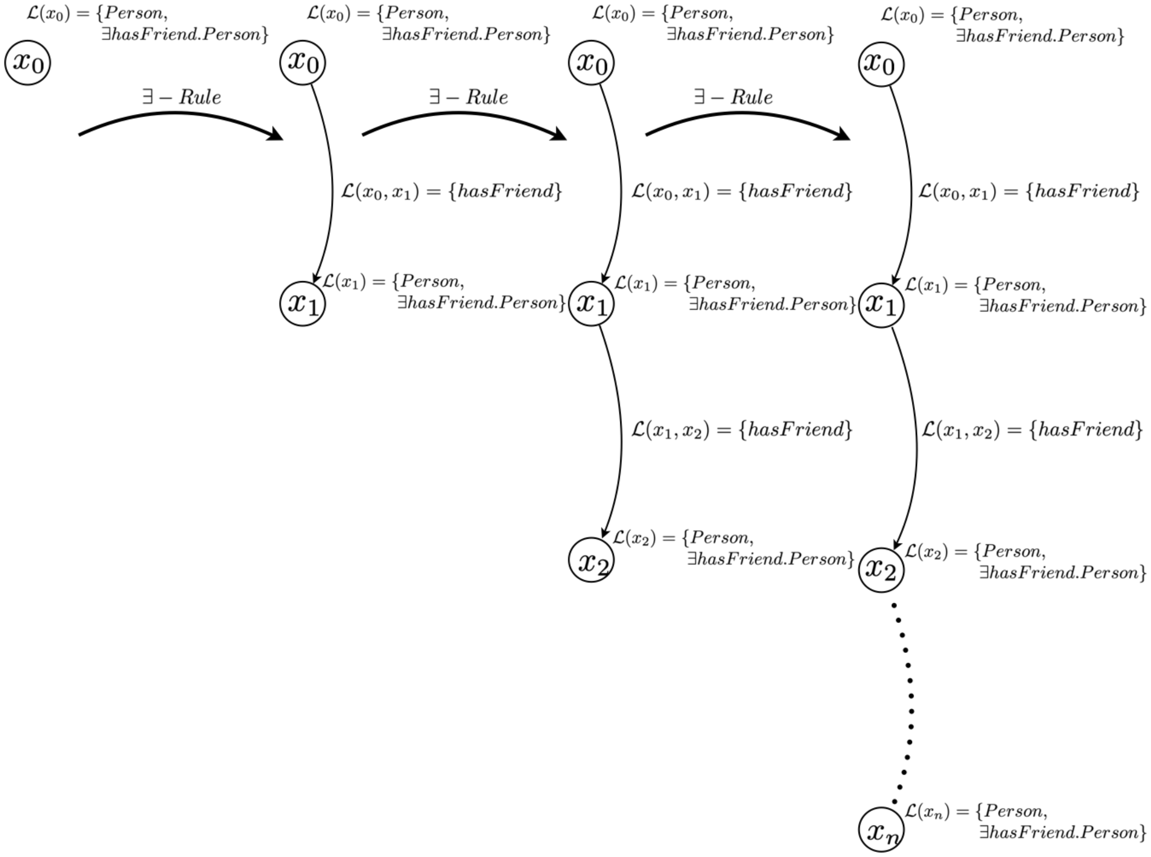

$\mathcal{L}(x_0) = \{Person, \exists hasFriend.Person\}$
x_0
$\exists - Rule$
$\mathcal{L}(x_0) = \{Person, \exists hasFriend.Person\}$
x_0
$\mathcal{L}(x_0, x_1) = \{hasFriend\}$
$\mathcal{L}(x_1) = \{Person, \exists hasFriend.Person\}$
x_1
$\exists - Rule$
$\mathcal{L}(x_0) = \{Person, \exists hasFriend.Person\}$
x_0
$\mathcal{L}(x_0, x_1) = \{hasFriend\}$
$\mathcal{L}(x_1) = \{Person, \exists hasFriend.Person\}$
x_1
$\mathcal{L}(x_1, x_2) = \{hasFriend\}$
$\mathcal{L}(x_2) = \{Person, \exists hasFriend.Person\}$
x_2
$\exists - Rule$
$\mathcal{L}(x_0) = \{Person, \exists hasFriend.Person\}$
x_0
$\mathcal{L}(x_0, x_1) = \{hasFriend\}$
$\mathcal{L}(x_1) = \{Person, \exists hasFriend.Person\}$
x_1
$\mathcal{L}(x_1, x_2) = \{hasFriend\}$
$\mathcal{L}(x_2) = \{Person, \exists hasFriend.Person\}$
x_2
$\mathcal{L}(x_n) = \{Person, \exists hasFriend.Person\}$
x_n

Figure 2.4: Application of tableau expansion rule without blocking

Blocking

In some cases, expanding the completion graph does not lead to a complete graph. This can happen if a Tbox contains cyclic inclusion axioms (where a concept name appears on both sides of the axiom). For example, if a Tbox $\mathcal{T}$ contains the axiom: $A \sqsubseteq \exists R.A$, then the algorithm can go on generating new individuals with repeating structure and the satisfiability of A w.r.t. $\mathcal{T}$ will never stop. Figure 2.4 shows that during a satisfiability test of the concept Person (defined in axiom 5), the algorithm generates new individuals repeatedly which leads to a non-terminating tableau model.

$$\mathsf{Person} \sqsubseteq \exists \mathsf{hasFriend.Person} \tag{5}$$

Therefore, in order to guarantee termination, the algorithm needs an extra mechanism called

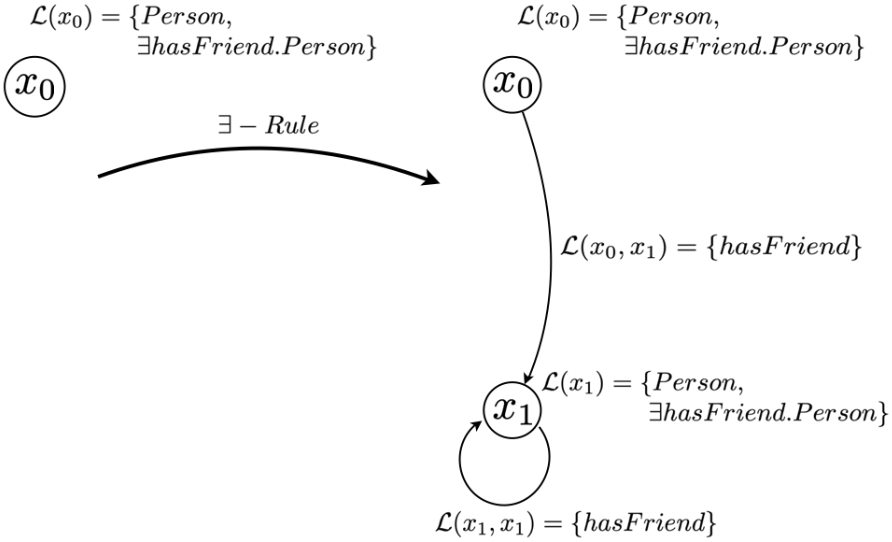

$\mathcal{L}(x_0) = \{Person, \exists hasFriend.Person\}$
x_0
$\exists - Rule$
$\mathcal{L}(x_0) = \{Person, \exists hasFriend.Person\}$
x_0
$\mathcal{L}(x_0, x_1) = \{hasFriend\}$
$\mathcal{L}(x_1) = \{Person, \exists hasFriend.Person\}$
x_1
$\mathcal{L}(x_1, x_1) = \{hasFriend\}$

Figure 2.5: Application of tableau expansion rule with blocking

blocking. In a blocking algorithm, cyclic computations are detected to prevent the further application of expansion rules. The main idea behind the blocking mechanism is to prevent a node from applying expansion rules if it needs to satisfy a concept expression already satisfied by one of its ancestors. For example, in the previous example (shown in Figure 5), the node x_1 can use the role successors of x_0 instead of generating new ones as shown in Figure 2.5.

A node x is called a *blocked* node by a node y if

- there is an ancestor y of x such that $\mathcal{L}(x) \subseteq \mathcal{L}(y)$ (*subset blocking*), or
- if there is an ancestor z of x such that z is blocked.

If a node x is blocked and none of its ancestors is blocked, then we say that x is *directly blocked*.

Subset blocking is sufficient for logics without inverse roles. However, blocking is more complex when inverse roles are added to the logic. For inverse roles, the blocking condition must be based on label *equality,* such that $\mathcal{L}(x) = \mathcal{L}(y)$, and a block should not be established on a once and for all basis. The equality blocking is important because inverse roles can propagate back additional concepts which can invalidate the model. Moreover, further expansion in other parts of the tree can invalidate the block by extending the labels of the blocking and/or blocked nodes. Therefore, *dynamic blocking* was introduced in [39], where blocks can be established and broken dynamically.

Extending the logic further with functional restrictions[1] or QCRs requires a more sophisticated blocking strategy, called *pairwise blocking* [39]. The pairwise blocking is discussed in more detail in the next chapter.

2.3.2 Consequence-based Reasoning

Although the tableau-based approach is currently the most widely used technique for reasoning in DLs, other approaches have been developed as well. A recent notable progress in DL reasoning is consequence-based reasoning. The reasoning algorithm proposed in [2] for the lightweight logic $\mathcal{EL}$ can be seen as the first such calculus.

For certain logics and tasks, other approaches are preferable to the tableau-based approach. For example, it is shown in [11, 2] that subsumption in $\mathcal{EL}$ remains polynomial even in the presence of general TBoxes but it is not clear how can we obtain the same results for subsumption in $\mathcal{EL}$ by using a tableau-based algorithm.

For a given DL $\mathcal{L}$, an $\mathcal{L}$-terminology (called $\mathcal{L}$-TBox) is a finite set $\mathcal{T}$ of axioms of the form $C \sqsubseteq D$ (called general concept inclusion (GCI)) or $A \doteq D$ (called definition) or $r \sqsubseteq s$ (called simple role inclusion axiom (SRI)), where C and D are concept descriptions defined in $\mathcal{L}$, $A \in N_{con}$, and $r, s \in N_{role}$ where N_{con} denotes a set of concepts names and N_{role} denotes a set of role names. A concept name $A \in N_{con}$ is called *defined* in $\mathcal{T}$ iff $\mathcal{T}$ contains one or more axioms of the form $A \sqsubseteq D$ or $A \doteq D$. A TBox that contains GCIs is called *general*. The DL $\mathcal{EL}$ admitting SRIs in TBoxes is denoted by $\mathcal{ELH}$.

The polynomial-time subsumption algorithm for $\mathcal{ELH}$, proposed in [11], simultaneously computes all subsumption relationships between the concept names occurring in TBox $\mathcal{T}$. In the first step, the algorithm normalizes the Tbox and after that, it computes implication sets.

A general $\mathcal{ELH}$-Tbox is normalized if it only contains GCIs and SRIs and all of the GCIs have

[1] They are restrictions of the form $\leq 1R$ that allow an individual to relate to at most one other individual by the role R.

one of the following forms:

$$C \sqsubseteq D \qquad C_1 \sqcap C_2 \sqsubseteq D \qquad C \sqsubseteq \exists r.D \qquad \exists r.C \sqsubseteq D$$

where C, C_1, C_2, D are concept names. A given TBox can be transformed into a normalized one by applying normalization rules. An example is used to illustrate these rules, and GCIs that need further rewriting are underlined (adapted from [2] [5]):

$$\exists r.C \sqcap \exists r.\exists s.C \sqsubseteq C \sqcap D \quad \rightsquigarrow \quad \exists r.C \sqsubseteq D_1, \underline{D_1 \sqcap \exists r.\exists s.C \sqsubseteq C \sqcap D},$$

$$D_1 \sqcap \exists r.\exists s.C \sqsubseteq C \sqcap D \quad \rightsquigarrow \quad \underline{\exists r.\exists s.C \sqsubseteq D_2}, \underline{D_1 \sqcap D_2 \sqsubseteq C \sqcap D},$$

$$\exists r.\exists s.C \sqsubseteq D_2 \quad \rightsquigarrow \quad \exists s.C \sqsubseteq D_3, \exists r.D_3 \sqsubseteq D_2,$$

$$D_1 \sqcap D_2 \sqsubseteq C \sqcap D \quad \rightsquigarrow \quad D_1 \sqcap D_2 \sqsubseteq C, D_1 \sqcap D_2 \sqsubseteq D$$

In the second step, for every concept name $A \in N_{con}^{\mathcal{T},\top}$[2], the algorithm defines an implication set $S(A)$. Initially $S(A) := \{A, \top\}$ which is then extended by applying completion rules. Similarly, for every role r the algorithm defines $S(r)$ which is the set of all roles included in r. These sets satisfy the following invariants: (i) for every $A, B \in N_{con}^{\mathcal{T},\top}$, $B \in S(A)$ implies $A \sqsubseteq_{\mathcal{T}} B$, i.e., $S(A)$ contains only subsumers of A. (ii) for every $r, s \in N_{role}^{\mathcal{T}}$, $s \in S(r)$ implies $r \sqsubseteq_{\mathcal{T}} s$.

The fact that subsumption in $\mathcal{EL}$ with respect to general TBoxes can be decided in polynomial time is proved by showing that (i) $\mathcal{T}$ can be normalized in polynomial time, and (ii) the sets $S(A)$ and $S(r)$ can be computed in polynomial time in the size of $\mathcal{T}$. Moreover, it is also proved in [11] that subsumption becomes co-NP hard when adding one of the constructors number restriction ($\leq n.r$ and $\geq n.r$), disjunction ($\sqcup$), and allsome ($\forall\exists$)[3].

This result is extended in [2] to the DL $\mathcal{EL}^{++}$, which extends $\mathcal{EL}$ with the bottom concept (and

[2] $N_{con}^{\mathcal{T},\top} := N_{con}^{\mathcal{T}} \cup \{\top\}$

[3] A concept $\forall\exists.C$ is equivalent to $\forall.C \sqcap \exists r.C$

thus disjointness constraints on concepts in the form of $C \sqcap D \sqsubseteq \bot$), nominals, a restricted form of concrete domains (e.g., reference to numbers and strings), and a restricted form of role-value maps which can express transitivity (e.g., $r \circ r \sqsubseteq r$) and the right-identity rule (e.g., $r \circ s \sqsubseteq r$ whose right-hand side is composition of role names). They proved that the subsumption problem remains tractable when adding these constructors. Additionally, they demonstrated that the extension of $\mathcal{EL}$ with any standard DL constructor not present in $\mathcal{EL}^{++}$ leads to the intractability of the subsumption problem. They used the term CBox to represent an $\mathcal{EL}^{++}$ constraint box which is defined as a finite set of GCIs and role inclusions (RIs). After normalizing CBox $\mathcal{C}$, the algorithm computes mappings S and R which satisfies two invariants: (i) $B \in S(A)$ implies $A \sqsubseteq_{\mathcal{C}} B$, and (ii) $(A, B) \in R(r)$ implies $A \sqsubseteq_{\mathcal{C}} \exists r.B$. Then the sets $S(A)$ and $R(r)$ are extended by applying completion rules. They also proved that once the completion algorithm has terminated, all subsumption relationships between concept names occurring in $\mathcal{C}$ can be determined.

$\mathcal{EL}^{++}$ is further extended in [3] with reflexive roles (i.e., role inclusions of the form $\varepsilon \sqsubseteq r$) and range restrictions. They impose a restriction on the structure of TBoxes in order to avoid intractability. To apply the subsumption algorithm for the original version of $\mathcal{EL}^{++}$, they first converted TBoxes into a normal form and after that they eliminated the range restrictions. They also proved that range restrictions can be eliminated in quadratic time without loss of any (non)-subsumption. This extended version of $\mathcal{EL}^{++}$ allows for the capture of additional ontologies, such as certain versions of the thesaurus of the US National Cancer Institute (NCI)[4].

2.3.3 Tableau-based Reasoning vs. Consequence-based Reasoning

Tableau-based and consequence-based methods are two well-known approaches to reasoning in DLs. Tableau-based methods work by building a counter model to test entailments. Whereas, consequence-based methods work by computing all possible logical consequences of axioms in the ontology using inference rules. Historically, consequence-based methods have been applied to lightweight DLs, most prominently, the $\mathcal{EL}$ family [11, 2, 3], for which they are more efficient

[4] https://www.cancer.gov/

than tableau, both theoretically and practically. However, consequence-based reasoning procedures become incomplete if the ontology is extended with axioms that use features of more expressive Description Logics, e.g., disjunctions. On the other hand, tableau-based methods are mostly used for very expressive fragments of the DL family because reasoning in expressive logics requires various forms of case analysis and tableau can handle cases via backtracking, whereas, designing inference rule systems has not been easy. The tableau algorithms are also easier to extend to new constructors because they follow more closely their semantics.

However, as discussed earlier, a fundamental reasoning problem in applications of DLs is to determine the subsumption i.e., whether each instance of a concept C is also an instance of a concept D in all models of an ontology. For expressive DLs, this problem has a high worst-case complexity. Tableau-based algorithms construct a finite representation of a canonical model of the ontology in order to test subsumption. In some cases, these algorithms construct very large model representations, which is a source of performance problems. Therefore, the performance of such algorithms is relatively brittle and there are some ontologies (e.g., GALEN[5], SNOMED CT[6]) that none of these algorithms is able to process.

In order to provide the more robust performance of reasoning and to make reasoning easy, one can reduce language expressivity. However, the expressive power of the DL must be restricted in an appropriate way so that it can express the important notions of the application domain.

It is also investigated to combine different reasoning techniques in order to improve the expressive power of DL and to provide efficient reasoning services at the same time. For that purpose, an approach for tightly coupling tableau-based and consequence-based saturation procedures is proposed in [69]. This approach is implemented in the OWL DL reasoner Konclude [72] and evaluation shows that this combination significantly improves the reasoning performance on a wide range of ontologies.

[5] http://www.opengalen.org/

[6] http://www.ihtsdo.org/snomed-ct

Table 2.3: The worst-case complexity of reasoning in DL Languages

DL Languages	**KB Consistency (General TBox)**	**Finite model property**	**Tree model property**
$\mathcal{EL}$	PTime-complete	Yes	Yes
$\mathcal{ALC}$	ExpTime-complete	Yes	Yes
$\mathcal{SHIQ}$	ExpTime-complete	No	No
$\mathcal{SHOQ}$	ExpTime-complete	Yes	No
$\mathcal{SHOI}$	ExpTime-complete	-	No
$\mathcal{SHOIQ}$	NExpTime-complete	No	No

2.3.4 Complexity of Reasoning

Description logic reasoning complexity increases with the expressivity of the DL language. There are also two important properties, finite model property and tree model property, that represent important characteristics of DL that significantly speed up the reasoning.

Definition 22. (Finite model property) DL has a *finite model property* if the consistent knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ has a model $\mathcal{I}$ where $\Delta^{\mathcal{I}}$ is a finite set (called *finite model*).

Definition 23. (Tree model property) DL has a *tree model property* if a concept C is satisfiable w.r.t. the knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, and $\mathcal{K}$ has a tree-shaped model $\mathcal{I}$ whose root belongs to $C^{\mathcal{I}}$.

Most expressive Description Logics (DLs) do not enjoy the finite model property and tree model property. Therefore, there is always a trade-off between the complexity and expressivity of the DL language. The goal is that DL should be sufficiently expressive, and reasoning should be decidable with an acceptable run-time behaviour and complexity. However, contemporary implementations of DLs exhibit reasonable runtime behavior on real-world problems, even though reasoning in these DLs is ExpTime-complete. The worst-case complexity of reasoning in DL Languages is shown in Table 2.3.

2.4 The Algebraic Method

Algebraic reasoning was initially introduced in [54] for set description languages. Later, in [55], they investigated how the concept subsumption and the consistency problem can be reduced to equation-solving problems.

2.4.1 Atomic Decomposition

The atomic decomposition technique for reasoning about sets was first proposed in [54]. The atomic decomposition technique encodes the numerical restrictions on concepts and role fillers into inequalities. These inequalities are then solved to decide the satisfiability of the numerical restrictions. Suppose we have a finite set of sets, $S = \{s_1, ..., s_n\}$, the atomic decomposition considers all possible ways to decompose S into mutually disjoint atomic sets.

For example, assume we want to translate the following numerical restrictions into arithmetic inequalities (adapted from [54]):

$$\leq 3 hasSon \sqcap \leq 3 hasDaughter \sqcap \geq 5 hasChild \tag{6}$$

here, $hasSon \sqsubseteq hasChild$ and $hasDaughter \sqsubseteq hasChild$ and $S = \{hasChild, hasSon, hasDaughter\}$, for short $S = \{c, s, d\}$. In the case of the entailment in 6, the atomic decomposition operates on sets of $hasSon$-fillers, $hasDaughter$-fillers, and $hasChild$-fillers, represented using arbitrarily overlapping sets Son, $Daughter$, $Children$, respectively. Consequently, a partitioning $\mathcal{D}$ is established, encompassing all subsets of S except the empty set, as illustrated in Figure 2.6, therefore,

$$\mathcal{D} = \{\{c\}, \{s\}, \{d\}, \{cs\}, \{cd\}, \{sd\}, \{csd\}\}$$

The partitions for this set of restrictions can be defined using set conjunctions and complement operations, as:

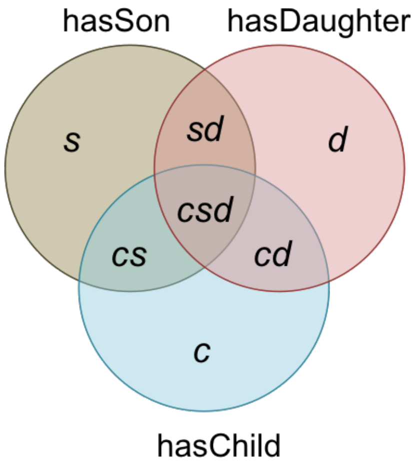


Figure 2.6: Atomic Decomposition on $S = \{hasChild, hasSon, hasDaughter\}$

$$c = children \cap \neg son \cap \neg daughter$$

$$s = \neg children \cap son \cap \neg daughter$$

$$d = \neg children \cap \neg son \cap daughter$$

$$cs = children \cap son \cap \neg daughter$$

$$cd = children \cap \neg son \cap daughter$$

$$sd = \neg children \cap son \cap daughter$$

$$csd = children \cap son \cap daughter$$

Since the decomposed subsets are mutually disjoint, we can encode these numerical restrictions into the following inequalities:

$$\leq 3hasSon \implies s + cs + sd + csd \leq 3$$

$$\leq 3hasDaughter \implies d + cd + sd + csd \leq 3$$

$$\geq 5hasChild \implies c + cs + cd + csd \geq 5$$

2.4.2 Integer Linear Programming

Linear Programming (LP) is the study of determining the minimum (or maximum) value of a linear function $f(x_1, x_2, ..., x_n)$ subject to a finite number of linear constraints. These constraints consist of linear inequalities involving variables $x_1, x_2, ..., x_n$ [13].

Definition 24. (Linear Function) If $c_1, c_2, ..., c_n$ are real numbers, then the function f of real variables $x_1,\ x_2, ...,\ x_n$ defined by

$$f(x_1,\ x_2, ...,\ x_n) = c_1x_1 + c_2x_2 + ... + c_nx_n = \sum_{j=1}^{n} c_jx_j$$

is called a linear function.

Definition 25. (Linear Constraints) If f is a linear function and if b is a real number, then the equation

$$f(x_1,\ x_2, ...,\ x_n) = b$$

is called a *linear equation* and the inequalities

$$f(x_1,\ x_2, ...,\ x_n) \leq b$$
$$f(x_1,\ x_2, ...,\ x_n) \geq b$$

are called *linear inequalities*. Linear equations and linear inequalities both are called *linear constraints*.

We call f the objective function which must be either minimized or maximized. A linear program is called a *maximization linear program* if we are to maximize the objective function, whereas it is called a *minimization linear program* if we are to minimize the objective function. Linear programs can be written under the *standard form*:

Maximize

$$\sum_{j=1}^{n} c_jx_j \tag{7}$$

Subject to

$$\sum_{j=1}^{n} a_{ij}x_j \leq b \quad (i = 1, 2, ..., m) \tag{8}$$

$$a_{ij} \in \mathbb{R}, \quad x_j \geq 0 \ \ (j = 1, 2, ..., n) \tag{9}$$

All constraints are linear inequalities and all variables are non-negative. The variables x_j are referred to as *decision variables*. A n-tuple $(x_1, ..., x_n)$ satisfying the constraints of a linear program is a *feasible solution* for this problem. A solution that maximizes the objective function of the problem is called an *optimal solution*.

If all of the variables are required to have integer values, then the problem is called *Integer Programming* (IP) or *Integer Linear Programming* (ILP).

The *Simplex method*, proposed by G. B. Dantzig [17], is one of the most frequently used methods to solve LP problems. The simplex algorithm takes as input a linear program and returns an optimal solution by traversing the boundary of the feasibility space. Although LP is known to be solvable in polynomial time [48], the simplex method can behave exponentially for certain problems. Karmarkar's algorithm [44] is the first efficient polynomial time method for solving a linear program. In contrast to the simplex method, Karmarkar's algorithm finds an optimal solution by traversing the interior of the feasibility space. However, all these approaches are not sufficiently efficient in solving problems with a huge number of variables. Therefore, decomposition techniques were considered to address such challenges.

2.4.3 Column Generation

The column generation technique is used to solve problems with a huge number of variables in order to either find an optimal integer solution or to detect their infeasibility. The column generation method was first proposed in the context of a multi-commodity network flow problem by Ford and Fulkerson [27]. Afterward, Dantzig and Wolfe adapted the column generation technique for solving LP problems and proposed an algorithm called Dantzig–Wolfe decomposition [18]. They proposed the idea of decomposing a large LP into the master problem and the subproblem. This technique was then implemented by Gilmore and Gomory [29, 30] to solve the cutting stock problem.

The column generation method is based on the concept of *duality*. According to the duality principle, optimization problems can be viewed from two perspectives, (a) the primal problem and

(b) the dual problem. For example, LP (7) - (9) is a primal problem and its dual problem is (10) - (12), where y_i is a vector of dual variables associated with constraints (8):

Minimize

$$\sum_{i=1}^{m} b_i y_i \tag{10}$$

Subject to

$$\sum_{i=1}^{m} a_{ij} y_j \geq c_j \quad (j = 1, 2, ..., n) \tag{11}$$

$$y_i \geq 0 \quad (i = 1, 2, ..., m) \tag{12}$$

If LP has an optimal solution, and the objective value of dual is the same as the primal, then it is a strong duality. Conversely, it is considered weak duality if the objective value of the dual problem at any feasible solution is greater than or equal to the objective value of the primal.

One can explain the duality theory using the concept of columns, where x_j is a decision variable and its associated column, and c_j is the cost of column j.

If for LP (7) - (9) the number of variables n is much larger than the number of constraints m, then the column generation method works with a subset of variables $n' \subseteq n$ and builds the Restricted Master Problem (RMP). The column generation method works in the following steps:

1. Start with a small set of columns (RMP)

2. Solve RMP in order to obtain the current optimal objective function value $\bar{x}$ and dual values associated with its constraints.

3. Use these dual values to provide prices to the subproblem called pricing problem (PP) and solve PP to identify the column with negative reduced cost.

4. Check if the cost is greater or equal to 0. If yes, then $\bar{x}$ cannot be improved, therefore, terminate the process. Otherwise, add a new column in RMP and go to step 2.

2.4.4 Branch-and-bound algorithm

The branch-and-bound framework [19] is mostly used for designing the algorithms for a large class of integer programs. This method is based on the column generation technique and it decomposes the total set of feasible solutions into smaller subsets. These smaller subsets are then evaluated until the optimal solution is found. Mostly it is in the form of a decision tree where the leaves of this tree represent all possible solutions. In order to find the optimal solution (i.e. one of the leaves) the following steps are followed:

1. Make a series of decisions and find the feasible solution $\hat{x}$ and the objective value $\hat{v}$ and set them as the benchmark.

2. Initially make the root node active.

3. Select an active node i and find the optimal solution x_i and the objective v_i by LP relaxation of $\mathsf{Problem}_i$

4. Check the following cases:

    (a) If $v_i \leq \hat{v}$ then prune node i

    (b) If $v_i > \hat{v}$ and x_i is feasible, then replace $\hat{x}$ with x_i and mark the nodes of a subtree of node i as active.

    (c) If $v_i > \hat{v}$ and x_i is not feasible, then prune node i

5. If there are any active nodes then got to step 3. Otherwise, $\hat{x}$ is the optimal solution.

2.4.5 Branch-and-price algorithm

The column generation technique may not necessarily give an integral solution for an LP relaxation and applying a standard branch-and-bound method to RMP with its existing columns will not guarantee an optimal or feasible solution. It is also possible that after branching, there exists a column that would price out favorably but is not present in the master problem. Therefore, it is necessary to generate columns after branching in order to find the optimal solution. Hence, Barnhart et al. [8] proposed a branch-and-price method that is a hybrid of column generation and the branch-and-bound method. The branch-and-price algorithm begins by formulating the master problem by using Dantzig–Wolfe decomposition [18]. The algorithm then considers a subset of columns to get the restricted master problem (RMP) and follows the following steps:

1. Build the restricted master problem (RMP).

2. Solve relaxation of RMP and get dual values for a subproblem called the pricing problem (PP).

3. Solve PP to find the column with negative reduced cost.

4. Check the following cases:

    (a) If a column is found, add such a column to RMP and go to step 1

    (b) If a column is not found, check the solution:

        i. if integral, terminate

        ii. if not integral, branch and go to step 2

Chapter 3

DL Reasoning with Nominals, Inverse Roles and QCRs

Most DL reasoners supporting $\mathcal{SHOIQ}$ (DL with nominals, inverse roles and QCRs) are based on a tableau calculus. These reasoners implemented a wide range of optimization techniques to make reasoning services more efficient. As the numerical restrictions imposed by nominals are global and nominals also introduce non-determinism (discussed in Section 2.2), we need enhanced optimization techniques to handle them. Moreover, nominals' interaction with inverse roles and QCRs makes it even more complex. In section 3.1, we discuss tableau-based reasoning for DL $\mathcal{SHOIQ}$ in more detail. Section 3.2 provides an overview of some state-of-the-art optimization techniques. The remaining sections of this chapter discuss some existing DL reasoners.

3.1 Tableau-based Reasoning for Expressive DLs

Tableau algorithms have been extended in many ways for handling the semantics of more expressive DL constructs. Some new completion rules were introduced for handling QCRs, transitive roles and nominals.

Nominal Nodes and Blockable Nodes:

When handling nominals, it is crucial for the algorithm to differentiate between nominal nodes and blockable nodes. A node x is a nominal node if its label contains a nominal. Whereas, all other nodes are considered as blockable nodes. This distinction is necessary to preserve the semantics of nominals while applying expansion rules or blocking strategies. For instance, if a nominal node and a blockable node are needed to be merged in order to satisfy an at-most restriction, then for preserving nominal semantics, a blockable node must be merged with a nominal node. A nominal node is added corresponding to every nominal in the domain knowledge.

Blocking for DL $\mathcal{SHOIQ}$:

In the presence of functional restrictions or QCRs along with the inverse roles, subset and equality blocking are not sufficient. Therefore, *pairwise blocking* has been proposed in [39]. In this blocking technique, blocks are established between pairs of nodes connected by the same role: a node x is blocked by a node y if x has ancestors x', y and y', and the following conditions are satisfied:

1. x has a predecessor x' and y has a predecessor y',
2. x, y and all the nodes on the path from y to x are blockable,
3. the labels of x, y are equal such that $\mathcal{L}(x) = \mathcal{L}(y)$,
4. the labels of their predecessors x' and y' are equal such that $\mathcal{L}(x') = \mathcal{L}(y')$, and
5. the labels of the edges connecting x' to x and y' to y are equal such that $\mathcal{L}(\langle x', x\rangle) = \mathcal{L}(\langle y', y\rangle)$.

3.1.1 The Expansion Rules for DL $\mathcal{SHOIQ}$:

In the preceding chapter, tableau expansion rules for the fundamental Description Logic ($\mathcal{ALC}$) have been outlined. To accommodate additional constructors, extending $\mathcal{ALC}$ to $\mathcal{SHOIQ}$, the expansion rules need to be augmented. Therefore, the expansion rules depicted in Figure 2.2 are extended with additional rules to handle more expressive DL constructs. Figure 3.1 illustrates these additional expansion rules presented by [41].

The $\forall_+$-Rule handles the transitive roles. The $\geq$-Rule and $\leq$-Rule ensure that at-least and at-most restrictions are satisfied. For example, for a concept $\geq nR.C \in \mathcal{L}(x)$, if there are not n safe R-neighbours of x, then the $\geq$-Rule creates them with C in their label. An R-neighbour y of a node x is *safe* if either x is blockable or if y is not blocked and x is a nominal node. Moreover, the $\geq$-Rule makes these new nodes disjoint in order to prevent them from merging.

On the other hand, if $\leq nR.C \in \mathcal{L}(x)$ and there are more than n R-neighbours of x with C in their label, then the $\leq$-Rule nondeterministically merges R-neighbours of x for satisfying the at-most number allowed by the restriction. The function $\mathsf{Merge}(z, y)$ merges a node z into a node y by

1. adding the label of z to the label of y, i.e., $\mathcal{L}(y) = \mathcal{L}(y) \cup \mathcal{L}(z)$,

2. moving all edges leading to z so that they lead to y. For instance, for all the nodes x such that $(x, z) \in E$

   (a) if $\{(y, x), (x, y)\} \cap E = \emptyset$, then add (x, y) to E and set $\mathcal{L}(x, y) = \mathcal{L}(x, z)$,

   (b) if $(x, y) \in E$, then set $\mathcal{L}(x, y) = \mathcal{L}(x, y) \cup \mathcal{L}(x, z)$,

   (c) if $(y, x) \in E$, then set $\mathcal{L}(y, x) = \mathcal{L}(y, x) \cup \{R^- \mid R \in \mathcal{L}(x, z)\}$,

   (d) remove edge (x, z) from E;

3. moving all edges leading from z to nominal nodes so that they lead from y to the same nominal nodes. For instance, for all nominal nodes x such that $(z, x) \in E$

   (a) if $\{(y, x), (x, y)\} \cap E = \emptyset$, then add (y, x) to E and set $\mathcal{L}(y, x) = \mathcal{L}(z, x)$,

   (b) if $(y, x) \in E$, then set $\mathcal{L}(y, x) = \mathcal{L}(y, x) \cup \mathcal{L}(z, x)$,

   (c) if $(x, y) \in E$, then set $\mathcal{L}(x, y) = \mathcal{L}(x, y) \cup \{R^- \mid R \in \mathcal{L}(z, x)\}$,

$\forall_+$-Rule **if** $\forall S.C \in \mathcal{L}(x)$ and there exist U, R with $R \in N_{R_+}$ and $U \sqsubseteq_* R$, $R \sqsubseteq_* S$, and a node y with $U \in \mathcal{L}(x, y)$ and $\forall R.C \notin \mathcal{L}(y)$
then set $\mathcal{L}(y) = \mathcal{L}(y) \cup \{\forall R.C\}$

$\geq$-Rule **if** $\geq nR.C \in \mathcal{L}(x)$, x is not blocked and there are not n safe R-neighbours $y_1, ..., y_n$ 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 node $y_1, ..., y_n$ and set $\mathcal{L}(y_i) = \{C\}$, $\mathcal{L}(x, y_i) = \{R\}$, and $y_i \neq y_j$ for $1 \leq i < j \leq n$

$\leq$-Rule **if** $\leq nR.C \in \mathcal{L}(x)$, x is not blocked and there are m safe R-neighbours $y_1, ..., y_m$ of x with $C \in \mathcal{L}(y_i)$ and $m \geq n$ for $1 \leq i \leq m$,
and there are two R-neighbours y_i and y_j of x with $C \in \mathcal{L}(y_i) \cap \mathcal{L}(y_j)$ and $y_i \neq y_j$ for $1 \leq i < j \leq m$
then

1. **if** y_i is a nominal node, then $\mathsf{Merge}(y_j, y_i)$
2. **else if** y_j is a nominal node or an ancestor of y_i, then $\mathsf{Merge}(y_i, y_j)$
3. **else** $\mathsf{Merge}(y_j, y_i)$

ch-Rule **if** $\leq nR.C \in \mathcal{L}(x)$, and there is an R-neighbours y of x with $\{C, \neg C\} \cap \mathcal{L}(y) = \emptyset$
then set $\mathcal{L}(y) = \mathcal{L}(y) \cup \{C, \neg C\}$

o-Rule **if** for some $o \in N_o$ there are nodes x, y with $o \in \mathcal{L}(x) \cap \mathcal{L}(y)$, $x \neq y$
then

1. **if** x is an initial node, then $\mathsf{Merge}(y, x)$,
2. **else** $\mathsf{Merge}(x, y)$

NN-Rule **if** $\leq nR.C \in \mathcal{L}(x)$, $o \in \mathcal{L}(x)$ for some $o \in N_o$, and there exists a blockable R-predecessor y of x such that $C \in \mathcal{L}(y)$
then

1. guess m with $1 \leq m \leq n$ and set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{(\leq mR.C)\}$
2. create m new nodes $z_1, ..., z_m$ with $\mathcal{L}(x, z_i) = \{R\}$, $\mathcal{L}(z_i) = \{C, o_i\}$ with $o_i \in N_o$ new in G, and $z_i \neq z_j$ for $1 \leq i < j \leq m$.

$\leq_o$-Rule **if**

1. $\leq nR.C \in \mathcal{L}(x)$, $o \in \mathcal{L}(x)$ for some $o \in N_o$, and there exists a blockable R-neighbour y of x such that $C \in \mathcal{L}(y)$,
2. there exist m nominal R-neighbour $z_1, ..., z_m$ of x such that $C \in \mathcal{L}(z_i)$ and $z_i \neq z_j$ for all $1 \leq i < j \leq m$, and
3. there is a nominal R-neighbour z of x such that $C \in \mathcal{L}(z)$

then $\mathsf{Merge}(y, z)$

Figure 3.1: The tableau expansion rules for handling the semantics of the added constructors extending $\mathcal{ALC}$ to $\mathcal{SHOIQ}$

(d) remove edge (z, x) from E;

4. removing z and blockable sub-tree of z.

In the case when $\leq nR.C \in \mathcal{L}(x)$, it is necessary to know how many R-neighbours of x have C and how many have $\neg C$ in their label. Due to the open world assumption in description logics, the ch-Rule nondeterministically adds C or $\neg C$ in the label of every R-neighbour of x. It ensures the soundness and completeness of the algorithm. However, since this semantic branching is achieved using a nondeterministic way, it can be a major source of inefficiency in most reasoners. For instance, if $\leq nR.C \in \mathcal{L}(x)$ and there are m R-neighbours of x, the application of ch-Rule opens 2^m branches in the search space in order to add C or $\neg C$ to the label of m R-neighbours of x.

As nominals are interpreted as singletons, the o-rule ensures this semantics and immediately merges two nodes having the same nominal in their label. Due to this merging, the blockable part of the graph can be non-tree shaped because a blockable predecessor of a nominal node can be merged with another blockable neighbor.

To avoid this situation, the NN-Rule and the $\leq_o$-Rule are proposed. The termination of the algorithm in the presence of inverse roles, nominal nodes, and number restrictions is guaranteed by fixing the upper bound on the number of nominal nodes and by using standard blocking technique (pairwise blocking) for blockable nodes.

The NN-Rule guesses the exact number of new nominal nodes, for example if $\leq nR.C \in \mathcal{L}(x)$ where x is a nominal node, the NN-Rule nondeterministically creates m new nominal nodes with $1 \leq m \leq n$ and C in their label. It also ensures that all of these m new nominal nodes are pairwise disjoint, preventing them from being merged with each other.

After that, the $\leq_o$-rule merges the blockable R-neighbour of x with one of the nominal nodes for satisfying the at-most restriction $\leq nR.C \in \mathcal{L}(x)$. Therefore, the NN-Rule and the $\leq_o$-Rule together ensure that the completion graph for the blockable part remains tree-shaped.

3.2 Optimization Techniques

DL reasoners with naive implementations of tableau decision procedures often exhibit poor performance in practice, as the nondeterministic expansion rules can create a very large search space. Therefore, the reasoners are equipped with a set of optimization techniques in order to achieve a reasonable run-time behaviour on real-world problems.

The optimization techniques are mostly categorized as preprocessing optimizations and consistency checking optimizations.

3.2.1 Preprocessing Optimizations

Preprocessing optimization techniques are used to rewrite and reformulate parts of the knowledge base in order to improve reasoning performance. In the following sections, we discuss some most common preprocessing optimizations.

3.2.1.1 Lexical normalization and simplification

In order to detect a clash quickly, concept expressions are transformed into their lexically equivalent expressions. For example, the lexically normalized form of $\exists R.C$ is $\neg\forall R.\neg C$. If a node x has a label $\mathcal{L}(x) = \{\exists R.(A \sqcap \neg B \sqcap C), \forall R.(\neg A \sqcup B \sqcup \neg C)\}$, where A, B and C are concept names. The algorithm creates an R-successor y of x and adds concepts A, B, C $\neg A$, $\neg B$ and $\neg D$ in a label of y in order to detect the inconsistency. On the other hand, by representing $\exists R.(A \sqcap \neg B \sqcap C)$ as $\neg\forall R.(\neg A \sqcup B \sqcup \neg C)$, the algorithm can detect a clash quickly in the label of x. This type of lexical normalization technique is widely used along with other simplification techniques that enable faster clash detection. Simplification helps in eliminating redundancy. Figure 3.2 shows some normalization and simplification rules [43].

Concept Expression	Normalization	Concept Expression	Simplification
$\bot$	$\neg\top$	$\forall R.\top$	$\top$
$\exists R.C$	$\neg\forall R.\neg C$	$C \sqcap \top$	C
$C \sqcup D$	$\neg(\neg C \sqcap \neg D)$	$C \sqcap \neg\top$	$\neg\top$
$\neg\neg C$	C	$C \sqcap \neg C$	$\neg\top$

Figure 3.2: Lexical normalization and simplification

General concept inclusion axioms	Primitive definition axioms
$C_1 \sqcap C_2 \sqsubseteq D$	$C_1 \sqsubseteq D \sqcup \neg C_2$
$C_1 \sqcup C_2 \sqsubseteq D$	$C_1 \sqsubseteq D$ and $C_2 \sqsubseteq D$
$C \sqsubseteq D_1 \sqcap D_2$	$C \sqsubseteq D_1$ and $C \sqsubseteq D_2$

Figure 3.3: Axiom equivalences used in absorption

3.2.1.2 Absorption

Absorption is an optimization technique that tries to eliminate general concept inclusion (GCI) axioms in order to reduce non-determinism.

Concept Absorption

The idea of the concept absorption is presented in [43]. The basic idea is that a GCI axiom of the form $C \sqsubseteq D$, where C is not a concept name, can be absorbed into an equivalent primitive definition axiom $A \sqsubseteq D'$, where A is a concept name and D' is a concept expression. Some axiom equivalences are shown in Figure 3.3. Similarly, if there is an existing primitive definition axiom $A \sqsubseteq C'$, then that axiom could be merged with $A \sqsubseteq D'$. Then this would become $A \sqsubseteq C' \sqcap D'$.

Role Absorption

The domain and range constraints on roles are typically supported by ontology languages. The domain constraint $\mathsf{Domain}(R, C)$ defines that if an individual x is an R-neighbour of y, then x must be an instance of concept C (that is the domain of R). Whereas, the range constraint $\mathsf{Range}(R, C)$ defines that if an individual x is an R-neighbour of y, then y must be an instance of concept C (that is the domain of R).

$domain$-**Rule**	**if** there exists $\mathsf{Domain}(R, C)$, x is not blocked, and there is an R-neighbour y of x with $C \notin \mathcal{L}(x)$ **then** set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C\}$
$range$-**Rule**	**if** there exists $\mathsf{Range}(R, C)$, x is not blocked, and there is an R-neighbour y of x with $C \notin \mathcal{L}(y)$ **then** set $\mathcal{L}(y) = \mathcal{L}(y) \cup \{C\}$

Figure 3.4: Basic Role Absorption Rules

These domain and range constraints can be transformed into the GCIs $\exists R.\top \sqsubseteq C$ and $\top \sqsubseteq \forall R.C$ respectively. As discussed earlier, GCIs are the one of the main sources of inefficiency in the reasoning services, these constraints are handled by using the role absorption technique presented in [73]. They introduced two types of role absorptions: basic and extended role absorptions.

In the *basic role absorption*, the domain and range constraints are handled by using two new rules as shown in Figure 3.4. In the case of $\mathsf{Domain}(R, C)$, if there exists a node x with an R-neighbour y and $C \notin \mathcal{L}(x)$, then the algorithm will add C to the label of x. Similarly, in case of $\mathsf{Range}(R, C)$, if there exists a node x with an R-neighbour y and $C \notin \mathcal{L}(y)$, then the algorithm will add C to the label of y.

The *extended role absorption* deals with a wider range of axioms. They used rewriting techniques similar to the ones used in concept absorption, for example, an axiom $\exists R.C \sqsubseteq D$ can be rewritten as $\exists R.\top \sqsubseteq D \sqcup \neg(\exists R.C)$ which can be absorbed into a domain constraint $\mathsf{Domain}(R, D \sqcup \neg(\exists R.C))$. Similarly, an axiom $D \sqsubseteq \forall R.C$ can be rewritten as $\exists R.\top \sqsubseteq \neg D \sqcup \neg(\exists R.\neg C)$ which can be absorbed into a domain constraint $\mathsf{Domain}(R, \neg D \sqcup \neg(\exists R.\neg C))$. After that, the basic role absorption technique rules can be used to handle these constraints.

3.2.1.3 Nominal Absorption

Nominals can be defined with the *hasValue* and the *oneOf* constructors (as discussed in the previous chapter). The idea of absorption was extended for both of these constructors and presented in [67]. These absorption techniques are discussed below.

$\equiv_{nom}$-**Rule**	**if** $A \equiv \{a_1, ..., a_n\}$ with $a_1, ..., a_n \in N_o$, atomic concept A, **then** replace it with $A \sqsubseteq \{a_1, ..., a_n\}$ and $a_1 : A, ..., a_n : A$
$\exists_{nom}$-**Rule**	**if** $\exists R.\{o\} \sqsubseteq A$ with with $o \in N_o$ **then** replace it with $\{o\} \sqsubseteq \forall R^{-}.A$

Figure 3.5: Nominal Absorption Rules for tableau algorithms

OneOf Absorption

Nominals can be used for defining concepts by finite enumeration of its elements, e.g., we can define the major rose colours as:

$$RoseColour \equiv \{red, white, yellow, pink\} \tag{13}$$

Nominal absorption transforms these definitions into a primitive definition[1] and a set of Abox assertions. For example, Axiom (13) is logically equivalent to a Tbox Axiom and a set of Abox Assertions in (14).

$$\begin{cases} RoseColour \sqsubseteq \{red, white, yellow, pink\}, \text{ and} \\ red : RoseColour \text{ and ... and } pink : RoseColour \end{cases} \tag{14}$$

HasValue Absorption

Nominals are also used for defining concepts in terms of existential restrictions on a nominal, e.g., we can define that the *RedRose* is a type of *Rose* that has *red* colour:

$$RedRose \equiv Rose \sqcap \exists hasColour.\{red\} \tag{15}$$

[1] $A \sqsubseteq B$ is a primitive concept definition when the left-hand side is an atomic concept.

Considering that there are other inclusion axioms in the ontology with the concept $RedRose$ in its left-hand side, Axiom (15) is transformed into Axiom (16) and Axiom (17).

$$RedRose \sqsubseteq Rose \sqcap \exists hasColour.\{red\} \tag{16}$$

$$Rose \sqcap \exists hasColour.\{red\} \sqsubseteq RedRose \tag{17}$$

Therefore, we are again left with the GCI axiom (i.e., Axiom 17). However, nominal absorption transforms Axiom (17) into Axiom (18), which is equivalent to an Abox assertion, as proved in [67].

$$\{red\} \sqsubseteq \forall hasColour^{-}.(RedRose \sqcup \neg Rose) \tag{18}$$

These two rules are defined in Figure 3.5.

3.2.2 Consistency checking optimizations

For consistency checking, the algorithm needs to construct the model (i.e., completion graph). To make this task more efficient, some optimization techniques are employed by many reasoners. We will briefly explain some of the most relevant techniques that significantly improve reasoning performance for real-world ontologies.

3.2.2.1 Lazy Unfolding

As discussed in the previous chapter, Tbox axioms are mostly in the form $C \sqsubseteq D$ (called general concept inclusion (GCI) axioms) or $A \equiv D$ (called definitional axioms). A Tbox that contains GCIs is called *general*. A Tbox can be internalized into a concept that is added to each node label in order to ensure that each node of the completion graph satisfies all axioms of the

$\sqsubseteq_1$**-Rule**	**if** $A \in \mathcal{L}(x)$, $A \sqsubseteq B$ with atomic concept A, $B \notin \mathcal{L}(x)$, and x is not blocked, **then** set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{B\}$
$\sqsubseteq_2$**-Rule**	**if** $\{A_1, A_2\} \subseteq \mathcal{L}(x)$, $A_1 \sqcap A_2 \sqsubseteq B$ with A_i atomic concepts, $B \notin \mathcal{L}(x)$, and x is not blocked, **then** set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{B\}$

Figure 3.6: Lazy unfolding rules for tableau algorithms

Tbox. For example, if a Tbox $\mathcal{T}$ contains the following axioms:

$$A \sqsubseteq \exists R.(C \sqcap D) \tag{19}$$

$$A \sqcap B \sqsubseteq \exists R.C \tag{20}$$

then we can internalize them into a single axiom $\top \sqsubseteq C_{\mathcal{T}}$ such that $C_{\mathcal{T}} := \bigsqcap_{C \sqsubseteq D \in \mathcal{T}} nnf(\neg C \sqcup D)$, where nnf transforms a given concept expression to its negation normal form. Therefore, here

$$C_{\mathcal{T}} := ((\neg A \sqcup \exists R.(C \sqcap D) \sqcap (\neg A \sqcup \neg B \sqcup \exists R.C))$$

In each node label we add axiom $\top \sqsubseteq C_{\mathcal{T}}$. Since it introduces a large number of disjunctions in each node label, we possibly require several nondeterministic choices and backtracking in case of clashes. Therefore, the idea of lazy unfolding was first introduced in [4] and it was further refined for general Tboxes in [38].

Lazy unfolding rules, as shown in Figure 3.6, are introduced to reduce the number of GCIs in the Tbox. For Axiom (19), where A is an atomic concept, the $\sqsubseteq_1$**-Rule** adds $\exists R.(C \sqcap D)$ in a node label if A is satisfied at that node. Similarly, for Axiom (20), the $\sqsubseteq_2$**-Rule** adds $\exists R.C$ in a node label if both A and B exist at that node. Therefore, we do not need to internalize axioms of the form $A \sqsubseteq B$ or $A_1 \sqcap A_2 \sqsubseteq B$ and that significantly reduces the non-determinism.

Moreover, instead of fully unfolding the concept expressions in the beginning, lazy unfolding delays them until it is required. This way a lot of unnecessary unfolding can be avoided. This method is more efficient, especially in case of large and complex concept expressions. For example, in order to test the satisfiability of an expression $\exists R.C \sqcap \forall R.\neg C$, where C is a concept expression, the unfolding of C can be delayed until the algorithm creates an R-successor y with $\mathcal{L}(y) = \{C, \neg C\}$. The algorithm easily detects a contradiction without unfolding C which saves a lot of wasted work.

3.2.2.2 Dependency Directed Backtracking

Dependency directed backtracking (DDB) is a widely used optimization technique that avoids unproductive backtracking search by preventing the evaluation of irrelevant nondeterministic alternatives.

For example, suppose there is a node x with a label

$$\mathcal{L}(x) = \{\leq 1R.\top \sqcup \forall R.\neg B, C_1 \sqcup D_1, ..., C_n \sqcup D_n, \geq 2R.(A \sqcap B)\}$$

and the concepts in the label of x are processed in the same order as shown above. Therefore, from the first disjunction $\leq 1R.\top \sqcup \forall R.\neg B$, the algorithm nondeterministically selects $\leq 1R.\top$. After that, the algorithm processes the remaining n disjunctions $C_1 \sqcup D_1, ..., C_n \sqcup D_n$ and then in order to satisfy $\geq 2R.(A \sqcap B)$, it tries to create 2 R-successors of x. But this results in a clash with the disjunct $\leq 1R.\top$, chosen from the first disjunction. Now with simple backtracking (also known as *thrashing*), the algorithm will also try to evaluate all irrelevant disjunctions $C_1 \sqcup D_1, ..., C_n \sqcup D_n$. This problem is handled by adapting the dependency directed backtracking technique called *backjumping* [43].

The idea is to label each concept in the completion graph with a dependency set. As the initial completion graph is deterministic, the dependency set of each concept is initialized with an empty set. Afterwards, when a new concept is added by an expansion rule application, its dependency set becomes a union of the dependency sets of the concepts that cause the application of that rule. For example, a concept $\leq 1R.\top \sqcup \forall R.\neg B$ has a dependency set $\mathcal{D}_1 = \{\emptyset\}$, then after application of the $\sqcup$-Rule, a concept $\leq 1R.\top$ is added to the label with a dependency set $\mathcal{D}_2 = \{\mathcal{D}_1 \cup 1\}$, here 1 is a dependency that is set for disjunction $\leq 1R.\top \sqcup \forall R.\neg B$. Similarly, the chosen concepts from $C_1 \sqcup D_1, ..., C_n \sqcup D_n$ are also labelled with the dependency sets. Now, in the case of a clash, the algorithm easily identifies the relevant nondeterministic decision and jumps back directly to that decision, choosing the other option if available. For example, here a concept $\geq 2R.(A \sqcap B)$ has a dependency set $\mathcal{D}_k = \{\emptyset\}$, and when the algorithm tries to create 2 R-successors of x with A and B in their label, it detects a clash with $\leq 1R.\top$, having a dependency set $\mathcal{D}_2$. Therefore, the resulting clash set is $\mathcal{D}_l = \{\mathcal{D}_2 \cup \mathcal{D}_k\}$. Now instead of exploring the irrelevant disjunctions $C_1 \sqcup D_1, ..., C_n \sqcup D_n$, the algorithm directly jumps back to a disjunction $\leq 1R.\top \sqcup \forall R.\neg B$ and chooses $\forall R.\neg B$ which also results in a clash. Since there is no other option left the algorithm identifies this label as unsatisfiable.

3.2.2.3 Semantic Branching

Standard tableaux algorithms handle disjunction by using syntactic branching which is not very efficient. In syntactic branching, each disjunct is added to search the different models. If a disjunct is unsatisfiable, then there is nothing to prevent the algorithm from repeatedly adding that disjunct in different branches. This problem is handled by using the semantic branching technique [43]. Semantic branching can optimize the handling of nondeterministic alternatives by adding the negation of an unsatisfiable disjunct. For example, from a disjunction $A \sqcup B$, if a disjunct A is evaluated and found unsatisfiable, then $A \sqcup (\neg A \sqcap B)$ (that is semantically equivalent with $(A \sqcup B)$) would be added to a node label to check the satisfiability. Therefore, this would prevent the algorithm from reevaluation of an unsatisfiable disjunct in different branches. For example, now if a disjunction $A \sqcup C$ is added to the label of the same node, then the algorithm would deterministically expand this disjunction by adding C to the node label.

However, semantic branching causes some additional overhead. For example, if $\neg A$ is a very large or complex concept, then its expansion might require applying many (potentially nondeterministic) rules. Nevertheless, semantic branching is still considered more efficient than syntactic branching in practice.

3.2.2.4 Boolean constant propagation (BCP)

Boolean constraint propagation (BCP) [28] is a simplification technique that is useful for the reduction of the search space resulting from the application of nondeterministic expansion rules. Before the $\sqcup$-Rule application, BCP deterministically expands disjunctions to the label of a node by adding a disjunct if the negations of all other disjuncts are satisfied at the node. For example, if a node label contains the concepts $\{A, (\neg A \sqcup \neg B), (\neg A \sqcup B \sqcup C)\}$, then BCP deterministically expands the disjunction $\neg A \sqcup \neg B$ and adds $\neg B$. After that for the disjunction $\neg A \sqcup B \sqcup C$, BCP deterministically adds C to the label. BCP can also be used with syntactic branching, but it is more effective if used with semantic branching [1].

3.2.2.5 Caching

It is possible that different nodes have identical labels in several completion graphs. This results in repeated expansion of these identical concepts that can be avoided by using caching [43]. The basic idea behind this technique is to store the labels of blockable nodes of complete and clash-free completion graphs in a cache. If the algorithm encounters a node with an identical label to one stored in the cache, it would block the expansion of that node, as it already knows that it is satisfiable. However, this caching technique may not work properly in the presence of more expressive DL constructs. Therefore, this technique is extended for inverse roles and nominals in [53, 20, 34, 32, 70].

3.3 (Hyper-)Tableau Reasoners

Most state-of-the-art reasoners, such as RacerPro [36], Pellet [57], Fact++ [74], HermiT [66], have implemented traditional tableau algorithms.

RacerPro [36] is a tableau-based reasoner for DL $\mathcal{SHIQ}$ and it was the first Abox reasoner for a very expressive logic. RacerPro also implemented many standard optimization techniques for Tbox and Abox reasoning. It was the first highly optimized reasoner that combined tableau-based reasoning with algebraic reasoning [37]. However, it does not handle nominals.

Pellet [57] supports all OWL DL constructs including nominals. It parses an OWL ontology into RDF triples, which are then converted to Tbox axioms and Abox assertions. It also incorporates various optimization techniques such as absorption, lazy unfolding, dependency directed backtracking, semantic branching and early blocking strategies (detailed descriptions of these optimizations can be found in [1]).

Fact++ [74] uses a *ToDo list* architecture to control the application of the expansion rules. The ToDo list is implemented as priority queues in which it sorts entries in a specified order and returns the first entry from the list. This ordering has a huge impact on reasoning performance and is good for more complex tableaux algorithms. Fact++ includes some preprocessing optimizations, such as: 1) Lexical normalization, simplification and synonym replacement for early clash detection. 2) GCI absorption that tries to eliminate GCIs for reducing nondeterminism. 3) Told cycle elimination for performance improvement. Fact++ also includes dependency-directed backtracking, boolean constraint propagation and semantic branching.

Tableau reasoners try to construct a model for a knowledge base in order to perform consistency tests. The two main sources of inefficiency in model construction are:

1. A large number of possible models which tableau reasoners must possibly explore in order to show inconsistency/unsatisfiability.

2. These models can be extremely large even for small ontologies.

Both of these problems, non-determinism and large model size, are tried to be addressed in HermiT [66] by using a hyper-tableau calculus. This calculus uses some additional absorption techniques and tries to reduce the number of possible models. HermiT also limits the model size by using the *anywhere blocking* strategy which improves reasoning performance on many complex and difficult ontologies. They introduced three rules for handling nominals: the o-rule, NN-rule and $\leq_o$-rule (shown in Figure) [41]. The NN-rule application can be highly nondeterministic and it can increase the model size. Therefore, in order to handle this inefficiency they tried to optimize the NN-rule and proposed the NI-rule in [52]. The NI-rule limits the number of new nominal nodes that can be introduced to satisfy a number restriction. However, it is still nondeterministic in choosing nominal nodes for merging.

Although these reasoners have implemented many optimization techniques, they still prove to be highly inefficient in handling a large number of nominals and number restrictions.

3.4 Consequence-based Reasoners

CB methods are considered more efficient than tableau for lightweight DLs. However, CB reasoning algorithms are also extended to more expressive DLs such as Horn-$\mathcal{SHIQ}$ [45], Horn-$\mathcal{SROIQ}$ [56], $\mathcal{ELQ}$ [75] and $\mathcal{SHOQ}$ [77]. CB algorithms use a graph structure similar to a (hyper-)tableau; however, they avoid constructing very large model representations like resolution-based algorithms [2, 45, 46, 10] by deriving logical consequences.

The first consequence-based calculus for deciding concept subsumption in the DL $\mathcal{SRIQ}$ is proposed in [10, 9]. They have implemented their calculus in a new reasoner called Sequoia.

CB reasoning algorithm for DL $\mathcal{SHOI}$ is proposed in [14] which supports nominals and inverse roles. This calculus is extended in [15] for supporting number restrictions. The calculus for the logic $\mathcal{SROIQ}$ has been implemented as an extension of Sequoia [16].

The saturation-based algebraic reasoners for DL $\mathcal{ELQ}$ [75] combine saturation rules with algebraic reasoning based on Integer Linear Programming (ILP). Similarly, the CB $\mathcal{SHOQ}$ [77] combines CB reasoning with algebraic reasoning based on ILP. However, the former does not support nominals and inverse roles, and the latter does not handle inverse roles.

3.5 Hybrid of Tableau reasoning and Consequence-based procedures

Konclude [72] is a highly efficient reasoner for DL $\mathcal{SROIQV}$ which extends $\mathcal{SROIQ}$ with nominal schemas [49]. Konclude is primarily based on tableau calculus but also incorporates consequence-based reasoning [69]. It uses several new optimization techniques along with standard optimizations.

There are three main stages for handling reasoning requests: parsing, loading, and reasoning. The parsing stage includes parsing of ontology and queries. In the loading stage, it loads an ontology, builds its internal representation and preprocesses axioms. The reasoning stage is the main stage that includes all major reasoning tasks such as consistency tests and ontology classification.

As consequence-based reasoning is more efficient for lightweight DLs, Konclude utilizes a saturation procedure to process only those parts of knowledge bases that can easily and efficiently be handled. Afterwards, it detects the parts that are not completely handled by the saturation and uses tableau procedures to process them. For an easy coupling with tableau procedures, the saturation procedure works on the same data structures and knowledge bases as the tableau algorithms. Therefore, this makes it easy to use the node labels from the saturation directly in the tableau algorithm.

Konclude also supports parallel processing at several levels of its processing architecture. Parallelization is applied to the tableau procedure and for deriving consequences for the individuals. However, it is not applied to the saturation of Tbox related parts and other preprocessing and precomputation steps.

Konclude incorporated many standard optimizations including lazy unfolding, semantic branching, dependency directed backtracking, boolean constant propagation, anywhere blocking, and caching of satisfiability status. It also implemented some new optimization techniques such as absorption-based handling of nominal schemas [68], pool-based merging [71], and known/possible set classification and realization approach [31].

3.6 Algebraic Reasoners

A hybrid tableau algorithm for DL $\mathcal{SHQ}$ is proposed in [26] which extends the basic description logic $\mathcal{ALC}$ with role hierarchies, transitive roles, and qualified number restrictions. The algorithm uses an atomic decomposition technique to encode number restrictions into a set of inequalities. Then this set of inequalities is processed by an inequality solver based on integer linear programming. The inequality solver tries to find a minimal non-negative integer solution satisfying the inequalities or returns a clash if no solution exists. Since all numerical restrictions imposed by at-least and at-most restrictions are satisfied by this solution, the algorithm creates only one *proxy individual,* which represents a set of role fillers. Moreover, the algorithm does not satisfy any at-least restriction by violating any at-most restriction because all the information about arithmetic expressions is collected before creating any role filler. Therefore, no extra mechanism is required for merging role fillers. The algorithm proceeds in three steps: (i) preprocessing of Abox and Tbox, (ii) atomic decomposition of role fillers, and (iii) applying completion rules.

The algebraic reasoner for DL $\mathcal{SHIQ}$, proposed in [58], handles the interaction between inverse roles and number restrictions by driving inequalities. These inequalities are then solved by using the Simplex method [17]. The semantics of inverse roles are preserved as numerical restrictions. For example, there is a node x with $\geq 1R.C$ and an R-successor y with $\mathcal{L}(y) \leftarrow C$ is created to satisfy this at least restriction. Since $R \in \mathcal{L}(x, y)$, an implied back edge $R^{-} \in \mathcal{L}(y, x)$ is also imposed. Therefore, to preserve the semantics of this edge, the algorithm adds a set of number restrictions $\{\geq 1R^{-}_{yx}, \leq 1R^{-}_{yx}\}$ to the label of node y.

The interaction between nominals and number restrictions makes reasoning more complex. For example, consider a small ontology that defines the European Union (EU) member states using 28 mutually disjoint nominals, each representing a state *{United Kingdom, Germany, Belgium, France, Italy, Luxembourg, Netherlands, Bulgaria, Croatia, Cyprus, Czech Republic, Malta, Poland, Portugal, Romania, Ireland, Latvia, Lithuania, Greece, Spain, Slovakia, Finland, Hungary, Sweden, Denmark, Estonia, Slovakia, and Austria}*. Additionally, there are Future EU members that need to be related to at least 30 different EU member states (adapted from [22]). We can define it as:

$$\mathsf{EUMembers} \equiv \{\mathsf{UnitedKingdom}, \mathsf{Germany}, ..., \mathsf{Austria}\}$$

$$\mathsf{FutureEU} \sqsubseteq \geq 30\mathsf{hasMember}.\mathsf{EUMembers}$$

Since a standard tableau algorithm first satisfies all at-least restrictions, it will create 30 anonymous mutually distinct individuals of EUMembers. After that it will try to merge them in order to satisfy the implicit numeric restriction imposed by nominals which states that we can only have at most 28 EUMembers. This merging is highly nondeterministic because the reasoner tries all possible choices of merging 30 anonymous individuals with 28 EUMembers which causes a significant performance degradation. This problem is successfully addressed using algebraic reasoning in [23, 22]. Since nominals carry cardinality restrictions, their semantics are also preserved as numerical restrictions. Based on the nominals semantics $\sharp\{o\}^{\mathcal{I}} = 1$ for each $o \in N_o$, the cardinality of a partition element with a nominal o can only be equal to 1. This semantic is encoded into inequalities using a set ξ such that $\xi(\{o\}, \geq, 1)$ and $\xi(\{o\}, \leq, 1)$. These algebraic reasoners perform very efficiently in handling huge values in number restrictions.

Integer programming has also been used in finite model reasoning [59, 50, 60].

However, to the best of our knowledge, no algebraic calculus can handle DLs supporting nominals, inverse roles and number restrictions simultaneously.

Chapter 4

An Algebraic Tableau Calculus for $\mathcal{SHOIQ}$

As discussed earlier, the numerical restrictions imposed by nominals are global and their interaction with inverse roles and QCRs makes reasoning even more complex. Therefore, we need some optimization techniques that can handle these numerical restrictions more efficiently. This chapter demonstrates how algebraic reasoning is used for handling numerical restrictions. The algebraic reasoning incorporates a branch-and-price technique with tableau-based reasoning. The algebraic tableau algorithm either computes an optimal solution or detects infeasibility. This hybrid calculus reduces reasoning complexity that is caused by the interaction between QCRs, nominals and inverse roles while maintaining soundness, completeness and termination. Section 4.3, provides an overview of the reasoning process. The expansion rules for $\mathcal{SHOIQ}$ are presented in Section 4.3.2. In Section 4.3.1, we demonstrate how to use column generation and branch-and-price technique for generating inequalities. Section 4.4 provides the proof of correctness of the proposed hybrid calculus.

4.1 A Tableau for $\mathcal{SHOIQ}$

We define a tableau for DL $\mathcal{SHOIQ}$ based on the standard tableau for DL $\mathcal{SHOIQ}$, introduced in [41]. For convenience, we assume that all concept descriptions are in negation normal form, i.e., the negation sign only appears in front of concept names (atomic concepts). A label is assigned to each node in the completion graph (CG) and that label is a subset of possible concept expressions. We define $clos(C)$ as the closure of a concept expression C.

Definition 26. (Closure) The closure $clos(C)$ for a concept expression C is the smallest set of concepts such that:

- $C \in clos(C)$,
- $\neg D \in clos(C) \quad \Longrightarrow \quad D \in clos(C)$,
- $(B \sqcup D) \in clos(C)$ *or* $(B \sqcap D) \in clos(C) \quad \Longrightarrow \quad B \in clos(C), D \in clos(C)$,
- $\forall R.D \in clos(C) \quad \Longrightarrow \quad D \in clos(C)$
- $\exists R.D \in clos(C) \quad \Longrightarrow \quad D \in clos(C)$
- $\bowtie nR.D \in clos(C) \quad \Longrightarrow \quad D \in clos(C)$

where $B, D \in N$, $R \in N_R$ and $\bowtie nR.D$ represents $\geq nR.D$ or $\leq nR.D$. For a Tbox $\mathcal{T}$, if ($C \sqsubseteq D \in \mathcal{T}$) or ($C \equiv D \in \mathcal{T}$) then $clos(C) \subseteq clos(\mathcal{T})$ and $clos(D) \subseteq clos(\mathcal{T})$. Similarly, for an Abox $\mathcal{A}$, if ($a : C \in \mathcal{A}$) then $clos(C) \subseteq clos(\mathcal{A})$.

Definition 27. ($\mathcal{SHOIQ}$ Tableau) If $(\mathcal{T}, \mathcal{R})$ is a $\mathcal{SHOIQ}$ knowledge base w.r.t. Tbox $\mathcal{T}$ and role hierarchy $\mathcal{R}$, a tableau T for $(\mathcal{T}, \mathcal{R})$ is defined as a triple $(\mathbf{S}, \mathcal{L}, \mathcal{E})$ such that: $\mathbf{S}$ is a set of individuals, $\mathcal{L} : \mathbf{S} \longrightarrow 2^{clos(\mathcal{T})}$ maps each individual to a set of concepts, and $\mathcal{E} : N_R \longrightarrow 2^{\mathbf{S} \times \mathbf{S}}$ maps each role in N_R to a set of pairs of individuals in $\mathbf{S}$. For all $x, y \in \mathbf{S}$, $C, C_1, C_2 \in clos(\mathcal{T})$, and $U, R, S \in N_R$, the following properties must always hold:

(P1) if $C \in \mathcal{L}(x)$, then $\neg C \notin \mathcal{L}(x)$,

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

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

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

(P5) if $\exists R.C \in \mathcal{L}(x)$, then there is some $y \in \mathbf{S}$ such that $\langle x, y\rangle \in \mathcal{E}(R)$ and $C \in \mathcal{L}(y)$,

(P6) if $\forall S.C \in \mathcal{L}(x)$ and $\langle x, y\rangle \in \mathcal{E}(U)$ for some U, R with $U \sqsubseteq_* R$, $R \sqsubseteq_* S$, and $R \in N_{R_+}$, then $\forall R.C \in \mathcal{L}(y)$,

(P7) if $\geq nR.C \in \mathcal{L}(x)$, then $\sharp R^T(x, C) \geq n$,

(P8) if $\leq nR.C \in \mathcal{L}(x)$, then $\sharp R^T(x, C) \leq n$,

(P9) if $\leq nR.C \in \mathcal{L}(x)$ and $\langle x, y\rangle \in \mathcal{E}(R)$, then $\{C, \neg C\} \cap \mathcal{L}(y) \neq \emptyset$,

(P10) if $\langle x, y\rangle \in \mathcal{E}(R)$ and $R \sqsubseteq_* S \in \mathcal{R}$, then $\langle x, y\rangle \in \mathcal{E}(S)$,

(P11) $\langle x, y\rangle \in \mathcal{E}(R)$ iff $\langle y, x\rangle \in \mathcal{E}(\mathsf{Inv}(R))$,

(P12) if $o \in \mathcal{L}(x) \cap \mathcal{L}(y)$ for some $o \in N_o$, then $x = y$, and

(P13) for each $o \in N_o$ occurring in $\mathcal{T}$, $\sharp\{x \in \mathbf{S} \mid o \in \mathcal{L}(x)\} = 1$.

where $R^T(x, C) = \{y \in \mathbf{S} \mid \langle x, y\rangle \in \mathcal{E}(R) \text{ and } C \in \mathcal{L}(y)\}$

Lemma 28. *A $\mathcal{SHOIQ}$ knowledge base $(\mathcal{T}, \mathcal{R})$ is consistent iff there exists a tableau for $(\mathcal{T}, \mathcal{R})$.*

Proof. The proof is analogous to the one presented in [40, 41]. If $\mathrm{T} = (\mathbf{S}, \mathcal{L}, \mathcal{E})$ is a tableau for $(\mathcal{T}, \mathcal{R})$, a model $\mathcal{I}$ of $(\mathcal{T}, \mathcal{R})$ can be defined as:

$$\begin{aligned}
\text{for a set of individuals:}\quad & \Delta^{\mathcal{I}} := \mathbf{S} \\
\text{for concept names } C \in \mathit{clos}(\mathcal{T})\text{:}\quad & C^{\mathcal{I}} := \{x \mid C \in \mathcal{L}(x)\} \\
\text{for role names } R \in \mathcal{R}\text{:}\quad & R^{\mathcal{I}} := \begin{cases} \mathcal{E}(R)^+ & \text{if } R \in N_{R_+} \\ \mathcal{E}(R) \cup \bigcup_{U \sqsubseteq_* R, U \neq R} U^{\mathcal{I}} & \text{otherwise} \end{cases}
\end{aligned}$$

where the transitive closure of $\mathcal{E}(R)$ is denoted by $\mathcal{E}(R)^+$.

The non-transitive roles are interpreted while considering those non-transitive roles that have a transitive subrole. From the definition of $R^{\mathcal{I}}$, **(P10)** and **(P11)**, if $\langle x, y\rangle \in S^{\mathcal{I}}$, then either $\langle x, y\rangle \in$

$\mathcal{E}(S)$ or there exists a path $\langle x, x_1\rangle, \langle x_1, x_2\rangle, \ldots, \langle x_n, y\rangle \in \mathcal{E}(R)$ for some R with $R \sqsubseteq_* S \in \mathcal{R}$ and $R \in N_{R_+}$. Hence, $\mathcal{I}$ is a model of $\mathcal{R}$.

Moreover, in order to prove that $\mathcal{I}$ is a model of $\mathcal{T}$, we show that $C \in \mathcal{L}(x) \Rightarrow x \in C^{\mathcal{I}}$ for any $x \in \mathbf{S}$ by induction on the structure of concepts.

- If $C \in \mathcal{L}(x)$, then by definition $x \in C^{\mathcal{I}}$.

- If $\neg C \in \mathcal{L}(x)$, then by **(P1)** $C \notin \mathcal{L}(x)$. Hence, $x \notin C^{\mathcal{I}}$.

- If $C = (C_1 \sqcap C_2)$, then $C \in \mathcal{L}(x)$ and **(P2)** imply $C_1 \in \mathcal{L}(x)$ and $C_2 \in \mathcal{L}(x)$, so by induction $x \in (C_1)^{\mathcal{I}}$ and $x \in (C_2)^{\mathcal{I}}$. Hence, $x \in (C_1 \sqcap C_2)^{\mathcal{I}}$.

- If $C = (C_1 \sqcup C_2)$, then $C \in \mathcal{L}(x)$ and **(P3)** imply $C_1 \in \mathcal{L}(x)$ or $C_2 \in \mathcal{L}(x)$, so by induction $x \in (C_1)^{\mathcal{I}}$ or $x \in (C_2)^{\mathcal{I}}$. Hence, $x \in (C_1 \sqcup C_2)^{\mathcal{T}}$.

- If $C = \exists R.C_1$, then $C \in \mathcal{L}(x)$ and **(P5)** imply that there exists an individual $y \in \mathbf{S}$ such that $\langle x, y\rangle \in \mathcal{E}(R)$ and $C_1 \in \mathcal{L}(y)$. Since by definition $\langle x, y\rangle \in R^{\mathcal{I}}$ and by induction $y \in C_1^{\mathcal{I}}$, $x \in (\exists R.C_1)^{\mathcal{I}}$.

- If $C = \forall S.C_1$ and $C \in \mathcal{L}(x)$, and $y \in \mathbf{S}$ such that $\langle x, y\rangle \in R^{\mathcal{I}}$, then either

  - $\langle x, y\rangle \in \mathcal{E}(R)$, then **(P4)** implies $C_1 \in \mathcal{L}(y)$, or
  - $\langle x, y\rangle \notin \mathcal{E}(R)$, then there exists a path $\langle x, x_1\rangle, \langle x_1, x_2\rangle, \ldots, \langle x_n, y\rangle \in \mathcal{E}(R)$ for some R with $R \sqsubseteq_* S \in \mathcal{R}$ and $R \in N_{R_+}$. **(P6)** implies $\forall S.C_1 \in \mathcal{L}(x_i)$ for all $1 \leq i \leq n$ and, $C_1 \in \mathcal{L}(y)$ also holds because of **(P4)**.

  In both cases, by induction $y \in C_1^{\mathcal{I}}$, hence $x \in (\forall S.C_1)^{\mathcal{I}}$.

- If $C =\geq nR.C_1$, then $C \in \mathcal{L}(x)$ and **(P7)** imply that there are n individuals $y_1, \ldots, y_n$ such that $y_i \in \mathbf{S}$, $y_i \neq y_j$ for $1 \leq i < j \leq n$, $\langle x, y_i\rangle \in \mathcal{E}(R)$ and $C_1 \in \mathcal{L}(y_i)$. Since by definition $\langle x, y_i\rangle \in R^{\mathcal{I}}$ and $\mathcal{E}(R) \subseteq R^{\mathcal{I}}$ and by induction $y_i \in C_1^{\mathcal{I}}$, $x \in (\geq nR.C_1)^{\mathcal{I}}$.

- If $C = \leq nR.C_1$ and R is a simple role such that $\mathcal{E}(R) = R^{\mathcal{I}}$, then $C \in \mathcal{L}(x)$ and **(P8)** imply $\sharp R^T(x, C_1) \leq n$. **(P9)** implies that either $C_1 \in \mathcal{L}(y)$ or $\neg C_1 \in \mathcal{L}(y)$. We can prove $\sharp R^{\mathcal{I}}(x, C_1) \leq \sharp R^T(x, C_1)$ by a proof of contradiction and assume $\sharp R^{\mathcal{I}}(x, C_1) > \sharp R^T(x, C_1)$. This assumption implies that there exists an individual y with $\langle x, y\rangle \in R^{\mathcal{I}}$ with $y \in C_1^{\mathcal{I}}$, however, since $\mathcal{E}(R) = R^{\mathcal{I}}$, $C_1 \notin \mathcal{L}(y)$. Therefore, **(P9)** implies $\neg C_1 \in \mathcal{L}(y)$, which yields $y \in \neg C_1^{\mathcal{I}}$, in contradiction to $y \in C_1^{\mathcal{I}}$.

- Furthermore, **(P12)** and **(P13)** ensure that the nominal semantics are preserved by interpreting them as singletons.

For the converse, if $\mathcal{I}$ is a model of $(\mathcal{T}, \mathcal{R})$, a tableau $\mathrm{T} = (\mathbf{S}, \mathcal{L}, \mathcal{E})$ for $(\mathcal{T}, \mathcal{R})$ can be defined as:

$$\begin{aligned}
&\mathbf{S} = \Delta^{\mathcal{I}} \\
&\mathcal{L}(x) = \{C \in clos(\mathcal{T}) \mid x \in C^{\mathcal{I}}\} \\
&\mathcal{E}(R) = R^{\mathcal{I}}
\end{aligned}$$

In order to demonstrate that T is a tableau for $(\mathcal{T}, \mathcal{R})$, we need to show that T satisfies the properties **(P1)**$-$**(P13)**.

- T satisfies **(P1)**$-$**(P5)** as a direct consequence of the definition of the semantics of $\mathcal{SHOIQ}$-concepts.

- By a proof of contradiction, we show if $x \in (\forall S.C)^{\mathcal{I}}$ and $\langle x, y\rangle \in R^{\mathcal{I}}$ for some R with $R \sqsubseteq_* S \in \mathcal{R}$ and $R \in N_{R_+}$, then $y \in (\forall S.C)^{\mathcal{I}}$. Assume there is some z such that $\langle y, z\rangle \in R^{\mathcal{I}}$ and $z \notin C^{\mathcal{I}}$. Now if $\langle x, y\rangle \in R^{\mathcal{I}}$, $\langle y, z\rangle \in R^{\mathcal{I}}$ and $R \in N_{R_+}$, then $\langle x, z\rangle \in R^{\mathcal{I}}$, which yields $\langle x, z\rangle \in S^{\mathcal{I}}$ and $x \notin (\forall S.C)^{\mathcal{I}}$, in contradiction to $x \in (\forall S.C)^{\mathcal{I}}$. Therefore, T satisfies **(P6)**.

- T satisfies **(P7)**$-$**(P9)** as a direct consequence of the semantics of QCRs.

- Since $\mathcal{I}$ is a model of $\mathcal{R}$, **(P10)** is satisfied.

- T satisfies **(P11)** as a direct consequence of the semantics of inverse roles.

- T satisfies **(P12)**−**(P13)** as a direct consequence of the nominal semantics.

□

4.2 The Algebric Method for $\mathcal{SHOIQ}$

The algebraic method for SHOIQ is used to reduce the satisfiability of concept descriptions involving QCRs and/or nominals to equation-solving problems. A crucial technique for enabling this algebraic reasoning is atomic decomposition (see Section 2.4.1 for details) which facilitates the decomposition of a set of elements into mutually disjoints subsets.

4.2.1 Encoding Numerical Restrictions into Inequalities

The atomic decomposition technique [54] is used to encode numerical restrictions on concepts and role fillers into inequalities. These inequalities are then solved to decide the satisfiability of the numerical restrictions. The existential restrictions are converted into ≥ 1 inequalities. The cardinality of a partition element containing a nominal o is equal to 1 due to the nominal semantics; $\sharp\{o\}^I = 1$ for each nominal $o \in N_o$.

Definition 29. (Decomposition Set) Let $Q = Q_{\geq} \cup Q_{\leq} \cup Q_{\forall} \cup Q_o$ define the decomposition set, where $Q_{\forall}$ contains universal restrictions, $Q_{\geq}$ ($Q_{\leq}$) contains at-least (at-most) restrictions and Q_o contains all related nominals. A partitioning $\mathcal{P}$ that is the power set of Q containing all subsets of Q except the empty set.

Each partition element $e \in \mathcal{P}$ represents the intersection of its elements. Each element $R_q \in Q_{\geq} \cup Q_{\leq} \cup Q_{\forall}$ represents a role $R \in N_R$ and its qualification concept expression q and each element $I_q \in Q_o$ represents a nominal $q \in N_o$. The elements in $Q_{\forall}$ are used in AR to ensure the semantics of universal restrictions. The set of related nominals $Q_o \subseteq N_o$ is defined as $Q_o = \{o \mid o \in clos(q) \wedge R_q \in Q_{\geq} \cup Q_{\leq} \cup Q_{\forall}\}$ where $clos(q)$ is the closure of concept expression q. The atomic decomposition considers all possible ways to decompose Q into sets that are semantically pairwise disjoint.

4.3 The Algebraic Tableau Algorithm for $\mathcal{SHOIQ}$

The algorithm takes a $\mathcal{SHOIQ}$ Tbox $\mathcal{T}$ and its role hierarchy $\mathcal{R}$ as input and tries to create a complete and clash-free completion graph in order to check Tbox consistency.

Definition 30. (Completion Graph) The completion graph for a $\mathcal{SHOIQ}$ Tbox $\mathcal{T}$ is defined as follows:

- Let $G = (V, E, \mathcal{L}, \mathcal{B}, \mathcal{U})$ be a completion graph where V is a set of nodes and E a set of edges. Each node $x \in V$ is labelled with three labels: $\mathcal{L}(x)$, $\mathcal{B}(x)$, and $\mathcal{U}(x)$, and each edge $\langle x, y \rangle \in E$ is labeled with a set of role names $\mathcal{L}(x, y)$.
  - $\mathcal{L}(x)$ denotes a set of concept expressions,
  - $\mathcal{B}(x)$ denotes the neighbours of x that can potentially be reused in order to avoid merging later on. For each node $x \in V$, if $\mathcal{L}(x)$ contains a universal restriction or at-most restriction on role R and there exists an R-neighbour of x, then $\mathcal{B}(x)$ contains a tuple of the form $\langle v, \mathcal{L}(x, v) \rangle$ where $v \in V$ is an R-neighbour of x.
  - $\mathcal{U}(x)$ contains a set of universal restrictions that ensure that the at-most restrictions are not violated. For each node $x \in V$, if $\mathcal{L}(x)$ contains an at-most restriction e.g., $\leq nR.C$, then $\mathcal{U}(x)$ contains a universal restriction $\forall R.(C \sqcup \neg C)$ in order to avoid the violation of the at-most restriction.
- The cardinality of a node v is denoted by $\sharp v$.
- Nodes in G are categorized into two types: nominal nodes and blockable nodes. If a node x has a nominal in its label $\mathcal{L}(x)$ then it is a nominal node; otherwise, it is a blockable node.
- G utilizes proxy nodes (see Definition 31) as representatives for domain elements distributed within the same partition. The concept of proxy nodes was initially introduced in [35].

Definition 31. (Proxy Node) A proxy node is used to represent a partition element $e \in \mathcal{P}$. According to Proposition 32, one proxy node can be used to represent the n individuals in e.

By creating a proxy node x for e in G, one can test the satisfiability of the concepts in e. If x satisfies the concepts, then n nodes can also satisfy them. Since n is decided after considering all inequalities and related axioms, x cannot violate cardinality bounds on role fillers and nominals.

Proposition 32. *Given a completion graph G of a model $\mathcal{I}$ for a Tbox $\mathcal{T}$. Let e be a partition element of $\mathcal{P}$ and according to the solution σ, $\sharp e = n$. It is sufficient to create one proxy node in G in order to represent the n individuals in e.*

For convenience, we assume that all concept descriptions are in negation normal form. We divided our hybrid reasoning into two parts:

1. Tableau Reasoning (TR)

2. Algebraic Reasoning (AR)

In the TR part, the algorithm starts with some preprocessing and reduces all the concept axioms in a Tbox $\mathcal{T}$ to a single axiom $\top \sqsubseteq C_{\mathcal{T}}$ such that $C_{\mathcal{T}} := \bigsqcap_{C \sqsubseteq D \in \mathcal{T}} nnf(\neg C \sqcup D)$, where nnf transforms a given concept expression to its negation normal form. The algorithm checks the consistency of $\mathcal{T}$ by testing the satisfiability of $o \sqsubseteq C_{\mathcal{T}}$ where $o \in N_o$ is a fresh nominal in $\mathcal{T}$, which means that at least $o^{\mathcal{I}} \in {C_{\mathcal{T}}}^{\mathcal{I}}$ and ${C_{\mathcal{T}}}^{\mathcal{I}} \neq \emptyset$. Moreover, since $\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$, every domain element must also satisfy $C_{\mathcal{T}}$. For creating a complete and clash-free completion graph, the algorithm applies expansion rules (see Section 4.3.2).

In the AR part, the algorithm handles all numerical restrictions using *Integer Linear Programming* (ILP). It generates inequalities and solves them using the branch-and-price technique (see Section 4.3.1 for details). We use pairwise blocking [42] due to the presence of inverse roles and QCRs (discussed in the previous chapter, see Section 3.1 for details).

4.3.1 Generating Inequalities

Dantzig and Wolfe [18] proposed a column generation technique (see Section 2.4.3) for solving linear programming (LP) problems, called Dantzig–Wolfe decomposition, where a large LP is decomposed into a master problem and a subproblem (or pricing problem). In case of LP problems with a huge number of variables, column generation works with a small subset of variables and builds a Restricted Master Problem (RMP). The Pricing Problem (PP) generates a new variable with the most reduced cost if added to RMP (see [13, 76] for details). However, column generation may not necessarily yield an integral solution for an LP relaxation, i.e., at least one variable may not have an integer value. Therefore, the branch-and-price method [8] has been used which is a combination of column generation and branch-and-bound technique [19] (see Section 2.4.5). We employ this technique by mapping number restrictions to linear inequality systems using a column generation ILP formulation (see [76] for details). The feasibility test for the linear inequalities can be computed in polynomial time, as shown in [51]. CPLEX[1] has been used to solve our ILP formulation.

4.3.1.1 Branch-and-Price Method

In the following, we use a Tbox $\mathcal{T}$ and its role hierarchy $\mathcal{R}$, a completion graph G, a decomposition set Q and a partitioning $\mathcal{P}$ that is the power set of Q containing all subsets of Q except the empty set. Each partition element $e \in \mathcal{P}$ represents the intersection of its elements.

We decompose our problem into two subproblems:

1. Restricted Master Problem (RMP), and

2. Pricing Problem (PP).

RMP contains a subset of columns and PP computes a column that can maximally reduce the cost of RMP's objective. Whenever a column with negative reduced cost is found, it is added to RMP. Number restrictions are represented in RMP as inequalities, with a restricted set of variables. The flowchart in Figure 4.1 illustrates the whole process.

[1] CPLEX is an optimization software tool for solving linear optimization problems.
https://www.ibm.com/analytics/cplex-optimizer

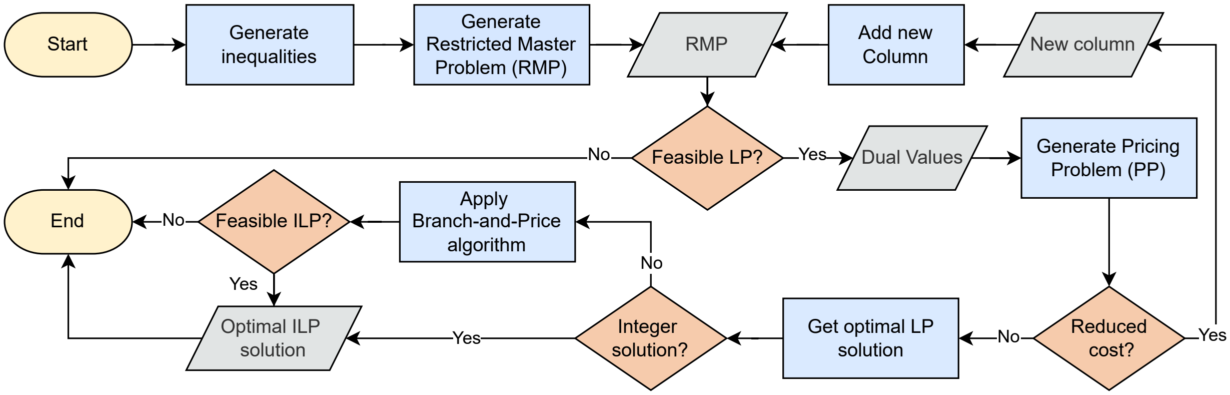


Figure 4.1: Overview of the algebraic reasoning process

Restricted Master Problem

RMP is obtained by considering only variables x_e with $e \in \mathcal{P}'$ and $\mathcal{P}' \subseteq \mathcal{P}$ and relaxing the integrality constraints on the x_e variables. The ILP model associated with the feasibility problem of Q is as follows:

$$\text{Min} \sum_{e \in \mathcal{P}'} cost_e x_e \tag{21}$$

$$\text{Subject to} \quad \sum_{e \in \mathcal{P}'} a_e^{R_q} x_e \geq \underline{\delta}_{R_q} \quad R_q \in Q_{\geq} \tag{22}$$

$$\sum_{e \in \mathcal{P}'} a_e^{R_q} x_e \leq \bar{\delta}_{R_q} \quad R_q \in Q_{\leq} \tag{23}$$

$$\sum_{e \in \mathcal{P}'} a_e^{I_q} x_e = \; 1 \quad I_q \in Q_o \tag{24}$$

$$x_e \in \mathbb{R}^+ \text{ with } e \in \mathcal{P}' \tag{25}$$

$$a_e^{R_q}, a_e^{I_q} \in \{0, 1\} \text{ with } R_q \in Q_{\geq} \cup Q_{\leq},\; I_q \in Q_o \tag{26}$$

where a decision variable x_e represents the elements of the partition element $e \in \mathcal{P}'$. $\underline{\delta}_{R_q}$ ($\bar{\delta}_{R_q}$) is the cardinality of an at-least (at-most) restriction on a role R. The coefficients a_e are associated

Algorithm 4.1 generateRMP$(S_{\geq}, S_{\leq}, S_{\exists}, S_o)$

```
Input: A set $S_{\geq}$ of at-least restrictions, a set $S_{\leq}$ of at-most restrictions, a set $S_{\exists}$ of existential
    restrictions, and a set $S_o$ of related nominals
Output: RMP
$S'_{\exists} \leftarrow \{s'_{\exists} \mid s'_{\exists} \leftarrow$ convert $s_{\exists} \in S_{\exists}$ into $\geq 1$ inequalities $\}$
$S_{\geq} \leftarrow S_{\geq} \cup S'_{\exists}$
RMP $\leftarrow$ fresh model
RMP $\leftarrow$ add initial objective with artificial variables
for all $s_{\geq} \in S_{\geq}$ do
    RMP $\leftarrow$ add constraint for $s_{\geq}$ to handle at-least restrictions
end for all
for all $s_{\leq} \in S_{\leq}$ do
    RMP $\leftarrow$ add constraint for $s_{\leq}$ to handle at-most restrictions
end for all
for all $s_o \in S_o$ do
    RMP $\leftarrow$ add constraint for $s_o$ to handle nominals
end for all
```

with variables x_e and $a_e^{R_q}$ indicates whether an R-neighbour that is an instance of q exists in e. Similarly, $a_e^{I_q}$ indicates whether a nominal q exists in e. The weight $cost_e$ defines the cost of selecting e and it depends on the number of elements e contains. Since we minimize the objective function, $cost_e$ in the objective (21) ensures that only subsets with entailed concepts will be added which are the minimum number of concepts that are needed to satisfy all the axioms. Constraints (22) and (23) encode at-least and at-most restrictions. Constraint (24) encodes numerical restrictions imposed by nominals (i.e., $\sharp\{o\}^I = 1$). Constraint (25) states the integrality condition relaxed from $x_e \in \mathbb{Z}^+$ to $x_e \in \mathbb{R}^+$. Algorithm 4.1 (generateRMP) takes all related nominals, at-least, at-most and existential restrictions as input and generates RMP.

In order to begin the solution process, we need to find an initial set of columns satisfying the Constraints (22) and (24). However, it can be a cumbersome task to find an initial feasible solution. Therefore, initially $\mathcal{P}'$ is empty and RMP contains only artificial variables h to obtain an initial feasible inequality system. Each artificial variable corresponds to an element in $Q_{\geq} \cup Q_{\leq} \cup Q_o$ such that h_{R_q}, $R_q \in Q_{\geq} \cup Q_{\leq}$ and h_{I_q}, $I_q \in Q_o$. An arbitrarily large cost M is associated with every artificial variable. The objective of RMP is defined as the sum of all costs as shown in (27) of the RMP below. Since we minimize the objective function, by considering this large cost M

one can ensure that as the column generation method proceeds, the artificial variables will leave the basis. Therefore, in the case of a feasible set of inequalities, these artificial variables must not exist in the final solution.

$$\text{Min} \sum_{e \in \mathcal{P}'} cost_e x_e + M \sum_{R_q \in Q_\geq} h_{R_q} + M \sum_{R_q \in Q_\leq} h_{R_q} + M \sum_{I_q \in Q_o} h_{I_q} \tag{27}$$

$$\text{Subject to} \quad \sum_{e \in \mathcal{P}'} a_e^{R_q} x_e + h_{R_q} \geq \underline{\delta}_{R_q} \quad R_q \in Q_\geq \tag{28}$$

$$\sum_{e \in \mathcal{P}'} a_e^{R_q} x_e + h_{R_q} \leq \bar{\delta}_{R_q} \quad R_q \in Q_\leq \tag{29}$$

$$\sum_{e \in \mathcal{P}'} a_e^{I_q} x_e + h_{I_q} = \ 1 \qquad I_q \in Q_o \tag{30}$$

$$x_e \in \mathbb{R}^+ \text{ with } e \in \mathcal{P}' \tag{31}$$

$$a_e^{R_q}, a_e^{I_q} \in \{0,1\},\ h_{R_q}, h_{I_q} \in \mathbb{R}^+ \text{ with } R_q \in Q_\geq \cup Q_\leq,\ I_q \in Q_o \tag{32}$$

Pricing Problem:

The objective of PP uses the dual values π, λ, ω as coefficients of the variables that are associated with a potential partition element. The binary variables r_{R_q}, r_{I_q}, b_q $(q \in N)$ are used to ensure the description logic semantics. A binary variable $r_{R_\top}$ is used to handle role hierarchy. A variable b_q is set to 1 if there exists an instance of concept q and r_{R_q} is set to 1 if there exists an R-neighbour that is an instance of concept q. Likewise, r_{I_q} is set to 1 if there exists a nominal q. Otherwise, these variables are set to 0. The PP is given below.

$$\text{Min} \sum_{q \in N} b_q - \sum_{R_q \in Q_\geq} \pi_{R_q} r_{R_q} - \sum_{R_q \in Q_\leq} \lambda_{R_q} r_{R_q} - \sum_{I_q \in Q_o} \omega_{I_q} r_{I_q} \tag{33}$$

Table 4.1: PP inequalities for DL axioms ($n \geq 1$)

DL Axiom	Inequality in PP	Description
$A_1 \sqcap ... \sqcap A_n \sqsubseteq B$	$\sum_{i=1}^{n} b_{A_i} - (n-1) \leq b_B$	If a set contains $A_1, ..., A_n$, then it also contains B.*
$A \sqsubseteq B_1 \sqcup ... \sqcup B_n$	$b_A \leq \sum_{i=1}^{n} b_{B_i}$	If a set contains A, then it also contains at least one concept from $B_1, ..., B_n$.

*Encodes unsatisfiability and disjointness in case $B \sqsubseteq_* \bot$

$$\text{Subject to} \quad r_{R_q} - b_q \leq 0 \qquad R_q \in Q_{\geq}, R \in N_R, q \in N \tag{34}$$

$$b_q - r_{R_q} \leq 0 \qquad R_q \in Q_{\leq}, R \in N_R, q \in N \tag{35}$$

$$r_{I_q} - b_q = 0 \qquad I_q \in Q_o, q \in N_o \tag{36}$$

$$r_{R_q} - r_{R_\top} \leq 0 \qquad R \in N_R, q \in N \tag{37}$$

$$r_{R_\top} - b_q \leq 0 \qquad R_q \in Q_\forall, R \in N_R, q \in N \tag{38}$$

$$r_{R_\top} - r_{S_\top} \leq 0 \qquad R \sqsubseteq S \in \mathcal{R}, R, S \in N_R \tag{39}$$

$$b_q, r_{R_q}, r_{I_q}, r_{R_\top}, r_{S_\top} \in \{0, 1\}$$

where vector π, λ and ω are dual variables associated with (28) to (30) respectively. For each at-least restriction represented in (28), Constraint (34) is added to PP, ensuring that if $r_{R_q} = 1$, then the variable for b_q must exist in $\mathcal{P}'$. Similarly, Constraint (35) is added for each at-most restriction represented in (29). Constraint (36) ensures the semantics of nominals represented in (28). Constraints (37) - (39) ensure the semantics of universal restrictions and role hierarchies respectively.

We can also map the semantics of selected DL axioms, where only atomic concepts occur, into inequalities, as shown in Table 4.1. For every $\mathcal{T} \models A \sqcap B \sqsubseteq C$, the algorithm adds $b_A + b_B - 1 \leq b_C$ to PP. Therefore, if PP generates a partition containing A and B, then it must also contain C. Similarly, for every $\mathcal{T} \models A \sqsubseteq B \sqcup C$, the algorithm adds $b_A \leq b_B + b_C$ to PP. This inequality ensures that if a partition contains A, then it must also contain B or C. Algorithm 4.2 (generatePP) generates the pricing problem that ensures the semantics of all related DL axioms i.e., universal

Algorithm 4.2 generatePP$(S_{\geq}, S_{\leq}, S_o, S_{\forall}, S_{\sqsubseteq}, S_{\perp}, S_{\mathcal{R}})$

```
Input: A set $S_{\geq}$ of at-least restrictions, a set $S_{\leq}$ of at-most restrictions, a set $S_o$ of related nominals, a set $S_{\forall}$ of universal restrictions, a set $S_{\sqsubseteq}$ of related subsumptions, a set $S_{\perp}$ of related disjointness, a set $S_{\mathcal{R}}$ of related role hierarchy
Output: PP
PP ← fresh model
PP ← add initial objective
PP ← add constraints to ensure semantics of $S_{\geq}, S_{\leq}, S_o$
for all $s_{\forall} \in S_{\forall}$ do
    PP ← add constraint for $s_{\forall}$ to handle universal restrictions
end for all
for all $s_{\sqsubseteq} \in S_{\sqsubseteq}$ do
    PP ← add constraint for $s_{\sqsubseteq}$ to handle subsumption
end for all
for all $s_{\perp} \in S_{\perp}$ do
    PP ← add constraint for $s_{\perp}$ to handle disjointness
end for all
for all $s_{\mathcal{R}} \in S_{\mathcal{R}}$ do
    PP ← add constraint for $s_{\mathcal{R}}$ to handle role hierarchy
end for all
```

restrictions, subsumptions, disjointness, and role hierarchies.

The algorithm solves RMP and PP as depicted in Algorithm 4.3 (solveInequalities). The algorithm gets the dual values by solving RMP and updates the objective of PP accordingly. PP computes a column that can maximally reduce the cost of RMP's objective. Whenever a column with a negative reduced cost is found, it is added to RMP. The process terminates when PP cannot compute a new column with a reduced cost, i.e., when the value of the objective function of PP becomes greater or equal to 0.

In the case of a non-integer solution, Algorithm 4.4 (applyBranchAndPrice) tries to find the optimal integer solution. The algorithm selects one of the non-integer variables and gets two solution subsets. After that, it explores both branches, in order to find the integer solution. The process continues until the algorithm either finds an optimal integer solution or detects a clash.

Algorithm 4.3 solveInequalities(RMP, PP)

```
Input: RMP, and PP
Output: Solution Set $\sigma$ or Clash
 1: $\{\sigma, S_{DV}\} \leftarrow$ Solve RMP and get solution and dual values
 2: PP $\leftarrow$ update PP objective using $S_{DV}$
 3: $\{reducedCost, newColumn\} \leftarrow$ solve PP
 4: if $reducedCost$ is negative then
 5:     RMP $\leftarrow$ add $newColumn$ in RMP
 6:     goto 1
 7: else if $\sigma$ is feasible then
 8:     if $\sigma$ is integer solution then
 9:         return $\sigma$
10:     else if $\sigma$ is non-integer solution then
11:         return applyBranchAndPrice(RMP, PP, $\sigma$)
12:     end if
13: else if $\sigma$ is infeasible then
14:     return Clash
15: end if
```

Algorithm 4.4 applyBranchAndPrice(RMP, PP, σ)

```
Input: RMP, PP, and a solution set $\sigma$ with non-integer solution
Output: Solution Set $\sigma$ or Clash
for all $x \in \sigma$ do
    if $x$ is non-Integer then
        $\{x_1, x_2\} \leftarrow$ get two solution subsets
        break
    end if
end for all
RMP1 $\leftarrow$ add new constraint in RMP to explore the branch $x_1$
PP1 $\leftarrow$ add new constraints in PP to explore the branch $x_1$
Solution $\leftarrow$ solveInequalities(RMP1, PP1)
if Solution is Clash then
    RMP1 $\leftarrow$ add new constraint in RMP to explore the branch $x_2$
    PP1 $\leftarrow$ add new constraints in PP to explore the branch $x_2$
    return $\leftarrow$ solveInequalities(RMP1, PP1)
else if Solution is $\sigma$ then
    return $\sigma$
end if
```

Definition 33. **(ILP Solution σ)** Upon completion of the branch-and-price process, the algorithm produces a solution set denoted by σ. This set, σ, may either be empty or include a solution derived from feasible inequalities. In cases of infeasibility, the algorithm signals a clash. A solution within

σ is characterized by a collection of tuples of the form $\langle \mathsf{R}, \mathsf{C}, n, \mathsf{V} \rangle$, where:

- R represents a set of roles ($\mathsf{R} \subseteq N_R$),
- C represents a set of concepts ($\mathsf{C} \subseteq N$),
- n denotes a cardinality ($n \in \mathbb{N}$, $n \geq 1$),
- V is an optional set that represents a set of nodes to be reused $\mathsf{V} \subseteq V$.

Each tuple represents a partition element of n individuals. One proxy node is created for each tuple.

4.3.2 Expansion Rules

In order to check the consistency of a Tbox $\mathcal{T}$, the proposed algorithm creates a completion graph G using the expansion rules shown in Table 4.2.

A node x in G contains a clash if (i) $\{A, \neg A\} \subseteq \mathcal{L}(x)$ for $A \in N_C$, or (ii) there is no feasible solution for $\mathcal{L}(x)$ in AR. G is complete if no expansion rule is applicable to any node in G. $\mathcal{T}$ is consistent if G is complete and no node in G contains a clash.

The $\sqcap$-Rule, $\sqcup$-Rule and $\forall$-Rule are similar to standard tableau expansion rules for $\mathcal{ALC}$. The $\forall_+$-Rule preserves the semantics of transitive roles.

The nom_{merge}**-Rule** merges two nodes containing in their label the same nominal. Suppose there is $o \in \mathcal{L}(x)$ and $o \in \mathcal{L}(y)$, and nodes x and y are not the same, then nom_{merge}-Rule merges x into y by

1. adding the label of x to the label of y, i.e., $\mathcal{L}(y) = \mathcal{L}(y) \cup \mathcal{L}(x)$,
2. moving all edges leading to x so that they lead to y. For instance, for all the nodes z such that $(z, x) \in E$
   (a) if $\{(y, z), (z, y)\} \cap E = \emptyset$, then add (z, y) to E and set $\mathcal{L}(z, y) = \mathcal{L}(z, x)$,
   (b) if $(z, y) \in E$, then set $\mathcal{L}(z, y) = \mathcal{L}(z, y) \cup \mathcal{L}(z, x)$,

Table 4.2: The expansion rules for $\mathcal{SHOIQ}$

$\sqcap$-Rule	**if** $(C_1 \sqcap C_2) \in \mathcal{L}(x)$ and $\{C_1, C_2\} \not\subseteq \mathcal{L}(x)$ **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 $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C\}$ for some $C \in \{C_1, C_2\}$
$\forall$-Rule	**if** $\forall S.C \in \mathcal{L}(x)$ and there $\exists y$ with $R \in \mathcal{L}(x, y)$, $C \notin \mathcal{L}(y)$ and $R \sqsubseteq_* S$ **then** set $\mathcal{L}(y) = \mathcal{L}(y) \cup \{C\}$
$\forall_+$-Rule	**if** $\forall S.C \in \mathcal{L}(x)$ and there exist U, R with $R \in N_{R_+}$ and $U \sqsubseteq_* R$, $R \sqsubseteq_* S$, and a node y with $U \in \mathcal{L}(x, y)$ and $\forall R.C \notin \mathcal{L}(y)$ **then** set $\mathcal{L}(y) = \mathcal{L}(y) \cup \{\forall R.C\}$
nom_{merge}-Rule	**if** for some $o \in N_o$ there are nodes x, y with $o \in \mathcal{L}(x) \cap \mathcal{L}(y)$, $x \neq y$ **then** merge x into y
$inverse$-Rule	**if** $\{\leq nR^-.C, \forall R^-.C\} \cap \mathcal{L}(y) \neq \emptyset$, $R \in \mathcal{L}(x, y)$, and $\langle x, \mathcal{L}(y, x)\rangle \notin \mathcal{B}(y)$ **then** set $\mathcal{B}(y) = \mathcal{B}(y) \cup \{\langle x, \mathcal{L}(y, x)\rangle\}$
$reset$-Rule	**if** 1. $\leq nR.C$ or $\geq nR.C$ propagated back to $\mathcal{L}(x)$ due to inverse role and $\{\leq nR.C, \geq nR.C\} \cap \mathcal{L}(x) = \emptyset$ and $R \in \mathcal{L}(x, y)$, $C \in \mathcal{L}(x)$, or 2. $\forall R.C \in \mathcal{L}(y)$, $R^- \in \mathcal{L}(x, y)$ and $o \in clos(C)$ for some $o \in N_o$ and $\{o\} \notin \mathcal{L}(x)$ **then** reset x by removing the blockable subtree of x.
ch-Rule	**if** $\leq nR.C \in \mathcal{L}(x)$ and $\forall R.(C \sqcup \neg C) \notin \mathcal{U}(x)$ **then** set $\mathcal{U}(x) = \mathcal{U}(x) \cup \{\forall R.(C \sqcup \neg C)\}$
fil-Rule	**if** the solution set σ is not empty and x is not blocked **then** for each $\langle \mathsf{R}, \mathsf{C}, n, \mathsf{V}\rangle \in \sigma(x)$ 1. **if** $\mathsf{V} = \emptyset$ and there exists no safe R-neighbour[*] y of x with $\mathsf{C} \subseteq \mathcal{L}(y)$, $\#y \geq n$, **then** create a new node y with $\mathcal{L}(y) \leftarrow \mathsf{C}$ and $\#y \leftarrow n$ 2. **else** for all $v \in \mathsf{V}$ add C to $\mathcal{L}(v)$ and set $\#v = n$
e-Rule	**if** $\langle \mathsf{R}, \mathsf{C}, n, \mathsf{V}\rangle \in \sigma(x)$ and $\mathsf{C} \subseteq \mathcal{L}(y)$, $\#y \geq n$, $\mathsf{R} \not\subseteq \mathcal{L}(x, y)$ **then** merge R into $\mathcal{L}(x, y)$ and $\{\mathsf{Inv}(R) \mid R \in \mathsf{R}\}$ into $\mathcal{L}(y, x)$, and for all S with $R \sqsubseteq_* S \in \mathcal{R}$ add S to $\mathcal{L}(x, y)$ and $\mathsf{Inv}(S)$ to $\mathcal{L}(y, x)$
ni-Rule	**if** $\leq nR.C \in \mathcal{L}(x)$, $o \in \mathcal{L}(x)$ for some $o \in N_o$, and there exists a blockable R-predecessor y of x such that $C \in \mathcal{L}(y)$ **then** create n new nodes $z_1, ..., z_n$ with $\mathcal{L}(x, z_i) = \{R\}$, $\mathcal{L}(z_i) = \{C, o_i\}$ with $o_i \in N_o$ new in G with $1 \leq i \leq n$.
$\leq_{nom}$-Rule	**if** 1. $\leq nR.C \in \mathcal{L}(x)$, $o \in \mathcal{L}(x)$ for some $o \in N_o$, and there exists a blockable R-neighbour y of x such that $C \in \mathcal{L}(y)$, 2. there exists n nominal R-neighbours $z_1, ..., z_n$ of x with $C \in \mathcal{L}(z_i)$ and $1 \leq i \leq n$, and 3. there is a nominal R-neighbour z of x with $C \in \mathcal{L}(z)$ **then** merge y into z

[*] An R-neighbour y of a node x is *safe* if either x is blockable or if y is not blocked and x is a nominal node.

(c) if $(y, z) \in E$, then set $\mathcal{L}(y, z) = \mathcal{L}(y, z) \cup \{R^- \mid R \in \mathcal{L}(z, x)\}$,

(d) remove edge (z, x) from E;

3. moving all edges leading from x to nominal nodes so that they lead from y to the same nominal nodes. For instance, for all nominal nodes z such that $(x, z) \in E$

   (a) if $\{(y, z), (z, y)\} \cap E = \emptyset$, then add (y, z) to E and set $\mathcal{L}(y, z) = \mathcal{L}(x, z)$,

   (b) if $(y, z) \in E$, then set $\mathcal{L}(y, z) = \mathcal{L}(y, z) \cup \mathcal{L}(x, z)$,

   (c) if $(z, y) \in E$, then set $\mathcal{L}(z, y) = \mathcal{L}(z, y) \cup \{R^- \mid R \in \mathcal{L}(x, z)\}$,

   (d) remove edge (x, z) from E;

4. merging $\mathcal{B}(x)$ into $\mathcal{B}(y)$ and $\mathcal{U}(x)$ into $\mathcal{U}(y)$

5. removing x and blockable sub-trees below x and y

If $\mathcal{L}(x, y) = \{R\}$ and $\{\leq nR^-.C, \forall R^-.C\} \cap \mathcal{L}(y) \neq \emptyset$, then the $inverse$**-Rule** encodes the already existing R^--edge by adding a tuple $\langle x, \{R^-\}\rangle$ to $\mathcal{B}(y)$. This information is crucial in the case of at-most restrictions or when nominals occur in universal restrictions. For example, consider the axioms

$$A \sqsubseteq \exists R.B \tag{40}$$

$$B \sqsubseteq \geq 2R^-.C \sqcap \exists R^-.D \sqcap \forall S^-.\{o_1, o_2\} \tag{41}$$

$$o_1 \sqcap o_2 \sqsubseteq \bot \tag{42}$$

where $\{A, B, C, D\} \subseteq N_C$, $\{R, S\} \subseteq N_R$, $R \sqsubseteq S \in \mathcal{R}$ and $\{o_1, o_2\} \subseteq N_o$.

Suppose we have $A \in \mathcal{L}(x)$, $R, S \in \mathcal{L}(x, y)$ and $B \in \mathcal{L}(y)$. Since nominals carry numerical restrictions, $\forall S^-.\{o_1, o_2\}$ implies that we can have at most 2 S^--neighbours of y. However, standard tableau reasoners might create three new S^--neighbours of y without considering the existing S^--neighbour x of y. Then they try to merge these four nodes in a nondeterministic way to satisfy the numerical restriction imposed by nominals. In our approach, the $inverse$-Rule encodes information about an existing S^--neighbour of y by adding $\langle x, \{R^-, S^-\}\rangle$ to $\mathcal{B}(y)$ and the algorithm generates a deterministic solution.

The ch**-Rule** takes care of at-most restrictions. For instance, if $\leq nR.C \in \mathcal{L}(x)$ and there are R-neighbours of x, then the at-most restriction $\leq nR.C$ can be violated. We know that an R-neighbour y of x either has C or $\neg C$ in its label. Therefore, for every at-most restriction $\leq nR.C$, the ch-Rule adds the universal restriction $\forall R.(C \sqcup \neg C)$ to $\mathcal{U}(x)$ and the algorithm handles these universal restrictions in AR. This rule is similar to the ch-Rule in [41].

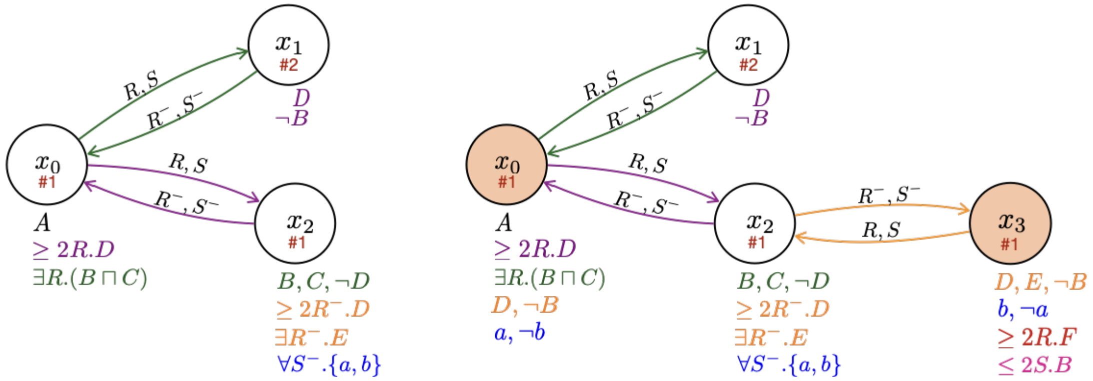
If a new numerical restriction on role R is added to $\mathcal{L}(x)$, and there exists an R-neighbour of x, then the $reset$**-Rule** resets the node x by removing the blockable subtree of x. Similarly, it resets the node x if a new nominal o is added to $\mathcal{L}(x)$.

For a node x, the algorithm transforms all nominals, numerical, existential, and universal restrictions to a corresponding system of inequalities. The algorithm then processes these inequalities and gives back a solution set $\sigma(x)$. The set $\sigma(x)$ is either empty or contains solutions derived from feasible inequalities. In case of infeasibility, the algorithm signals a clash. A proxy node (see Definition 31) is created corresponding to each tuple, which represents n R-neighbours of x (where R is a set of roles) that are all instances of all elements of C. V is an optional set that contains existing R-neighbours of x that must be reused and C is added to their labels. Consider the axioms (40) - (42), the algorithm returns the solution $\sigma(y) = \{\langle\{R^-, S^-\}, \{A, C, o_1\}, 1, \{x\}\rangle, \langle\{R^-, S^-\}, \{D, C, o_2\}, 1, \emptyset\rangle\}$.

The fil**-Rule** is used to generate proxy nodes based on the arithmetic solution that satisfies a set of inequalities. For the above solution, the fil-Rule creates only one node z with cardinality 1, such that $\mathcal{L}(z) \leftarrow \{D, C, o_2\}$ and $\sharp z = 1$, and reuses the existing node x, adding C and o_1 to $\mathcal{L}(x)$.

The e**-Rule** creates an edge between nodes y and z, and adds R^-, S^- to $\mathcal{L}(y, z)$ and $\mathsf{Inv}(R^-), \mathsf{Inv}(S^-)$ to $\mathcal{L}(z, y)$. The e-Rule always adds all implied superroles to edge labels.

The ni**-Rule** and the $\leq_{nom}$**-Rule** are employed to ensure termination by fixing the upper bound on the number of nominal nodes. This prevents the reasoner from repeatedly creating and merging



(a)

(b)

Figure 4.2: The application of the *fil*-Rule, the *e*-Rule and the *inverse*-Rule

nominal nodes. These two rules have been adapted from [41, 52], and are discussed in Section 3.1. The *ni*-Rule and the $\leq_{nom}$-Rule are not applicable in the absence of inverse roles, nominals or number restrictions in the input ontology.

4.3.3 Example Application of the Expansion Rules

Consider the small Tbox, given in axioms (43) - (48)

$$A \sqsubseteq \exists R.(B \sqcap C) \sqcap \geq 2R.D \tag{43}$$

$$C \sqsubseteq \geq 2R^-.D \sqcap \exists R^-.E \sqcap \forall S^-.\{a,b\} \tag{44}$$

$$a \sqcap b \sqsubseteq \bot \tag{45}$$

$$B \sqcap D \sqsubseteq \bot \tag{46}$$

$$E \sqsubseteq \geq 2R.F \sqcap \leq 2S.B \tag{47}$$

$$F \sqsubseteq B \tag{48}$$

where $\{A, B, C, D, E, F\} \subseteq N_C$, $\{R, S\} \subseteq N_R$, $R \sqsubseteq S \in \mathcal{R}$ and $\{a, b\} \subseteq N_o$.

Our algorithm starts with an initial completion graph G having a single node x_0 with

$\mathcal{L}(x_0) = \{A\}$, $\mathcal{B}(x_0) = \emptyset$ and $\mathcal{U}(x_0) = \emptyset$. After unfolding A in the label of x_0 and applying the $\sqcap$-Rule, we obtain $\mathcal{L}(x_0) = \{A, \exists R.(B \sqcap C), \geq 2R.D\}$. For a node x_0, the algorithm transforms all numerical and existential restrictions to a corresponding system of inequalities. After this, it processes these inequalities and gives back a solution set $\sigma(x_0) = \{\langle\{R\}, \{D, \neg B\}, 2, \emptyset\rangle, \langle\{R\}, \{B, C, \neg D\}, 1, \emptyset\rangle\}$.

For $\sigma(x_0)$, we apply the *fil*-Rule which creates two nodes; x_1 with cardinality 2 such that $\mathcal{L}(x_1) \leftarrow \{D, \neg B\}$ and $\sharp x_1 = 2$, and x_2 with cardinality 1 such that $\mathcal{L}(x_2) \leftarrow \{B, C, \neg D\}$ and $\sharp x_2 = 1$.

Next, the e-Rule creates an edge between nodes x_0 and x_1, adding R, S to $\mathcal{L}(x_0, x_1)$ and $\mathsf{Inv}(R), \mathsf{Inv}(S)$ to $\mathcal{L}(x_1, x_0)$. Similarly, it creates an edge between nodes x_0 and x_2, and adds R, S to $\mathcal{L}(x_0, x_2)$ and $\mathsf{Inv}(R), \mathsf{Inv}(S)$ to $\mathcal{L}(x_2, x_0)$. Therefore, G has been extended with two new nodes as shown in Figure 4.2a.

The algorithm proceeds by unfolding C in the label of x_2. Next, we apply the $\sqcap$-Rule and obtain $\mathcal{L}(x_2) = \{B, C, \neg D, \geq 2R^-.D, \exists R^-.E, \forall S^-.\{a, b\}\}$. The $inverse$-Rule encodes information about an existing S^--neighbour of x_2 by adding $\langle x_0, \{R^-, S^-\}\rangle$ to $\mathcal{B}(y)$. Since we do not have any at-most restrictions, $\mathcal{U}(x_2) = \emptyset$. The algorithm generates a corresponding system of inequalities. After processing these inequalities, a solution $\sigma(x_2) = \{\langle\{R^-, S^-\}, \{D, a, \neg B, \neg b\}, 1, \{x_0\}\rangle, \langle\{R^-, S^-\}, \{D, E, b, \neg B, \neg a\}, 1, \emptyset\rangle\}$ is generated.

For $\sigma(x_2)$, we apply the *fil*-Rule which creates only one node; x_3 with cardinality 1 such that $\mathcal{L}(x_3) \leftarrow \{D, E, b, \neg B, \neg a\}$ and $\sharp x_3 = 1$, and reuses the existing node x_0 with $\mathcal{L}(x_0) = \mathcal{L}(x_0) \cup \{D, a, \neg B, \neg b\}$.

The e-Rule creates an edge between nodes x_2 and x_3, and adds R^-, S^- to $\mathcal{L}(x_2, x_3)$ and $\mathsf{Inv}(R^-), \mathsf{Inv}(S^-)$ to $\mathcal{L}(x_3, x_2)$. The label of x_3 has been extended by unfolding E. Figure 4.2b shows the extended completion graph.

The ni-Rule is now applicable, which is managed by the AR. All necessary details for generating new S-successors of x_3 are encoded into inequalities, while the $inverse$-Rule encodes information about an existing S-neighbour. The algorithm facilitates the generation of new nominal nodes

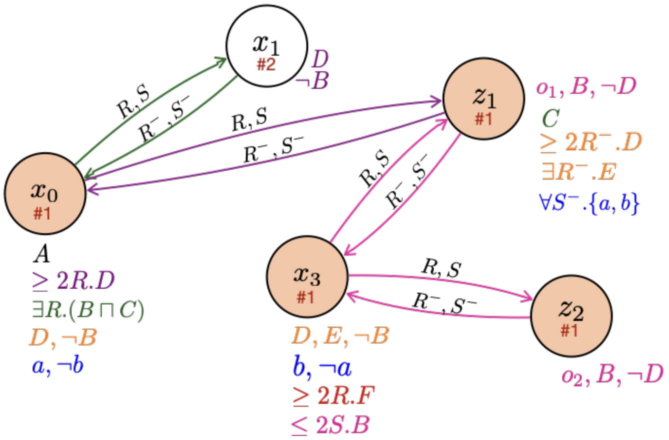


Figure 4.3: x_2 is merged into z_1 after the $\leq_{nom}$-Rule application

by introducing two new nominals o_1 and o_2. However, rather than creating two new S-successors of x_3 and then merging one with an existing S-neighbour of x_3, AR opts to utilize the existing x_2 of x_3, as depicted in Figure 4.3. This strategy not only incorporates the $\leq_{nom}$-Rule within AR but also avoids unnecessary merging. Furthermore, the combined application of the ni-Rule and the $\leq_{nom}$-Rule ensures the preservation of a tree-shaped completion graph for the blockable part.

The $inverse$-Rule encodes information about the existing S-neighbours of x_3 such that $\mathcal{B}(x_3) \leftarrow \{\langle z_1, \{R, S\}\rangle, \langle z_2, \{R, S\}\rangle\}$.

Since $\leq 2S.B \in \mathcal{L}(x_3)$, the ch-Rule becomes applicable. It adds the universal restriction $\forall S.\{B \sqcup \neg B\}$ to $\mathcal{U}(x_3)$. Therefore, for node x_3, we obtain $\mathcal{L}(x_3) = \{D, E, A, b, \neg B, \neg a, \geq 2R.F, \leq 2S.B\}$, $\mathcal{B}(x_3) = \{\langle z_1, \{R, S\}\rangle, \langle z_2, \{R, S\}\rangle\}$ and $\mathcal{U}(x_3) = \{\forall S.\{B \sqcup \neg B\}\}$. The algorithm transforms all numerical and universal restrictions to a corresponding system of inequalities. It also encodes the information present in $\mathcal{B}(x_3)$. The algorithm solves these inequalities and gives back a solution set $\sigma(x_3) = \{\langle\{R, S\}, \{F, B, \neg D\}, 1, z_1\rangle, \langle\{R, S\}, \{F, B, \neg D\}, 1, z_2\rangle\}$.

According to this solution, the *fil*-Rule reuses the existing nodes z_1 and z_2, and adds F in $\mathcal{L}(z_1)$ and $\mathcal{L}(z_2)$. The algorithm terminates as no more expansion rules are applicable. Figure 4.4 shows the final completion graph.

Since we are reusing the existing nodes, our algorithm minimizes the nondeterministic merging which is one of the main sources of inefficiency in standard tableau reasoning.

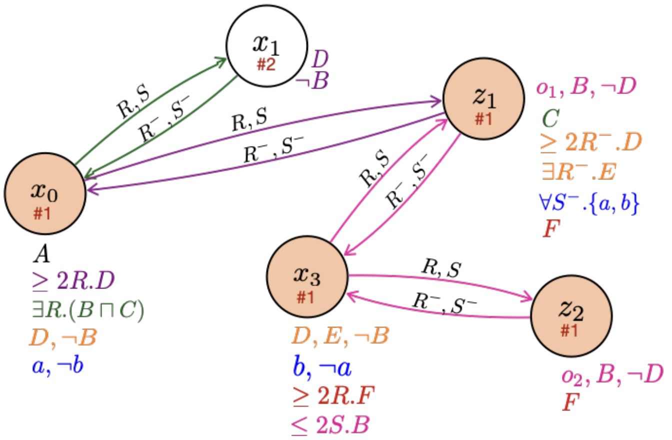


Figure 4.4: Final completion graph after applying all relevant expansion rules

4.3.4 Example Illustrating Inequalities Generation and ILP Formulation

4.3.4.1 Example with Integer Solution

Consider the axioms

$$B \sqsubseteq \geq 2R.C \sqcap \leq 1R.A \sqcap \forall S.\{o_1, o_2\} \tag{49}$$

$$o_1 \sqcap o_2 \sqsubseteq \bot \tag{50}$$

with $N_R = \{R, S\}$, $\{A, B, C\} \subseteq N_C, \{o_1, o_2\} \subseteq N_o$, and $R \sqsubseteq S \in \mathcal{R}$. For the sake of better readability, we apply in this example lazy unfolding [4, 38]. We start with a node x and its label $\mathcal{L}(x) = \{B\}$ and by unfolding B and applying the $\sqcap$-Rule we get $\mathcal{L}(x) = \{B, \geq 2R.C, \leq 1R.A, \forall S.\{o_1, o_2\}\}$. By applying the ch-Rule, we get $\mathcal{U}(x) = \{\forall R.(A \sqcup \neg A)\}$. Since $\{\geq 2R.C, \leq 1R.A, \forall S.\{o_1, o_2\}\} \subseteq \mathcal{L}(x)$, the algorithm generates a corresponding set of inequalities and applies ILP considering known subsumptions and disjointness.

For solving these inequalities, RMP starts with artificial variables, $\mathcal{P}'$ is initially empty, $Q_{\geq} = \{R_C\}$, $Q_{\leq} = \{R_A\}$, $Q_{\forall} = \{S_{\top}, R_{\top}\}$ and $Q_o = \{I_{o1}, I_{o2}\}$. The objective of (RMP 1) contains the sum of artificial variables along with the cost $M = 10$. Constraint (a) corresponds to the at-least restriction $\geq 2R.C$ and Constraint (b) corresponds to the at-most restriction $\leq 1R.A$. Moreover, Constraints (c) and (d) ensure the nominal semantics.

Min $$10h_{R_C} + 10h_{R_A} + 10h_{I_{o1}} + 10h_{I_{o2}} \tag{RMP 1}$$

Subject to

$$\begin{aligned} h_{R_C} &\geq 2 && (a) \\ h_{R_A} &\leq 1 && (b) \\ h_{I_{o1}} &= 1 && (c) \\ h_{I_{o2}} &= 1 && (d) \end{aligned}$$

Solution: $cost = 40$, $h_{R_C} = 2$, $h_{R_A} = 0, h_{I_{o1}} = 1, h_{I_{o2}} = 1$

Duals: $\pi_{R_C} = 20, \lambda_{R_A} = 0, \omega_{I_{o1}} = 10, \omega_{I_{o2}} = 10$

After solving (RMP 1), we get the optimal solution with cost 40 with 3 non-zero artificial variables. The objective of (PP 1) uses the dual values from (RMP 1). For at-least restriction, a constraint (i.e., $\geq 2R.C \rightsquigarrow r_{R_C} - b_C \leq 0$) is added to (PP 1), which indicates that if $r_{R_C} = 1$ then a variable b_C will also be 1. Similarly, for each at-most restriction, a constraint (i.e., $\leq 1R.A \rightsquigarrow b_A - r_{R_A} \leq 0$) is added. It means that if a partition contains a qualification of an at-most QCR (i.e., $b_A = 1$), then a corresponding variable containing A in its subscript must exist (that means $r_{R_A} = 1$). In order to ensure the semantics of nominals, a constraint corresponding to each nominal (e.g., $\{o_1\} \rightsquigarrow r_{I_{o1}} - b_{o1} = 0$) is added to (PP 1). Constraints (i) to (iii) represent the role hierarchy. The semantics of universal restriction are embedded in Constraints (iv) and (v). Constraints (vi) and (vii) ensure that both $o1$ and $o2$, and A and $\neg A$ cannot exist in the same partition element.

Min $$b_C + b_A + b_{\neg A} + b_{o1} + b_{o2} - 20r_{R_C} - 0r_{R_A} - 10r_{I_{o1}} - 10r_{I_{o2}} \tag{PP 1}$$

Subject to

$$\begin{aligned} r_{R_C} - b_C &\leq 0 \\ b_A - r_{R_A} &\leq 0 \\ r_{I_{o1}} - b_{o1} &= 0 \\ r_{I_{o2}} - b_{o2} &= 0 \\ r_{R_C} - r_{R_\top} &\leq 0 && (i) \\ r_{R_A} - r_{R_\top} &\leq 0 && (ii) && \text{(PPC 1)} \\ r_{R_\top} - r_{S_\top} &\leq 0 && (iii) \\ r_{R_\top} - (b_A + b_{\neg A}) &\leq 0 && (iv) \\ r_{S_\top} - (b_{o1} + b_{o2}) &\leq 0 && (v) \\ b_{o1} + b_{o2} &\leq 1 && (vi) \\ b_A + b_{\neg A} &\leq 1 && (vii) \end{aligned}$$

Solution: $cost = -27, b_C = 1, b_{o1} = 1, b_{\neg A} = 1, r_{R_C} = 1, r_{I_{o1}} = 1, r_{R_\top} = 1, r_{S_\top} = 1$

The values of r_{R_C}, $r_{I_{o1}}$, $r_{R_\top}$ and $r_{S_\top}$ are 1 in (PP 1), therefore, the variable $x_{S_\top R_\top R_C I_{o1}}$ is added to (RMP 2) in corresponding inequalities. Since three b variables, b_C, b_{o1}, $b_{\neg A}$, are 1, the cost of $x_{S_\top R_\top R_C I_{o1}}$ is 3. $\mathcal{P}' = \{\{R_C, I_{o1}, S_\top, R_\top\}\}$ and the value of the objective function is reduced from 40 in (RMP 1) to 23 in (RMP 2). However, (RMP 2) still has two non-zero h variables.

$$\text{Min} \quad 3x_{S_\top R_\top R_C I_{o1}} + 10h_{R_C} + 10h_{R_A} + 10h_{I_{o1}} + 10h_{I_{o2}} \qquad \text{(RMP 2)}$$

Subject to

$$\begin{aligned} x_{S_\top R_\top R_C I_{o1}} + h_{R_C} &\geq 2 \\ h_{R_A} &\leq 1 \\ x_{S_\top R_\top R_C I_{o1}} + h_{I_{o1}} &= 1 \\ h_{I_{o2}} &= 1 \end{aligned}$$

Solution: $cost = 23, x_{S_\top R_\top R_C I_{o1}} = 1, h_{R_C} = 1, h_{R_A} = 0, h_{I_{o1}} = 0, h_{I_{o2}} = 1$

Duals: $\pi_{R_C} = 10, \lambda_{R_A} = 0, \omega_{I_{o1}} = -7, \omega_{I_{o2}} = 10$

The objective of (PP 2) is updated using the dual values from (RMP 2).

$$\text{Min} \qquad b_C + b_A + b_{\neg A} + b_{o1} + b_{o2} - 10r_{R_C} - 0r_{R_A} + 7r_{I_{o1}} - 10r_{I_{o2}} \tag{PP 2}$$

Subject to (PPC 1)

Solution: $cost = -17, b_C = 1, b_{o2} = 1, b_{\neg A} = 1, r_{R_C} = 1, r_{I_{o2}} = 1, r_{R_\top} = 1, r_{S_\top} = 1$

The values of $r_{R_C}, r_{I_{o2}}, r_{R_\top}$ and $r_{S_\top}$ are 1 in (PP 2), therefore, the variable $x_{S_\top R_\top R_C I_{o2}}$ is added to (RMP 3) in corresponding inequalities. Since three b variables, b_C, b_{o2}, $b_{\neg A}$, are 1, the cost of $x_{S_\top R_\top R_C I_{o2}}$ is 3. $\mathcal{P}' = \{R_C, I_{o1}, S_\top, R_\top\}, \{R_C, I_{o2}, S_\top, R_\top\}\}$ and the cost is further reduced from 23 in (RMP 2) to 6 in (RMP 3).

$$\text{Min} \qquad 3x_{S_\top R_\top R_C I_{o2}} + 3x_{S_\top R_\top R_C I_{o1}} + 10h_{R_C} + 10h_{R_A} + 10h_{I_{o1}} + 10h_{I_{o2}} \tag{RMP 3}$$

Subject to

$$x_{S_\top R_\top R_C I_{o2}} + x_{S_\top R_\top R_C I_{o1}} + h_{R_C} \geq 2$$

$$h_{R_A} \leq 1$$

$$x_{S_\top R_\top R_C I_{o1}} + h_{I_{o1}} = 1$$

$$x_{S_\top R_\top R_C I_{o2}} + h_{I_{o2}} = 1$$

Solution:$cost = 6, x_{S_\top R_\top R_C I_{o1}} = 1, x_{S_\top R_\top R_C I_{o2}} = 1, h_{R_C}, h_{R_A}, h_{I_{o1}}, h_{I_{o2}} = 0$

Duals: $\pi_{R_C} = 10, \lambda_{R_A} = 0, \omega_{I_{o1}} = -7, \omega_{I_{o2}} = -7$

The objective of (PP 3) is updated using the dual values from (RMP 3).

$$\text{Min} \qquad b_C + b_A + b_{\neg A} + b_{o1} + b_{o2} - 10r_{R_C} - 0r_{R_A} + 7r_{I_{o1}} + 7r_{I_{o2}} \tag{PP 3}$$

Subject to (PPC 1)

Solution: $cost = 0$, all variables are 0.

All artificial variables in (RMP 3) are zero, indicating that we may have reached a feasible solution. The reduced cost of (PP 3) is not negative anymore which means that (RMP 3) cannot be improved further. Therefore, the algorithm terminates after the third ILP iteration and returns the optimal solution $\sigma(x) = \{\langle \{R, S\}, \{C, o_1, \neg A\}, 1, \emptyset\rangle, \langle \{R, S\}, \{C, o_2, \neg A\}, 1, \emptyset\rangle\}$.

The *fil*-Rule creates two new nodes x_1 and x_2 with $\mathcal{L}(x_1) \leftarrow \{C, o_1, \neg A\}$, $\mathcal{L}(x_2) \leftarrow \{C, o_2, \neg A\}$, $\sharp x_1 \leftarrow 1$ and $\sharp x_2 \leftarrow 1$.

The e-Rule creates edges $\langle x, x_1\rangle$ and $\langle x, x_2\rangle$ with $\mathcal{L}(\langle x, x_1\rangle) \leftarrow \{R, S\}$ and $\mathcal{L}(\langle x, x_2\rangle) \leftarrow \{R, S\}$ (because $R \sqsubseteq S \in \mathcal{R}$). It also creates back edges $\langle x_1, x\rangle$ and $\langle x_2, x\rangle$ with $\mathcal{L}(\langle x_1, x\rangle) \leftarrow \{R^-, S^-\}$ and $\mathcal{L}(\langle x_2, x\rangle) \leftarrow \{R^-, S^-\}$.

4.3.4.2 Example with Non-Integer Solution

We use a very simple example to show how we use the branch-and-price method to get an integer solution. Consider a concept, HappyFather, defined as a person with at most 2 children, among whom there should be an Artist, a Doctor, a Professor, and a Lawyer. Additionally, a Doctor cannot be a Lawyer. For simplicity, we use short names: $HF(HappyFather)$, $A(Artist)$, $D(Doctor)$, $P(Professor)$, $L(Lawyer)$, and we use R for the role hasChild.

Now consider a small Tbox:

$$HF \sqsubseteq \geq 1R.A \sqcap \geq 1R.D \sqcap \geq 1R.P \sqcap \geq 1R.L \sqcap \leq 2R.\top \tag{51}$$

$$D \sqcap L \sqsubseteq \bot \tag{52}$$

with $N_R = \{R\}$, and $\{HF, A, D, P, L\} \subseteq N_C$. We start with a node x and its label $\mathcal{L}(x) = \{HF\}$ and by unfolding HF and applying the $\sqcap$-Rule we get $\mathcal{L}(x) = \{HF, \geq 1R.A, \geq 1R.D, \geq 1R.P, \geq 1R.L, \leq 2R.\top\}$. The algorithm generates a corresponding set of inequalities and applies ILP considering known disjointness. For solving these inequalities, RMP starts with artificial variables, $\mathcal{P}'$ is initially empty, $Q_{\geq} = \{R_A, R_D, R_P, R_L\}$, $Q_{\leq} = \{R_{\top}\}$, $Q_{\forall} = \emptyset$ and $Q_o = \emptyset$. The objective of (RMP 4) contains the sum of artificial variables along with the cost $M = 10$. Constraints (a) to (d) correspond to at-least restrictions $\geq 1R.A$, $\geq 1R.D$, $\geq 1R.P$, and $\geq 1R.L$

respectively. Constraint (e) corresponds to at-most restriction $\leq 2R.\top$.

$$\text{Min} \qquad 10h_{R_A} + 10h_{R_D} + 10h_{R_P} + 10h_{R_L} + 10h_{R_\top} \qquad \text{(RMP 4)}$$

Subject to

$$\begin{aligned} h_{R_A} &\geq 1 && (a)\\ h_{R_D} &\geq 1 && (b)\\ h_{R_P} &\geq 1 && (c)\\ h_{R_L} &\geq 1 && (d)\\ h_{R_\top} &\leq 2 && (e) \end{aligned}$$

Solution: $cost = 40, h_{R_A} = 1, h_{R_D} = 1, h_{R_P} = 1, h_{R_L} = 1$

Duals: $\pi_{R_A} = 10, \pi_{R_D} = 10, \pi_{R_P} = 10, \pi_{R_L} = 10$

After solving (RMP 4), we get the optimal solution with cost 40 with 4 non-zero artificial variables. The objective of (PP 4) uses the dual values from (RMP 4). Constraint (i) ensures that D and L cannot exist in the same partition element.

$$\text{Min} \qquad b_A + b_D + b_P + b_L + b_\top - 10r_{R_A} - 10r_{R_D} - 10r_{R_P} - 10r_{R_L} - 0r_{R_\top} \qquad \text{(PP 4)}$$

Subject to

$$
\begin{aligned}
r_{R_A} - b_A &\le 0 \\
r_{R_D} - b_D &\le 0 \\
r_{R_P} - b_P &\le 0 \\
r_{R_L} - b_L &\le 0 \\
b_\top - r_{R_\top} &\le 0 \\
b_D + b_L &\le 1 \qquad\qquad (i) \\
b_A - b_\top &\le 0 \\
b_D - b_\top &\le 0 \\
b_P - b_\top &\le 0 \\
b_L - b_\top &\le 0
\end{aligned}
\tag{PPC 2}
$$

Solution: $cost = -26$, $b_A = 1$, $b_D = 1$, $b_P = 1$, $b_\top = 1$, $r_{R_A} = 1$, $r_{R_D} = 1$, $r_{R_P} = 1$, $r_{R_\top} = 1$

The values of r_{R_A}, r_{R_D}, r_{R_P}, and $r_{R_\top}$ are 1 in (PP 4), therefore, a variable $x_{R_A R_D R_P R_\top}$ is added to (RMP 5) in corresponding inequalities. Since four b variables, b_A, b_D, b_P, $b_\top$ are 1, the cost of $x_{R_A R_D R_P R_\top}$ is 4. $\mathcal{P}' = \{\{R_A, R_D, R_P, R_\top\}\}$ and the value of the objective function is reduced from 40 in (RMP 4) to 14 in (RMP 5). However, (RMP 5) still has one non-zero h variable.

Min $$4x_{R_A R_D R_P R_\top} + 10h_{R_A} + 10h_{R_D} + 10h_{R_P} + 10h_{R_L} + 10h_{R_\top} \tag{RMP 5}$$

Subject to

$$
\begin{aligned}
x_{R_A R_D R_P R_\top} + h_{R_A} &\ge 1 \\
x_{R_A R_D R_P R_\top} + h_{R_D} &\ge 1 \\
x_{R_A R_D R_P R_\top} + h_{R_P} &\ge 1 \\
h_{R_L} &\ge 1 \\
x_{R_A R_D R_P R_\top} + h_{R_\top} &\le 2
\end{aligned}
$$

Solution: $cost = 14, x_{R_A R_D R_P R_\top} = 1, h_{R_L} = 1$

Duals: $\pi_{R_A} = 4, \pi_{R_L} = 10$

The objective of (PP 5) is updated using the dual values from (RMP 5).

$$\text{Min} \qquad b_A + b_D + b_P + b_L + b_\top - 4r_{R_A} - 0r_{R_D} - 0r_{R_P} - 10r_{R_L} - 0r_{R_\top} \tag{PP 5}$$

Subject to (PPC 2)

Solution: $cost = -11, b_A = 1, b_L = 1, b_\top = 1, r_{R_A} = 1, r_{R_L} = 1, r_{R_\top} = 1$

A new variable $x_{R_A R_L R_\top}$ is added to (RMP 6) in corresponding inequalities.

$$\text{Min} \qquad 3x_{R_A R_L R_\top} + 4x_{R_A R_D R_P R_\top} + 10h_{R_A} + 10h_{R_D} + 10h_{R_P} + 10h_{R_L} + 10h_{R_\top} \tag{RMP 6}$$

Subject to

$$\begin{aligned} x_{R_A R_L R_\top} + x_{R_A R_D R_P R_\top} + h_{R_A} &\geq 1 \\ x_{R_A R_D R_P R_\top} + h_{R_D} &\geq 1 \\ x_{R_A R_D R_P R_\top} + h_{R_P} &\geq 1 \\ x_{R_A R_L R_\top} + h_{R_L} &\geq 1 \\ x_{R_A R_L R_\top} + x_{R_A R_D R_P R_\top} + h_{R_\top} &\leq 2 \end{aligned}$$

Solution: $cost = 7, x_{R_A R_L R_\top} = 1, x_{R_A R_D R_P R_\top} = 1$

Duals: $\pi_{R_D} = 4, \pi_{R_L} = 3$

The dual values from (RMP 6) are used to update the objective of (PP 6).

$$\text{Min} \qquad b_A + b_D + b_P + b_L + b_\top - 0r_{R_A} - 4r_{R_D} - 0r_{R_P} - 3r_{R_L} - 0r_{R_\top} \tag{PP 6}$$

Subject to (PPC 2)

Solution: $cost = -2, b_D = 1, b_\top = 1, r_{R_D} = 1, r_{R_\top} = 1$

A variable $x_{R_D R_\top}$ is added to (RMP 7) in corresponding inequalities.

$$\text{Min} \quad 2x_{R_D R_\top} + 3x_{R_A R_L R_\top} + 4x_{R_A R_D R_P R_\top} + 10h_{R_A} + 10h_{R_D} + 10h_{R_P} + 10h_{R_L} + 10h_{R_\top} \tag{RMP 7}$$

Subject to

$$x_{R_A R_L R_\top} + x_{R_A R_D R_P R_\top} + h_{R_A} \geq 1$$

$$x_{R_D R_\top} + x_{R_A R_D R_P R_\top} + h_{R_D} \geq 1$$

$$x_{R_A R_D R_P R_\top} + h_{R_P} \geq 1$$

$$x_{R_A R_L R_\top} + h_{R_L} \geq 1$$

$$x_{R_D R_\top} + x_{R_A R_L R_\top} + x_{R_A R_D R_P R_\top} + h_{R_\top} \leq 2$$

Solution: $cost = 7, x_{R_A R_L R_\top} = 1, x_{R_A R_D R_P R_\top} = 1$

Duals: $\pi_{R_D} = 2, \pi_{R_P} = 2, \pi_{R_L} = 3$

The objective of (PP 7) is updated using the dual values from (RMP 7).

$$\text{Min} \quad b_A + b_D + b_P + b_L + b_\top - 0r_{R_A} - 2r_{R_D} - 2r_{R_P} - 3r_{R_L} - 0r_{R_\top} \tag{PP 7}$$

Subject to (PPC 2)

Solution: $cost = -2, b_P = 1, b_L = 1, b_\top = 1, r_{R_P} = 1, r_{R_L} = 1, r_{R_\top} = 1$

A variable $x_{R_P R_L R_\top}$ is added to (RMP 8) in corresponding inequalities.

$$\text{Min} \quad 3x_{R_P R_L R_\top} + 2x_{R_D R_\top} + 3x_{R_A R_L R_\top} + 4x_{R_A R_D R_P R_\top} + 10h_{R_A} + 10h_{R_D} + 10h_{R_P} + 10h_{R_L} + 10h_{R_\top} \tag{RMP 8}$$

Subject to

$$x_{R_A R_L R_\top} + x_{R_A R_D R_P R_\top} + h_{R_A} \geq 1$$

$$x_{R_D R_\top} + x_{R_A R_D R_P R_\top} + h_{R_D} \geq 1$$

$$x_{R_P R_L R_\top} + x_{R_A R_D R_P R_\top} + h_{R_P} \geq 1$$

$$x_{R_P R_L R_\top} + x_{R_A R_L R_\top} + h_{R_L} \geq 1$$

$$x_{R_P R_L R_\top} + x_{R_D R_\top} + x_{R_A R_L R_\top} + x_{R_A R_D R_P R_\top} + h_{R_\top} \leq 2$$

Solution: $cost = 6, x_{R_P R_L R_\top} = 0.5, x_{R_D R_\top} = 0.5, x_{R_A R_L R_\top} = 0.5, x_{R_A R_D R_P R_\top} = 0.5$

Duals: $\pi_{R_A} = 1, \pi_{R_D} = 2, \pi_{R_P} = 1, \pi_{R_L} = 2$

The objective of (PP 8) is updated using the dual values from (RMP 8).

$$\text{Min} \qquad b_A + b_D + b_P + b_L + b_\top - 1r_{R_A} - 2r_{R_D} - 1r_{R_P} - 2r_{R_L} - 0r_{R_\top} \qquad \text{(PP 8)}$$

Subject to (PPC 2)

Solution: $cost = 0$, all variables are 0.

The reduced cost of (PP 8) is not negative anymore which means that (RMP 8) cannot be improved further. Moreover, all artificial variables in (RMP 8) are zero which might indicate that we have reached a feasible solution. Therefore, (RMP 8) is optimal and feasible. However, column generation does not return an integer solution. The values are as follows:

- $x_{R_P R_L R_\top} = 0.5, x_{R_D R_\top} = 0.5, x_{R_A R_L R_\top} = 0.5, x_{R_A R_D R_P R_\top} = 0.5$

To obtain an integer solution, we use the branch-and-price technique. The first step is to select one of the non-integer variables and then branch on that variable. Here, we select $x_{R_A R_L R_\top}$ and branch in RMP on $x_{R_A R_L R_\top} \geq 1$ and $x_{R_A R_L R_\top} \leq 0$. Therefore, now we have two solution subsets for finding the optimal integer solution as shown in Figure 4.5. First, we explore the branch $x_{R_A R_L R_\top} \geq 1$ and add this inequality in (RMP 9) as a new row. (PP 8) is also extended by adding the corresponding variable $r_{R_A R_L R_\top}$ in the objective, and (PPC 2) is extended with three constraints (ii), (iii) and (iv). Constraints (ii) and (iii) indicate that if $r_{R_A R_L R_\top} = 1$, then variables b_A and b_L will also be 1. Similarly, Constraint (iv) sets $r_{R_A R_L R_\top} = 1$ if b_A and b_L are 1.

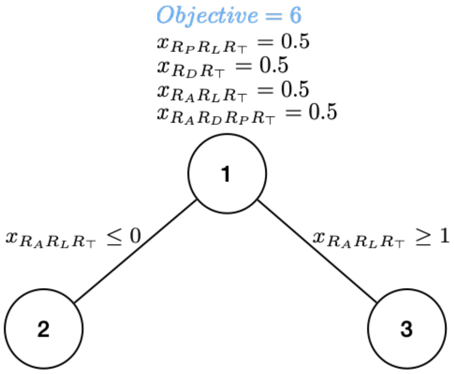

$Objective = 6$
$x_{R_P R_L R_\top} = 0.5$
$x_{R_D R_\top} = 0.5$
$x_{R_A R_L R_\top} = 0.5$
$x_{R_A R_D R_P R_\top} = 0.5$
1
$x_{R_A R_L R_\top} \leq 0$
$x_{R_A R_L R_\top} \geq 1$
2
3

Figure 4.5: Solution subsets for finding the optimal integer solution

$$\text{Min} \quad 3x_{R_P R_L R_\top} + 2x_{R_D R_\top} + 3x_{R_A R_L R_\top} + 4x_{R_A R_D R_P R_\top} + 10h_{R_A} + 10h_{R_D} + 10h_{R_P} + 10h_{R_L} + 10h_{R_\top} \tag{RMP 9}$$

Subject to

$$x_{R_A R_L R_\top} + x_{R_A R_D R_P R_\top} + h_{R_A} \geq 1$$

$$x_{R_D R_\top} + x_{R_A R_D R_P R_\top} + h_{R_D} \geq 1$$

$$x_{R_P R_L R_\top} + x_{R_A R_D R_P R_\top} + h_{R_P} \geq 1$$

$$x_{R_P R_L R_\top} + x_{R_A R_L R_\top} + h_{R_L} \geq 1$$

$$x_{R_P R_L R_\top} + x_{R_D R_\top} + x_{R_A R_L R_\top} + x_{R_A R_D R_P R_\top} + h_{R_\top} \leq 2$$

$$x_{R_A R_L R_\top} \geq 1$$

Solution: $cost = 7, x_{R_A R_L R_\top} = 1, x_{R_A R_D R_P R_\top} = 1$

Duals: $\pi_{R_D} = 2, \pi_{R_P} = 2, \pi_{R_A R_L R_\top} = 3$

$$\text{Min} \quad b_A + b_D + b_P + b_L + b_\top - 0r_{R_A} - 2r_{R_D} - 2r_{R_P} - 0r_{R_L} - 0r_{R_\top} - 3r_{R_A R_L R_\top} \tag{PP 9}$$

Subject to

$$
\begin{aligned}
r_{R_A} - b_A &\leq 0 \\
r_{R_D} - b_D &\leq 0 \\
r_{R_P} - b_P &\leq 0 \\
r_{R_L} - b_L &\leq 0 \\
b_\top - r_{R_\top} &\leq 0 \\
b_D + b_L &\leq 1 && (i) \\
b_A - b_\top &\leq 0 && && \text{(PPC 3)} \\
b_D - b_\top &\leq 0 \\
b_P - b_\top &\leq 0 \\
b_L - b_\top &\leq 0 \\
r_{R_A R_L R_\top} - b_A &\leq 0 && (ii) \\
r_{R_A R_L R_\top} - b_L &\leq 0 && (iii) \\
b_A + b_L - r_{R_A R_L R_\top} &\leq 1 && (iv)
\end{aligned}
$$

Solution: $cost = -1, b_D = 1, b_P = 1, b_\top = 1, r_{R_D} = 1, r_{R_P} = 1, r_{R_\top} = 1$

A variable $x_{R_D R_P R_\top}$ is added to (RMP 10) in corresponding inequalities.

$$
\text{Min} \quad \begin{aligned} & 3x_{R_D R_P R_\top} + 3x_{R_P R_L R_\top} + 2x_{R_D R_\top} + 3x_{R_A R_L R_\top} + 4x_{R_A R_D R_P R_\top} \\ & +10h_{R_A} + 10h_{R_D} + 10h_{R_P} + 10h_{R_L} + 10h_{R_\top} \end{aligned} \qquad \text{(RMP 10)}
$$

Subject to

$$x_{R_AR_LR_\top} + x_{R_AR_DR_PR_\top} + h_{R_A} \geq 1$$

$$x_{R_DR_PR_\top} + x_{R_DR_\top} + x_{R_AR_DR_PR_\top} + h_{R_D} \geq 1$$

$$x_{R_DR_PR_\top} + x_{R_PR_LR_\top} + x_{R_AR_DR_PR_\top} + h_{R_P} \geq 1$$

$$x_{R_PR_LR_\top} + x_{R_AR_LR_\top} + h_{R_L} \geq 1$$

$$x_{R_DR_PR_\top} + x_{R_PR_LR_\top} + x_{R_DR_\top} + x_{R_AR_LR_\top} + x_{R_AR_DR_PR_\top} + h_{R_\top} \leq 2$$

$$x_{R_AR_LR_\top} \geq 1$$

Solution: $cost = 6, x_{R_DR_PR_\top} = 1, x_{R_AR_LR_\top} = 1$

Duals: $\pi_{R_D} = 2, \pi_{R_P} = 1, \pi_{R_AR_LR_\top} = 3$

The objective of (PP 10) is updated using the dual values from (RMP 8).

$$\text{Min} \quad b_A + b_D + b_P + b_L + b_\top - 0r_{R_A} - 2r_{R_D} - 1r_{R_P} - 0r_{R_L} - 0r_{R_\top} - 3r_{R_AR_LR_\top} \tag{PP 10}$$

Subject to (PPC 3)

Solution: $cost = 0$, all variables are 0.

The reduced cost of (PP 10) is not negative anymore which means that (RMP 10) cannot be improved further. Moreover, all artificial variables in (RMP 10) are also zero and this solution has integer values. Therefore, the first branch is terminated and we computed the optimal integer solution as shown in Figure 4.6. Since we do not need to continue with the second branch $x_{R_AR_LR_\top} \geq 0$, the algorithm terminates and returns the solution $\sigma(x) = \{\langle\{R\}, \{D, P, \neg L\}, 1, \emptyset\rangle, \langle\{R\}, \{A, L, \neg D\}, 1, \emptyset\rangle\}$.

The *fil*-Rule creates two new nodes x_1 and x_2 with $\mathcal{L}(x_1) \leftarrow \{D, P, \neg L\}$, $\mathcal{L}(x_2) \leftarrow \{A, L, \neg D\}$, $\sharp x_1 \leftarrow 1$ and $\sharp x_2 \leftarrow 1$.

The e-Rule creates edges $\langle x, x_1\rangle$ and $\langle x, x_2\rangle$ with $\mathcal{L}(\langle x, x_1\rangle) \leftarrow \{R\}$ and $\mathcal{L}(\langle x, x_2\rangle) \leftarrow \{R\}$.

Therefore, the concept $\mathsf{HappyFather}$ is satisfiable.

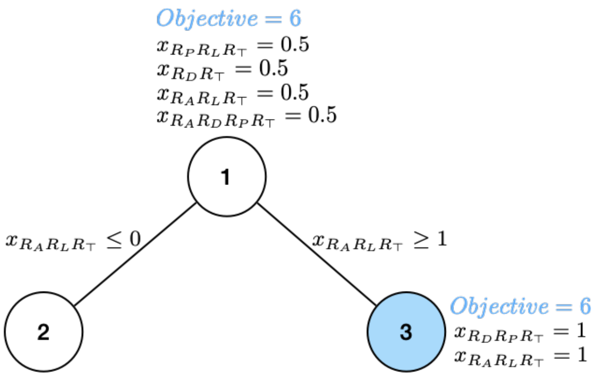

Objective = 6
$x_{R_P R_L R_\top} = 0.5$
$x_{R_D R_\top} = 0.5$
$x_{R_A R_L R_\top} = 0.5$
$x_{R_A R_D R_P R_\top} = 0.5$
1
$x_{R_A R_L R_\top} \leq 0$
$x_{R_A R_L R_\top} \geq 1$
2
3
Objective = 6
$x_{R_D R_P R_\top} = 1$
$x_{R_A R_L R_\top} = 1$

Figure 4.6: The optimal integer solution found at node 3 after applying the branch-and-price technique

4.4 Proof of Correctness

In this section we present a tableau for DL $\mathcal{SHOIQ}$ and proof of the algorithm's termination, soundness and completeness.

4.4.1 Soundness and Completeness of Algebraic Module

All number restrictions and nominals are converted into linear inequalities and added to RMP. Other axioms, such as universal restrictions, role hierarchy, subsumption and disjointness, are embedded in PP. In case of feasible inequalities, the branch-and-price algorithm returns a solution set that contains valid partition elements. Since the branch-and-price algorithm satisfies all the axioms embedded in RMP and PP, this solution is sound. Moreover, it is also complete because CPLEX is used to solve linear inequalities and it does not overlook any possible solution. Since the ILP feasibility test relies on a limited number of variables to determine the integer optimal solution, its best-case time complexity is polynomial with respect to the number of inequalities, as discussed in [76].

Proposition 34. *For a set of inequalities, the algorithm either generates an optimal solution which satisfies all inequalities or detects infeasibility.*

4.4.2 Proof of the algorithm's termination, soundness and completeness

Our proofs are guided by the ones given in [41] but address the algebraic part of this calculus correspondingly. Some of the completion rules given in Table 4.2 are very similar to the ones presented in [41]. Our rules dealing with inverse roles, QCRs, and nominals are very different due to the nature of our hybrid calculus that involves a tableau and arithmetic module. The correctness of our calculus relies on the correctness of the arithmetic module.

Lemma 35. *When started with a $\mathcal{SHOIQ}$ knowledge base $(\mathcal{T}, \mathcal{R})$, the algorithm terminates and is worst-case double exponential.*

Proof. Termination is a consequence of the following properties of the expansion rules:

1. Since a partitioning $\mathcal{P}$ is a power set of Q that contains all subsets of Q except the empty set, the size of $\mathcal{P}$ is bounded by $2^{\sharp Q} - 1$ where $\sharp Q$ is the size of Q. While the computational complexity of this process is exponential, it is important to note that $\mathcal{P}$ is computed only once for each node.

2. The worst-case complexity of the proposed algorithm is double exponential, due to the utilization of the branch-and-price method and CPLEX. However, it is noteworthy that even though the worst-case complexity of the branch and price algorithm is exponential, the average complexity is considerably lower. Furthermore, due to the fixed number ($2^{\sharp Q} - 1$) of variables, the solution for the linear inequalities can be computed in polynomial time in the best-case scenario. Moreover, it does not affect the termination of the expansion rules.

3. The blockable nodes have a tree-shaped structure and there can be only one predecessor node of the blockable node. Moreover, an upper bound on the number of new nominal nodes that can be added to G by the ni-Rule is also fixed. This is crucial for termination and for preventing the yo-yo problem that can occur due to the interaction among existing nominal nodes, inverse roles, and number restrictions (see [41, 52] for details).

4. The nom_{merge}-Rule, the $reset$-Rule and the $\leq_{nom}$-Rule are the shrinking rules that merge node labels, redirect edges and remove blockable nodes where necessary. As mentioned above, the blockable nodes form a tree-structure, therefore, when a node y is merged into a node x, or y is reset, these rules do not remove y or any of its predecessors. The $reset$-Rule is invoked by adding a new number restriction or a new nominal to the label of a node. Therefore, the maximum number of times the $reset$-Rule can fire for a node is bounded by the size of Q.

5. Each rule except the shrinking rules extends the completion graph by adding new nodes or extending node labels without removing nodes or elements from node.

6. The *fil*-Rule and the ni-Rule are the generating rules. New nodes are only added by these two generating rules and these rules can only be triggered once for each of a given concept for a node x.

   The *fil*-Rule can only be triggered once for a concept $\exists R.C \in \mathcal{L}(x)$ or $\geq nR.C \in \mathcal{L}(x)$. Suppose a neighbour y of x was generated by the *fil*-rule for a concept $\exists R.C \in \mathcal{L}(x)$ and $\mathcal{L}(x, y) = \{R\}$ is added by the e-Rule. Similarly, for a concept $\geq nR.C \in \mathcal{L}(x)$, the *fil*-rule generates m proxy individuals $y_1, ..., y_m$, which are R-neighbours of x and each represents a partition element of M_i individuals such that $\sum_{i=1}^{m} M_i = n$ and $C \in \mathcal{L}(y_i)$ for $1 \leq i \leq m$. If y is later merged in some other node z by the nom_{merge}-rule, then an R-edge toward z will be created and labels of both nodes will be merged. Therefore, there will always be some R-neighbour of x with C in its label. Hence, the *fil*-Rule cannot be applied to x for a concept $\exists R.C$ or $\geq nR.C \in \mathcal{L}(x)$ again.

   The ni-Rule introduces a set of n new nominal nodes $y_1, ..., y_n$ if applied for a concept $\leq nR.C \in \mathcal{L}(x)$ with $\mathcal{L}(y_i) = \{C\}$ for $1 \leq i \leq n$. The ni-Rule can also be triggered only once for each such concept. Moreover, blockable R-neighbours are merged with these new nominal nodes, and the number of these new nominal nodes is bounded to n.

7. Pairwise blocking [42] is used to prevent application of expansion rules when the construction becomes iterative. The blocking condition ensures that the blocking path contains only blockable nodes and does not contain any nominals.

8. The generating rules can not be applied to blocked nodes.

□

Lemma 36. *If the expansion rules can be applied to a $\mathcal{SHOIQ}$ knowledge base $(\mathcal{T},\mathcal{R})$ in such a way that they yield a complete and clash-free completion graph, then there exists a tableau for $(\mathcal{T},\mathcal{R})$.*

Proof. Let $G = (V, E, \mathcal{L}, \mathcal{B}, \mathcal{U})$ be a complete and clash-free completion graph resulting from the expansion rules initiated with $(\mathcal{T},\mathcal{R})$. A tableau $\mathrm{T} = (\mathbf{S}, \mathcal{L}', \mathcal{E})$ for $(\mathcal{T},\mathcal{R})$ can be derived from G as outlined below.

Drawing inspiration from Horrocks et al. [41], we employ a path construction to prove the soundness of the algorithm. Let a node $w(x)$ be the witness of a node x (i.e., $w(x)$ blocks x), and a path p as a sequence of pairs of blockable nodes of G in the form of $\langle(x_0, x'_0), ..., (x_n, x'_n)\rangle$. For p, $\mathsf{Tail}(p) := x_n$ and $\mathsf{Tail}'(p) := x'_n$, and with $\langle p \mid (x_{n+1}, x'_{n+1})\rangle$ the path is $\langle(x_0, x'_0), ..., (x_n, x'_n), (x_{n+1}, x'_{n+1})\rangle$. Therefore, the set of all paths $\mathsf{Paths}(G)$ is defined as follows:

- For each blockable node x of G that is a successor of a nominal node, $\langle(x, x)\rangle \in \mathsf{Paths}(G)$, and
- For a path $p \in \mathsf{Paths}(G)$ and a blockable successor y of $\mathsf{Tail}(p)$:
  - if y is not blocked, then $\langle p \mid (y, y)\rangle \in \mathsf{Paths}(G)$, and
  - if y is blocked and $w(y)$ blocks y, then $\langle p \mid (w(y), y)\rangle \in \mathsf{Paths}(G)$.

Due to the construction of Paths, all nodes occurring in a path are blockable, and for each $p \in \mathsf{Paths}(G)$ with $p = \langle p' \mid (x, x')\rangle$, the following facts hold:

- x' is not blocked,
- x' is blocked iff $x \neq x'$, and
- the predecessor of x' in G is not blocked, and
- the blocking condition implies $\mathcal{L}(x) = \mathcal{L}(x')$

If G does not contain any blocks, then there exists exactly one path $p \in \mathsf{Paths}(G)$ such that $\mathsf{Tail}(p) = \mathsf{Tail}'(p) = x$. However, if G contains block(s), then for an infinite tableau it constitutes a finite representation and both x and $w(x)$ occur in the same blockable tree-shaped part of the graph [41].

The set of nominal nodes in G is represented by $\mathsf{Nom}(G)$, and tableau $\mathrm{T} = (\mathbf{S}, \mathcal{L}', \mathcal{E})$ from G is defined as follows:

$$\mathsf{S} = \mathsf{Nom}(G) \cup \mathsf{Paths}(G)$$

$$\mathcal{L}'(p) = \begin{cases} \mathcal{L}(\mathsf{Tail}(p)) & \text{if } p \in \mathsf{Paths}(G) \\ \mathcal{L}(p) & \text{if } p \in \mathsf{Nom}(G) \end{cases}$$

$$\begin{aligned} \mathcal{E}(R) = &\{\langle p, q\rangle \in \mathsf{Paths}(G) \times \mathsf{Paths}(G) \mid \\ &\qquad q = \langle p \mid (x, x')\rangle \text{ and } x' \text{ is an } R\text{-successor of } \mathsf{Tail}(p) \text{ or} \\ &\qquad p = \langle q \mid (x, x')\rangle \text{ and } x' \text{ is an } \mathsf{Inv}(R)\text{-successor of } \mathsf{Tail}(q)\} \cup \\ &\quad \{\langle p, x\rangle \in \mathsf{Paths}(G) \times \mathsf{Nom}(G) \mid x \text{ is an } R\text{-neighbour of } \mathsf{Tail}(p)\} \cup \\ &\quad \{\langle x, p\rangle \in \mathsf{Nom}(G) \times \mathsf{Paths}(G) \mid \mathsf{Tail}(p) \text{ is an } R\text{-neighbour of } x\} \cup \\ &\quad \{\langle x, y\rangle \in \mathsf{Nom}(G) \times \mathsf{Nom}(G) \mid y \text{ is an } R\text{-neighbour of } x\} \end{aligned}$$

In order to show that T is a tableau for $(\mathcal{T}, \mathcal{R})$, we prove that T satisfies all properties (**P1**) - (**P13**) of tableau T (see Definition 27). This proof closely follows the one found in [41], however, it is somewhat different for (**P5**), (**P7**), (**P8**) and (**P9**).

- Since G is clash-free, (**P1**) holds for T.
- Since G is complete, (**P2**) and (**P3**) hold for T.

- For (**P4**), consider $\forall R.C \in \mathcal{L}'(x)$ and $\langle x, y \rangle \in \mathcal{E}(R)$. We consider four different cases:
  - If $\langle x, y \rangle \in \mathsf{Paths}(G) \times \mathsf{Paths}(G)$, then $\forall R.C \in \mathcal{L}(\mathsf{Tail}(x))$ and either
    - $\mathsf{Tail}'(y)$ is an R-successor of $\mathsf{Tail}(x)$, and G being a complete completion graph implies $C \in \mathcal{L}(\mathsf{Tail}'(y))$, and either $\mathsf{Tail}'(y) = \mathsf{Tail}(y)$ or $\mathcal{L}(\mathsf{Tail}'(y)) = \mathcal{L}(\mathsf{Tail}(y))$ according to the blocking condition; or
    - $\mathsf{Tail}'(x)$ is an $\mathsf{Inv}(R)$-successor of $\mathsf{Tail}(y)$, and either $\mathsf{Tail}'(y) = \mathsf{Tail}(y)$ or $\forall R.C \in \mathcal{L}(\mathsf{Tail}'(x))$ according to the blocking condition. Then, since G is complete, it implies that $C \in \mathcal{L}(\mathsf{Tail}(y))$.
  - If $\langle x, y \rangle \in \mathsf{Paths}(G) \times \mathsf{Nom}(G)$, then $\forall R.C \in \mathcal{L}(\mathsf{Tail}(x))$ and y is an R-neighbour of $\mathsf{Tail}(x)$. Since the $\forall$-rule is not applicable, it implies that $C \in \mathcal{L}(y)$.
  - If $\langle x, y \rangle \in \mathsf{Nom}(G) \times \mathsf{Paths}(G)$, then $\forall R.C \in \mathcal{L}(x)$ and $\mathsf{Tail}(y)$ is an R-neighbour of x. Since G is complete, it implies that $C \in \mathcal{L}(\mathsf{Tail}(y))$.
  - If $\langle x, y \rangle \in \mathsf{Nom}(G) \times \mathsf{Nom}(G)$, then $\forall R.C \in \mathcal{L}(x)$ and y is an R-neighbour of x, and G being a complete completion graph implies that $C \in \mathcal{L}(y)$.

  In all four cases, according to the definition of $\mathcal{L}'$, we have $C \in \mathcal{L}'(y)$.

- (**P6**) is similar to (**P4**).

- For (**P5**) and (**P7**), consider $\exists R.C \in \mathcal{L}'(x)$ or $\geq nR.C \in \mathcal{L}'(x)$ for some $x \in \mathbf{S}$. We are handling $\exists R.C$ as $\geq 1R.C$.
  - If $x \in \mathsf{Paths}(G)$, then G being a complete completion graph implies the existence of m proxy individuals $y_1, ..., y_m$, which are R-neighbours of $\mathsf{Tail}(x)$ with $y_i \neq y_j$ for each $i \neq j$, and each represents a partition element of M_i individuals such that $\sum_{i=1}^{m} M_i = n$ and $C \in \mathcal{L}(y_i)$ for $1 \leq i \leq m$. By construction, each y_i corresponds to a $t_i \in \mathbf{S}$ with $t_i \neq t_j$, for each $i \neq j$:

  - ∗ if y_i is blockable, then it can be blocked if it is a successor of $\mathsf{Tail}(x)$. The pair construction in paths ensures that $\langle p \mid (w(y_i), y_i)\rangle \neq \langle p \mid (w(y_j), y_j)\rangle$ even if $w(y_i) = w(y_j)$ for some $i \neq j$.
    - ∗ if y_i is a nominal node, then $\langle x, y_i\rangle \in \mathcal{E}(R)$.
  - If $x \in \mathsf{Nom}(G)$, then G being a complete completion graph implies the existence of m proxy individuals $y_1, ..., y_m$, which are safe R-neighbours of x with $y_i \neq y_j$ for each $i \neq j$, and each represents a partition element of M_i individuals such that $\sum_{i=1}^{m} M_i = n$ and $C \in \mathcal{L}(y_i)$ for $1 \leq i \leq m$. The safe R-neighbours are used to ensure that enough R-neighbours are generated for nominal nodes. By construction, each y_i corresponds to a $t_i \in \mathbf{S}$ with $t_i \neq t_j$, for each $i \neq j$:
    - ∗ if y_i is blockable, then it cannot be blocked, as it is a safe R-neighbour of x. The pair construction in paths ensures that $\langle p \mid (y_i, y_i)\rangle \in \mathbf{S}$ and $\langle x, \langle p \mid (y_i, y_i)\rangle\rangle \in \mathcal{E}(R)$.
    - ∗ if y_i is a nominal node, then $\langle x, y_i\rangle \in \mathcal{E}(R)$.

Hence, all t_i are distinct, and by construction, $C \in \mathcal{L}'(t_i)$ for each $1 \leq i \leq n$.

- Suppose $\leq nR.C \in \mathcal{L}'(x)$ for some $x \in \mathbf{S}$ and $\sharp R^T(x, C) \leq n$ is violated. This means that we have m proxy individuals $y_1, ..., y_m$, which are R-neighbours of x and each represents a partition element of M_i individuals such that $\sum_{i=1}^{m} M_i > n$ and $C \in \mathcal{L}(y_i)$ for $1 \leq i \leq m$. However, this cannot happen because: 1) The algorithm generates a solution after making sure that all at-least and at-most restrictions for x are satisfied. 2) Since G is clash-free, it implies that for each $\leq nR.C \in \mathcal{L}(x)$ we can have at-most m proxy individuals $y_1, ..., y_m$ each representing a partition element of M_i individuals such that $\sum_{i=1}^{m} M_i \leq n$ with $C \in \mathcal{L}(y_i)$. Moreover, each $y \in \mathbf{S}$ with $\langle x, y\rangle \in \mathcal{E}(R)$ corresponds to an R-neighbour y_i of x or $\mathsf{Tail}(x)$. Hence, **(P8)** is satisfied.

- Suppose $\leq nR.C \in \mathcal{L}'(x)$ for some $x \in \mathbf{S}$ and there exists R-neighbour y of either x in case $x \in \mathsf{Nom}(G)$ or $\mathsf{Tail}(x)$ in case $x \in \mathsf{Paths}(G)$ and $\{C, \neg C\} \cap \mathcal{L}(y) = \emptyset$. This means either $\forall R.(C \sqcup \neg C) \notin \mathcal{U}(x)$ or the algorithm generated an incomplete solution. However, this cannot happen because:

1. If $\forall R.(C \sqcup \neg C) \notin \mathcal{U}(x)$, then it would make the ch-Rule applicable to node x in G but that is not possible because G is complete.
2. According to Proposition 34, algebraic reasoning is sound and complete. Therefore, (**P9**) holds for T.

- (**P10**) and (**P11**) hold due to the definition of R-successor and R-neighbour. Furthermore, for (**P11**), consider $\langle x, y\rangle \in \mathcal{E}(R)$ in T that implies $\{\langle x, y\rangle \in E \mid \mathcal{L}(\langle x, y\rangle) \cap \{R\}\} \neq \emptyset$ in G. Since G is complete, it implies that $\{\langle y, x\rangle \in E \mid \mathcal{L}(\langle y, x\rangle) \cap \{\mathsf{Inv}(R)\}\} \neq \emptyset$ (due to the e-Rule) and consequently $\langle y, x\rangle \in \mathcal{E}(\mathsf{Inv}(R))$.

- For (**P12**), consider $o \in \mathcal{L}'(x) \cap \mathcal{L}'(y)$ where $x \neq y$, and assume there are two corresponding distinct nodes x and y in G with $o \in \mathcal{L}(x) \cap \mathcal{L}(y)$ for some nominal $o \in N_o$. In this case, the nom_{merge}-Rule would need to be fired but that is not possible because G is complete. Hence, (**P12**) holds for T.

- Since the partition elements of $\mathcal{P}$ are semantically pairwise disjoint, i.e., if $e, e' \in \mathcal{P}$, $e \neq e'$ then $e^{\mathcal{I}} \cap (e')^{\mathcal{I}} = \emptyset$, and due to the nominal semantics $\sharp\{o\}^{I} = 1$, the algebraic reasoner assigns the nominal o to only one partition element e. Therefore, the cardinality of e will always be 1. In addition, since nominals always exist, the nominal nodes are never removed through pruning. Hence, (**P13**) always holds.

□

Lemma 37. *If there exists a tableau of a $\mathcal{SHOIQ}$ knowledge base $(\mathcal{T}, \mathcal{R})$, then, the expansion rules can be applied to a $\mathcal{SHOIQ}$ knowledge base $(\mathcal{T}, \mathcal{R})$ in such a way that they yield a complete and clash-free completion graph.*

Proof. Let $\mathrm{T} = (\mathbf{S}, \mathcal{L}', \mathcal{E})$ be a tableau for $(\mathcal{T}, \mathcal{R})$. We use T to trigger the application of the expansion rules such that they yield a complete and clash-free completion graph $G = (V, E, \mathcal{L}, \mathcal{B}, \mathcal{U})$.

A function π is employed to map the nodes in G to elements of S in a manner that ensures the following properties hold consistently throughout the execution of the tableau algorithm for all $x, y \in V$ and $R \in N_R$:

A1. $\mathcal{L}(x) \subseteq \mathcal{L}'(\pi(x))$,

A2. if $\langle x, y\rangle \in E$ and $R \in \mathcal{L}(\langle x, y\rangle)$, then $\langle \pi(x), \pi(y)\rangle \in \mathcal{E}(R)$,

A3. $x \neq y$ implies $\pi(x) \neq \pi(y)$,

A4. if $\langle y, x\rangle \in E$ and $R^- \in \mathcal{L}(\langle y, x\rangle)$, then $\langle x, \{R^-\}\rangle \in \mathcal{B}(y)$, and according to (**A2**), $\langle \pi(y), \pi(x)\rangle \in \mathcal{E}(R^-)$, and

A5. if $\leq nR.C \in \mathcal{L}(x)$, $\forall R.\{C \sqcup \neg C\} \in \mathcal{U}(x)$ and $R \in \mathcal{L}(\langle x, y\rangle)$, then $\langle \pi(x), \pi(y)\rangle \in \mathcal{E}(R)$, $D \in \mathcal{L}'(\pi(x))$ for some $D \in \{C, \neg C\}$.

We show by applying the expansion rules defined in Table 4.2 in order to obtain G, the properties of mapping π are not violated.

- We initialize π as follows: the initial completion graph contains a node u_i, for each nominal $o_i \in N_o$, with $\mathcal{L}(u_i) = \{o_i\}$, and therefore, $\pi(u_i) = x_i$ for $x_i \in$ **S** with $o_i \in \mathcal{L}'(x_i)$. (**P13**) implies that x_i must exist.

- The $\sqcap$-Rule, $\sqcup$-Rule, $\forall$-Rule and $\forall_+$-Rule extend the label of a node x without violating π properties due to (**P2**) - (**P4**) and (**P6**) of T.

- **The** nom_{merge}**-Rule:** If $o \in \mathcal{L}(x) \cap \mathcal{L}(y)$ for some nominal $o \in N_o$, then $o \in \mathcal{L}'(\pi(x)) \cap \mathcal{L}'(\pi(y))$. Since T is a tableau, (**P12**) and (**P13**) imply $\pi(x) = \pi(y)$. Therefore, the nom_{merge}-Rule can be applied to merge nodes x and y without violating π properties.

- **The** $inverse$**-Rule:** Since T is a tableau, (**P11**) implies that if $\langle \pi(x), \pi(y)\rangle \in \mathcal{E}(R)$, then there is a back edge $\langle \pi(y), \pi(x)\rangle \in \mathcal{E}(\mathsf{Inv}(R))$. The $inverse$-Rule adds a tuple $\langle x, \{R^-\}\rangle$ to $\mathcal{B}(y)$ if $\forall R^-.C \in \mathcal{L}(y)$. Since the $inverse$-Rule only adds information about an edge that already exists and this information is only used in AR, it does not violate π properties.

- **The $reset$-Rule:** This rule is invoked when new number restrictions or nominals are added to the label of the node. The $reset$-Rule removes the blockable successor nodes, and the fil-Rule and the e-Rule create the new nodes and edges by also satisfying these new number restrictions and nominals. Therefore, the $reset$-Rule does not violate the properties of T or π.

- **The ch-Rule:** If $\leq nR.C \in \mathcal{L}(x)$ and $R \in \mathcal{L}(\langle x, y\rangle)$, then $\leq nR.C \in \mathcal{L}'(\pi(x))$ and $\langle \pi(x), \pi(y)\rangle \in \mathcal{E}(R)$ due to properties of π. Since T is a tableau, (**P9**) implies $\{C, \neg C\} \cap \mathcal{L}'(\pi(x)) \neq \emptyset$. Hence, the ch-Rule adds $\forall R.\{C \sqcup \neg C\} \in \mathcal{U}(x)$ and the algorithm generates a solution in order to add an appropriate concept $D \in \{C, \neg C\}$ to $\mathcal{L}(y)$ such that $\mathcal{L}(y) \subseteq \mathcal{L}'(\pi(y))$ holds.

- **The fil-Rule:** If $\exists R.C \in \mathcal{L}(x)$ or $\geq nR.C \in \mathcal{L}(x)$, then $\exists R.C \in \mathcal{L}'(\pi(x))$ or $\geq nR.C \in \mathcal{L}'(\pi(x))$ due to properties of π. The numerical restrictions imposed by existential restrictions and nominals along with the number restrictions are encoded into inequalities. The solution σ is returned by the algorithm that defines the distribution of fillers by satisfying the inequalities. The fil-Rule creates a node for each corresponding partition element returned by the algebraic reasoner. Since T is a tableau, (**P5**) and (**P7**) imply that there exist the individuals of **S** satisfying these existential and at-least restrictions. Therefore, the fil-Rule does not violate properties of T or π.

- **The e-Rule:** For each $\exists R.C \in \mathcal{L}(x)$ or $\geq nR.C \in \mathcal{L}(x)$, we have $\exists R.C \in \mathcal{L}'(\pi(x))$ or $\geq nR.C \in \mathcal{L}'(\pi(x))$ that means there exist $\pi(y_i) \in \mathbf{S}$, $C \in \mathcal{L}'(\pi(y_i))$ and $\langle \pi(x), \pi(y_i)\rangle \in \mathcal{E}(R)$. Since T is a tableau, (**P10**) implies $\langle \pi(x), \pi(y_i)\rangle \in \mathcal{E}(S)$ if $R \sqsubseteq_* S \in \mathcal{R}$ and (**P11**) implies $\langle \pi(y_i), \pi(x)\rangle \in \mathcal{E}(\mathsf{Inv}(R))$. The e-Rule is applied to connect x to its fillers, say y_i, by creating edges $\langle x, y_i\rangle \in E$ between them according to the solution σ. The e-Rule merges R into $\mathcal{L}(\langle x, y_i\rangle)$ and $\{\mathsf{Inv}(R) \mid R \in \mathsf{R}\}$ into $\mathcal{L}(\langle y_i, x\rangle)$ and all S with $R \sqsubseteq_* S \in \mathcal{R}$ into $\mathcal{L}(\langle x, y_i\rangle)$ and $\mathsf{Inv}(S)$ into $\mathcal{L}(\langle y_i, x\rangle)$ without violating π properties. Moreover, (**P10**) and (**P11**) of T are also preserved.

- **The ni-Rule:** If the ni-Rule is applied to x with $\leq nR.C \in \mathcal{L}(x)$, it introduces n new nominal nodes $z_1, ..., z_n$ with $C \in \mathcal{L}(z_i)$ for $1 \leq i \leq n$. Since T is a tableau, (**P8**) implies $\sharp R^T(x, C) \leq n$. As the upper bound on the number of new nominal nodes is fixed, then together with the $\leq_{nom}$-Rule, the ni-Rule does not violate properties of T or π.

- **The $\leq_{nom}$-Rule:** If $\leq nR.C \in \mathcal{L}(x)$, then $\leq nR.C \in \mathcal{L}'(\pi(x))$. Since T is a tableau, (**P8**) implies $\sharp R^T(x, C) \leq n$. If the $\leq_{nom}$-Rule is applicable to x, then it implies that $\sharp R^T(x, C) > n$ and there must be an R-neighbour y of x with $C \in \mathcal{L}(y)$ and the nominal nodes $z_1, ..., z_n$ with $C \in \mathcal{L}(z_i)$ for $1 \leq i \leq n$. Moreover, there must be two nodes y and z with $z \in \{z_1, ..., z_n\}$ such that $\pi(y) = \pi(z)$ because (**P8**) will not hold otherwise. Therefore, the $\leq_{nom}$-Rule can be applied without violating π properties.

□

Chapter 5

Cicada - An Algebraic Tableau Reasoner for $\mathcal{SHOIQ}$

This chapter introduces Cicada, a prototype system designed to implement the algebraic tableau calculus, proposed in Chapter 4, as proof of concept. Given that the satisfiability problem for $\mathcal{SHOIQ}$ is NExpTime-complete, addressing this challenge efficiently necessitates the incorporation of a wide range of optimization techniques. In response to this requirement, our implementation also integrates a few optimization techniques, complemented by column generation and branch-and-price techniques. Section 5.1 provides an overview of the reasoner architecture, explaining the foundational structure of the system. Subsequently, Sections 5.2 and 5.3 delve into a comprehensive discussion of all the components of the Tableau Module (TM) and the Algebraic Module (AM). Sequence diagrams are used to illustrate the sequential flow of activities within each component of a system. A sequence diagram is a kind of interaction diagram which shows interaction among processes and the order in which they interact with each other.

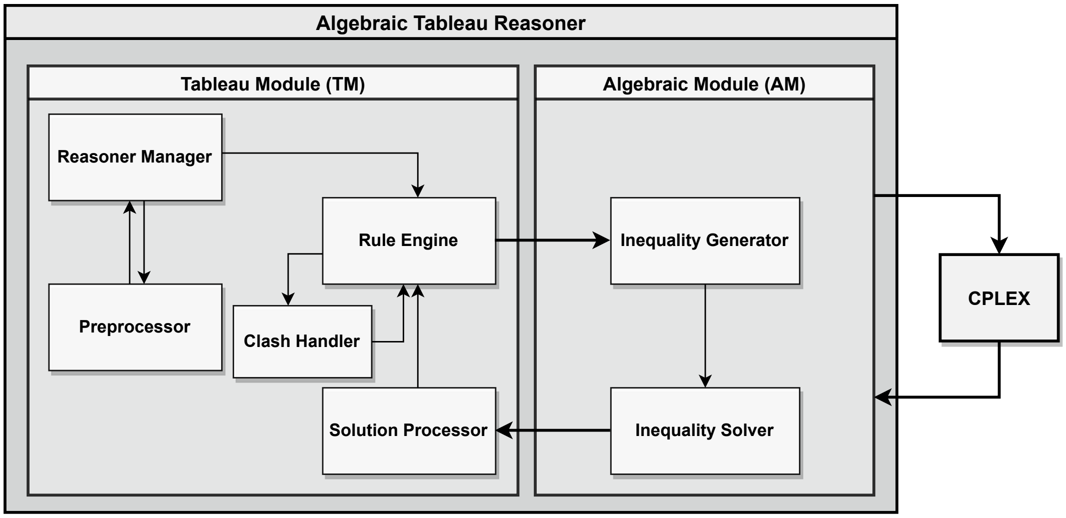


Figure 5.1: Algebraic Tableau Reasoner Architecture

5.1 Reasoner Architecture

Cicada is implemented using JAVA (JRE 1.8) and OWL-API (5.1.0)[1]. OWL-API is a Java API for working with OWL ontologies. IBM CPLEX (Optimization software)[2] is used to solve our Integer Linear Programming (ILP) formulation. The reasoner is divided into two main modules (as shown in Figure 5.1):

1. Tableau Module (TM)
2. Algebraic Module (AM)

The Tableau Module (TM) initiates its operation by loading the ontology and initializing the reasoning process. It does some preprocessing and applies expansion rules (see Section 4.3.2) for creating a complete and clash-free completion graph. In situations where an algebraic reasoner is required, TM transmits all numerical restrictions and other related information to the Algebraic Module (AM). This collaborative interaction ensures that the Algebraic Module receives the relevant data required for its specialized reasoning tasks.

[1] https://owlapi.sourceforge.net/

[2] CPLEX is an optimization software tool for solving linear optimization problems. https://www.ibm.com/analytics/cplex-optimizer

The Algebraic Module (AM) handles all numerical restrictions through Integer Linear Programming (ILP). It formulates inequalities and addresses them using the column generation and branch-and-price technique. The potential outcomes involve either the return of an optimal integer solution or the identification of a clash, signalling the absence of a feasible solution. Upon receiving the solution from AM, TM systematically processes the results. Subsequent sections elaborate on both modules by providing a more detailed understanding of their functionalities and contributions. Figure 5.2 presents the structure of a system by depicting the system's classes, their attributes, methods, and the relationships among objects.

5.2 Tableau Module (TM)

The Tableau Module (TM) commences its operation by loading the ontology (from the .owl file) and initiating the reasoning process through the instantiation of a new Reasoner instance. Subsequently, TM systematically processes the ontology, configuring its internal settings. All axioms undergo transformation to their negation normal forms, and several preprocessing optimization techniques are applied.

TM then constructs a completion graph (`CGraph`), comprising nodes and edges. Each node encapsulates information regarding its label and cardinality, along with details about incoming and outgoing edges. Similarly, each edge holds information about its label and the corresponding nodes. The expansion rules are employed to extend the completion graph `CGraph`.

During the consistency checking phase, TM incorporates several optimization techniques. Upon receiving a solution from the Algebraic Module (AM), TM processes it and applies the necessary expansion rules to generate proxy nodes. In case of a clash, TM employs backtracking mechanisms guided by dependencies to explore alternative options.

TM consists of five subcomponents, each contributing to the overall functionality and efficiency of the module.

1. Reasoner Manager

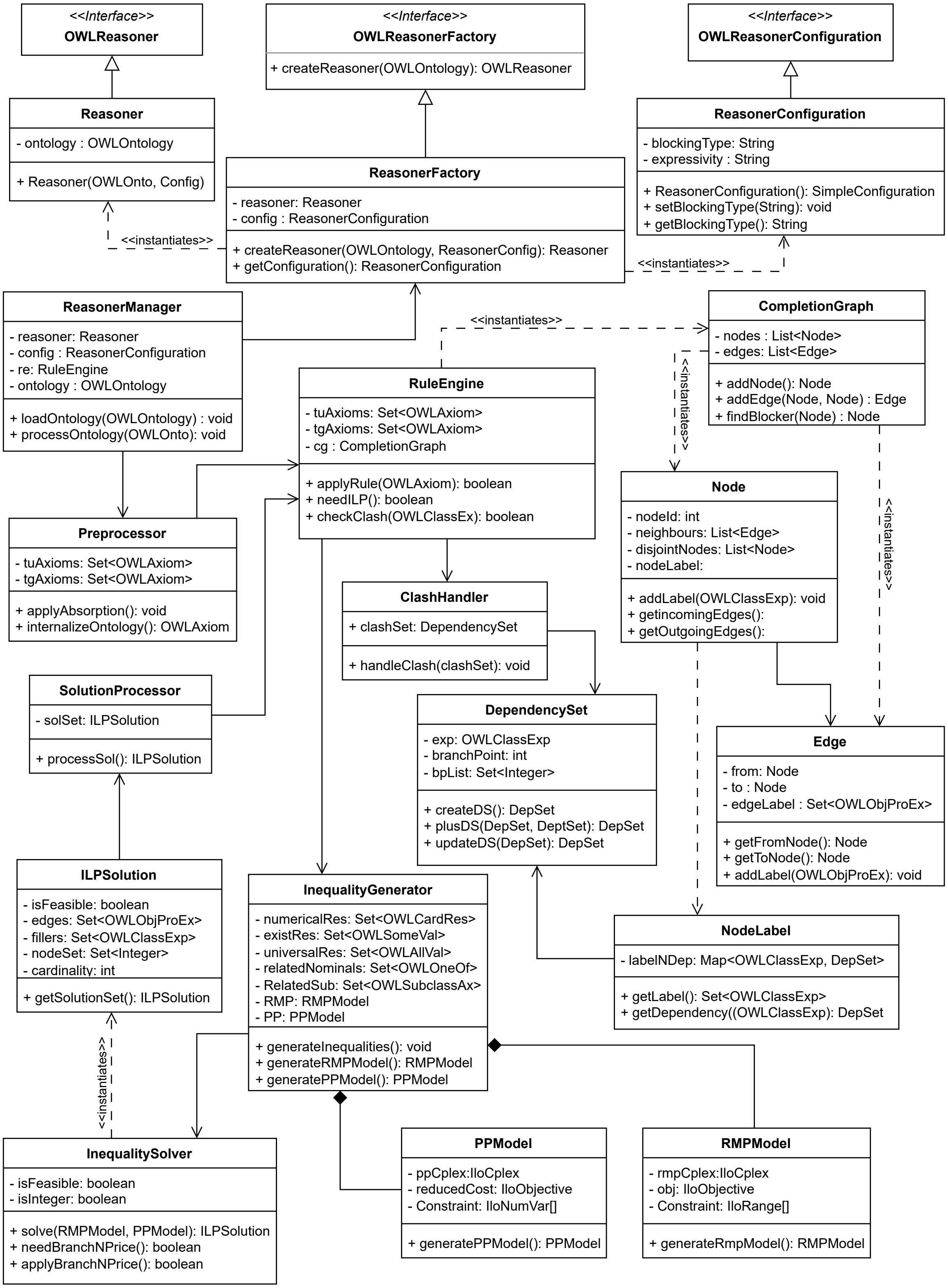

<<Interface>>
OWLReasoner
<<Interface>>
OWLReasonerFactory
+ createReasoner(OWLOntology): OWLReasoner
<<Interface>>
OWLReasonerConfiguration
Reasoner
- ontology : OWLOntology
+ Reasoner(OWLOnto, Config)
ReasonerFactory
- reasoner: Reasoner
- config : ReasonerConfiguration
+ createReasoner(OWLOntology, ReasonerConfig): Reasoner
+ getConfiguration(): ReasonerConfiguration
ReasonerConfiguration
- blockingType: String
- expressivity : String
+ ReasonerConfiguration(): SimpleConfiguration
+ setBlockingType(String): void
+ getBlockingType(): String
<<instantiates>>
<<instantiates>>
ReasonerManager
- reasoner: Reasoner
- config : ReasonerConfiguration
- re: RuleEngine
- ontology : OWLOntology
+ loadOntology(OWLOntology) : void
+ processOntology(OWLOnto): void
<<instantiates>>
CompletionGraph
- nodes : List<Node>
- edges: List<Edge>
+ addNode(): Node
+ addEdge(Node, Node) : Edge
+ findBlocker(Node) : Node
RuleEngine
- tuAxioms: Set<OWLAxiom>
- tgAxioms: Set<OWLAxiom>
- cg : CompletionGraph
+ applyRule(OWLAxiom): boolean
+ needILP(): boolean
+ checkClash(OWLClassEx): boolean
<<instantiates>>
<<instantiates>>
Node
- nodeId: int
- neighbours: List<Edge>
- disjointNodes: List<Node>
- nodeLabel:
+ addLabel(OWLClassExp): void
+ getincomingEdges():
+ getOutgoingEdges():
Preprocessor
- tuAxioms: Set<OWLAxiom>
- tgAxioms: Set<OWLAxiom>
+ applyAbsorption(): void
+ internalizeOntology(): OWLAxiom
ClashHandler
+ clashSet: DependencySet
+ handleClash(clashSet): void
SolutionProcessor
- solSet: ILPSolution
+ processSol(): ILPSolution
DependencySet
- exp: OWLClassExp
- branchPoint: int
- bpList: Set<Integer>
+ createDS(): DepSet
+ plusDS(DepSet, DeptSet): DepSet
+ updateDS(DepSet): DepSet
Edge
- from: Node
- to : Node
- edgeLabel : Set<OWLObjProEx>
+ getFromNode(): Node
+ getToNode(): Node
+ addLabel(OWLObjProEx): void
ILPSolution
- isFeasible: boolean
- edges: Set<OWLObjProEx>
- fillers: Set<OWLClassExp>
- nodeSet: Set<Integer>
- cardinality: int
+ getSolutionSet(): ILPSolution
InequalityGenerator
- numericalRes: Set<OWLCardRes>
- existRes: Set<OWLSomeVal>
- universalRes: Set<OWLAllVal>
- relatedNominals: Set<OWLOneOf>
- RelatedSub: Set<OWLSubclassAx>
- RMP: RMPModel
- PP: PPModel
+ generateInequalities(): void
+ generateRMPModel(): RMPModel
+ generatePPModel(): PPModel
NodeLabel
- labelNDep: Map<OWLClassExp, DepSet>
+ getLabel(): Set<OWLClassExp>
+ getDependency((OWLClassExp): DepSet
<<instantiates>>
InequalitySolver
- isFeasible: boolean
- isInteger: boolean
+ solve(RMPModel, PPModel): ILPSolution
+ needBranchNPrice(): boolean
+ applyBranchNPrice(): boolean
PPModel
- ppCplex:IloCplex
- reducedCost: IloObjective
- Constraint: IloNumVar[]
+ generatePPModel(): PPModel
RMPModel
- rmpCplex:IloCplex
- obj: IloObjective
- Constraint: IloRange[]
+ generateRmpModel(): RMPModel

Figure 5.2: System's classes, their attributes, methods, and the relationships among objects

2. Preprocessor
3. Rule Engine
4. Clash Handler
5. Solution Processor

5.2.1 Reasoner Manager

The Reasoner Manager (RM) assumes the pivotal role of coordinating and overseeing various tasks across different components. RM reads an ontology in the form of a .owl file, which can be seamlessly integrated into the system by providing the corresponding file link. RM initiates the creation of an instance of the `OWLOntologyManager` using the `OWLManager` class from the OWL API, as outlined in Algorithm 5.5. This establishes a structured and efficient mechanism for managing and manipulating ontological information within the system.

Algorithm 5.5 createOWLOntologyManager()

```
Output: An instance of OWLOntologyManager
$manager$ ← create OWLOntologyManager using OWLManager
```

Algorithm 5.6 loadOntology($fileName$)

```
Input: A string representing the path to the ontology file $fileName$
Output: $ontology$
$ontology$ ← instance of OWLOntology
$file$ ← create File instance using $fileName$
$ontology$ ← load ontology from $file$
```

Following this, RM proceeds to load the ontology file into an `OWLOntology` object by implementing Algorithm 5.6. This involves the effective transfer of the ontological content from the file into a structured and manipulable representation within the system. The utilization of the `OWLOntology` object facilitates further operations and reasoning tasks on the ontological data.

Subsequently, RM initiates the creation of the reasoner by invoking the $createReasoner()$ method of the `ReasonerFactory`. The `ReasonerFactory` class implements the

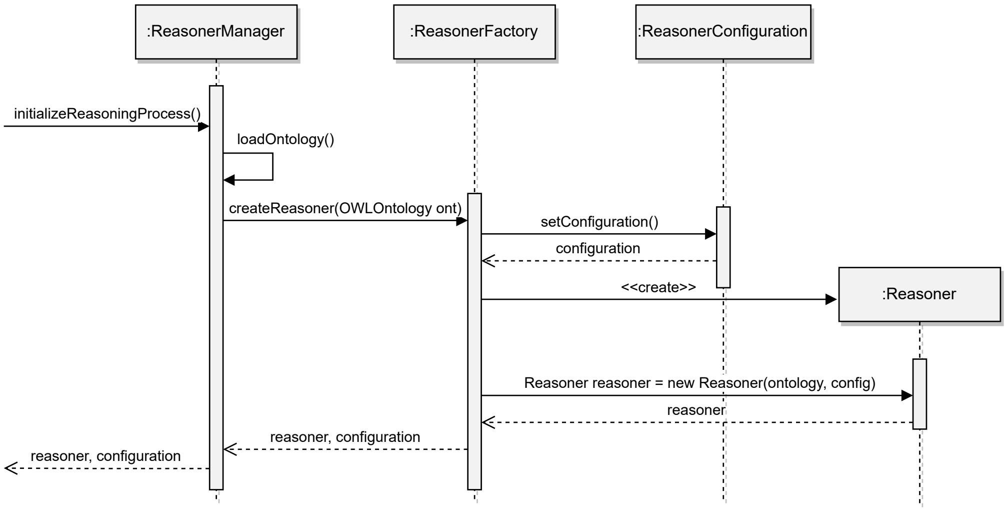


Figure 5.3: Sequence diagram for initialization of reasoning process

`OWLReasonerFactory` interface of the OWL API. As part of this process, the `ReasonerFactory` first invokes the $setConfiguration()$ method of the ReasonerConfiguration class to establish the necessary configurations.

The `ReasonerConfiguration` class, implementing the `OWLReasonerConfiguration` interface, plays a crucial role in configuring reasoning settings and determining the appropriate blocking strategy. These configurations are established after assessing the expressivity of the loaded ontology, ensuring that the reasoner is appropriately configured to handle the specific characteristics and complexities of the ontology. For $\mathcal{SHOIQ}$ ontologies we use pairwise blocking (see Section 3.1 for details).

Once `ReasonerFactory` gets configuration settings, it instantiates an object of the `Reasoner` class. The `Reasoner` object is created while considering both the characteristics of the loaded ontology and the specified configuration settings. This process is visualized in a sequence diagram presented in Figure 5.3.

Afterwards, RM engages the Preprocessor to handle the ontology. Following the completion of the preprocessing stage, RM forwards the processed ontology to the Rule Engine, which conducts an ontology consistency test.

Table 5.1: DL syntax and their correspondence OWL syntax

DL SHOIQ Syntax	OWL Syntax
$\top$	OWL:Thing
$\bot$	OWL:Nothing
A	OWLClass
$\neg C$	ObjectComplementOf(C)
$\{o\}$	OWLIndividual
$C \sqcap D$	ObjectIntersectionOf(C D)
$C \sqcup D$	ObjectUnionOf(C D)
R	OWLObjectProperty
$\exists R.C$	ObjectSomeValuesFrom(R C)
$\forall R.C$	ObjectAllValuesFrom(R C)
$\exists R.\{o\}$	ObjectSomeValuesFrom(R ObjectOneOf(o))
$\geq nR.C$	ObjectMinCardinality
$\leq nR.C$	ObjectMaxCardinality

5.2.2 Preprocessor

The Preprocessor undertakes several optimization techniques on the loaded ontology, represented as the `OWLOntology` object. All axioms within the ontology are assumed to be in their Negation Normal Form (NNF), wherein the negation sign ($\neg$) is exclusively positioned in front of concept names or nominals, as defined in Definition 20. The initial step of the Preprocessor involves transforming all axioms to their Negation Normal Forms.

Following this, the Preprocessor applies absorption techniques targeting concepts, roles, and nominals. This process aims to eliminate General Concept Inclusion (GCI) axioms, thereby mitigating nondeterminism. Importantly, these preprocessing transformations are executed on the `OWLOntology` object, ensuring that the original ontology file remains unaffected. Given that the loaded ontology is manipulated using the OWL API, references to our implemented procedures are made using OWL syntax. For clarity, Table 5.1 illustrates the correspondence between DL syntax and its equivalent OWL syntax.

5.2.2.1 Concept Absorption

The Preprocessor executes concept absorption, as discussed in Section 3.2.1.2. This step involves the application of optimization techniques aimed at enhancing the representation and efficiency of the ontology.

5.2.2.2 Role Absorption

Additionally, the Preprocessor applies both basic and extended role absorption techniques, as described in Section 3.2.1.2. This role absorption is also employed in the case of universal restrictions and Qualified Cardinality Restrictions (QCRs). The equivalences introduced by these axiom transformations are illustrated in Figure 5.4.

General concept inclusion axioms	Equivalent axioms after absorption
$\geq 3R.C \sqsubseteq D$	$\exists R.\top \sqsubseteq D \sqcup \leq 2R.C$
$\leq 3R.C \sqsubseteq D$	$\exists R.\top \sqsubseteq D \sqcup \geq 4R.C$
$\forall R.C \sqsubseteq D$	$\exists R.\top \sqsubseteq D \sqcup \exists R.\neg C$

Figure 5.4: Axiom equivalences used in extended role absorption

Moreover, in the presence of inverse roles, instead of rewriting $\exists R.C \sqsubseteq D$ as $\exists R.\top \sqsubseteq D \sqcup \neg(\exists R.C)$, it has been rewritten as $C \sqsubseteq \forall R^{-}.D$. Some axiom equivalences in the presence of inverse roles are shown in Figure 5.5.

General concept inclusion axioms	Equivalent axioms after absorption
$\exists R.C \sqsubseteq D$	$C \sqsubseteq \forall R^{-}.D$
$\exists R.(C_1 \sqcup C_2) \sqsubseteq D$	$C_1 \sqsubseteq \forall R^{-}.D$ and $C_2 \sqsubseteq \forall R^{-}.D$
$\exists R.(C_1 \sqcap C_2) \sqsubseteq D$	$C_1 \sqcap C_2 \sqsubseteq \forall R^{-}.D$

Figure 5.5: Axiom equivalences used in extended role absorption in the presence of inverse roles

5.2.2.3 Nominal Absorption

In the presence of nominals, the Preprocessor additionally applies techniques for nominal $OneOf$ and $HasValue$ absorption, see Section 3.2.1.3 for details. Moreover, when nominals are present in General Concept Inclusion (GCI) axioms, the Preprocessor tries to absorb these

Algorithm 5.7 createTGAxiom$(tgAx, df)$

Input: Set of OWLSubClassOfAxiom $tgAx$ representing subclass axioms, OWLDataFactory df
Output: OWLClassExpression $TGAxiom$

```
1: for all $sb \in tgAx$ do
2:     $union \leftarrow df$.getOWLObjectUnionOf($sb$.getSubClass().getComplementNNF(), $sb$.getSuperClass())
3:     $unionSet \leftarrow unionSet \cup union$
4: end for all
5: $TGAxiom \leftarrow df$.getOWLObjectIntersectionOf($unionSet$)
```

nominals to eliminate GCIs. Some axiom equivalences resulting from this process are presented in Figure 5.6. The assertion $\{a\} \sqsubseteq A$ is represented as $ClassAssertion(A\, o)$ in OWL Syntax.

General concept inclusion axioms	Equivalent axioms after absorption
$\top \sqsubseteq A \sqcup \neg\{a\}$	$\{a\} \sqsubseteq A$
$\top \sqsubseteq (A \sqcap B) \sqcup \neg\{a\}$	$\{a\} \sqsubseteq A$ and $\{a\} \sqsubseteq B$
$\top \sqsubseteq A \sqcup \neg\{a, b, c\}$	$\{a\} \sqsubseteq A$ and $\{b\} \sqsubseteq A$ and $\{c\} \sqsubseteq A$
$\top \sqsubseteq (\neg A \sqcup \neg B) \sqcup \{a\}$	$A \sqcap B \sqsubseteq \{a\}$
$\top \sqsubseteq (\neg A \sqcap \neg B) \sqcup \{a\}$	$A \sqsubseteq \{a\}$ and $B \sqsubseteq \{a\}$

Figure 5.6: Axiom equivalences used in nominal absorption

All the remaining axioms in the loaded ontology that are not absorbable are reduced to a single axiom called $\mathtt{T_GAxiom}$ such that $\mathtt{T_GAxiom} := \bigsqcap_{C \sqsubseteq D}(\neg C \sqcup D)$, as outlined in Algorithm 5.7.

The sequence diagram of ontology processing is shown in Figure 5.7.

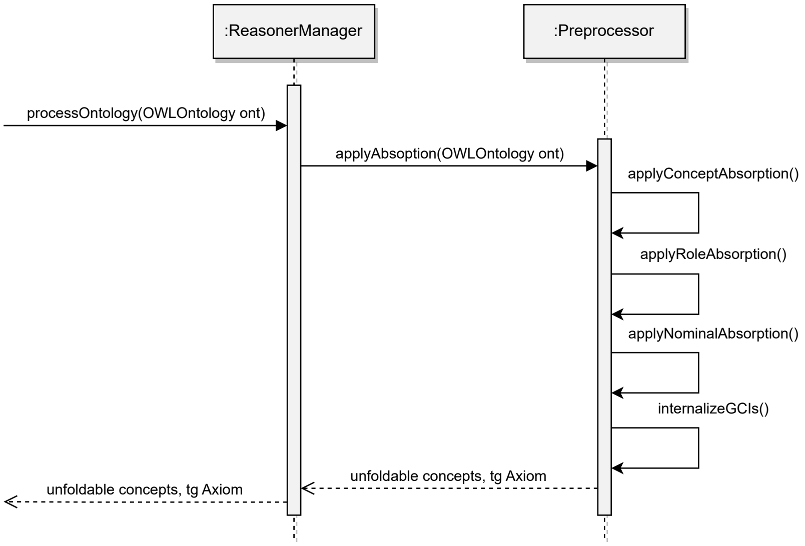

:ReasonerManager
:Preprocessor
processOntology(OWLOntology ont)
applyAbsoption(OWLOntology ont)
applyConceptAbsorption()
applyRoleAbsorption()
applyNominalAbsorption()
internalizeGCIs()
unfoldable concepts, tg Axiom
unfoldable concepts, tg Axiom

Figure 5.7: Sequence diagram for ontology processing

5.2.3 Rule Engine

The Rule Engine (RE) serves as a main component within this module. RE implements the tableau calculus presented in the previous chapter and performs the ontology consistency test. In the process of checking ontology consistency, RE constructs a completion graph (`CGraph`). `CGraph` is composed of a set of proxy nodes and a set of edges connecting these nodes. Each node within the graph encapsulates information about its label, cardinality, and details regarding all incoming and outgoing edges. Similarly, each edge retains information about its label and the corresponding nodes it connects. Notably, the nominal node's cardinality value is set to 1, while blockable nodes possess a non-negative integer cardinality.

`CGraph` is then extended by applying the expansion rules discussed in Section 4.3.2. RE assesses whether the ILP module is required. If ILP is indeed necessary, RE dispatches all related information to the Algebraic Module. This information encompasses numerical restrictions, universal restrictions, nominals, disjointness, and subsumptions. Upon receiving the solution from the Algebraic Module, RE applies the necessary expansion rules. RE also checks for a potential clash.

In case of a clash, RE invokes the Clash Handler to handle the clash. The Clash Handler determines whether the clash can be resolved. If resolution is possible, the Clash Handler communicates this to RE. In scenarios where alternative options exist, RE easily identifies the relevant nondeterministic decision and jumps back directly to that decision by using the dependency-directed backtracking technique (see Section 3.2.2.2). RE then selects the alternative option to address the clash.

If the clash cannot be handled, RE notifies the Reasoner Manager (RM) that the ontology is inconsistent. RE persists in extending the `CGraph` by applying expansion rules until a clash is detected or no expansion rule is applicable to any node in `CGraph`. The sequence diagram detailing the consistency checking and clash handling is presented in Figure 5.8. It depicts the objects, classes, and the sequence of messages exchanged between the objects required to carry out this process.

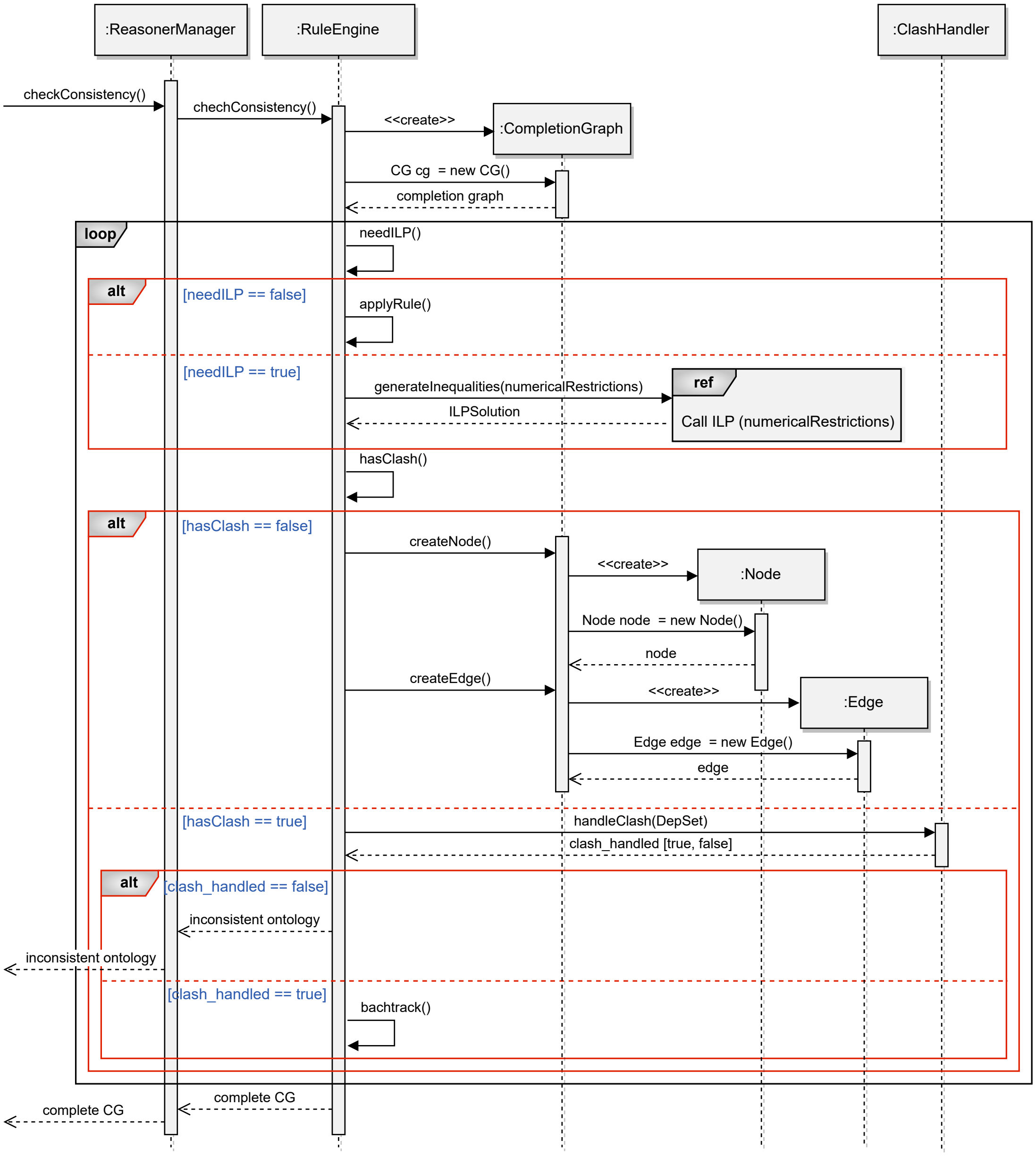


Figure 5.8: Sequence diagram for consistency checking and clash handling

Algorithm 5.8 addDependencySet($level$)

Input: Branching Level $level$

Output: DependencySet DS

```
$DS \leftarrow$ create new DependencySet using given branching $level$
$DS.branchingPoint \leftarrow level$
$DS.branchingPointList \leftarrow DS.branchingPointList \cup level$
```

Algorithm 5.9 plusDependencySets($DS1, DS2$)

Input: Two Dependency Sets $DS1$, $DS2$

Output: DependencySet DS

```
$DS \leftarrow$ create an empty DependencySet
if $DS1$ is not NULL then
    $DS \leftarrow DS \cup DS1$
end if
if $DS2$ is not NULL then
    $DS \leftarrow DS \cup DS2$
end if
```

5.2.4 Clash Handler

When the system detects a clash, it invokes the Clash Handler (CH) to determine whether the clash can be managed. As discussed in Section 3.2.2.2, each concept is associated with a dependency set. CH instantiates an object of the `DependencySet` class to represent concept dependencies.

If a concept is added as a result of a deterministic choice, it possesses an empty dependency set. However, in the case of a nondeterministic choice, a concept is added to the label of the node with the relevant dependencies. This dependency is incorporated using the $add()$ method of the `DependencySet` class, as shown in Algorithm 5.8.

In the case of a clash, the Clash Handler (CH) receives a `ClashSet`, which is a union of the dependency sets of concepts triggering the clash. This union of sets is formed using the $plus()$ method of the `DependencySet` class (see Algorithm 5.9). CH examines these dependencies, identifies the maximum one, and determines whether there are alternative choices to explore. If other options are available, CH requests the Rule Engine (RE) to directly backtrack to that decision and choose the alternative option.

Algorithm 5.10 removeLevel($DS, level$)

```
Input: Dependency Set $DS$, Branching Level $level$
Output: updated DependencySet $DS$
$DS.branchingPointList \leftarrow$ get $branchingPointList$ of $DS$
$DS.branchingPointList \leftarrow$ remove level from $DS.branchingPointList$
$maxValue \leftarrow$ get maximum value of $DS.branchingPointList$
$DS.branchingPoint \leftarrow maxValue$
```

If there are no other options at that dependency to choose, CH updates the `ClashSet` by removing that dependency and looks for alternative dependencies. CH implements Algorithm 5.10 to remove the current dependency. This process continues until either the `ClashSet` is empty or another option to explore is found. If the `ClashSet` is empty, it implies that there are no other options available. Consequently, CH generates a message indicating that the clash cannot be handled. Since there is no other option left, the system identifies this label as unsatisfiable.

CH manages three types of branchings: OR-branching, ILP-branching, and Merge-branching.

OR-branching

This type of branching occurs as a result of the $\sqcup$-Rule application. This leads to branching situations, introducing nondeterministic choices within the reasoning process. For example, we have two concepts $\leq 1R.\top \sqcup \forall R.\neg B$ and $\geq 2R.(A \sqcap B) \sqcup \geq 1R.A$. A concept $\leq 1R.\top \sqcup \forall R.\neg B$ has a dependency set $\mathcal{D}_1 = \{\emptyset\}$, then after application of the $\sqcup$-Rule, a concept $\leq 1R.\top$ is added to the label with a dependency set $\mathcal{D}_2 = \{\mathcal{D}_1 \cup 1\}$, here 1 is a dependency that is set for disjunction $\leq 1R.\top \sqcup \forall R.\neg B$. Similarly, a concept $\geq 2R.(A \sqcap B) \sqcup \geq 1R.A$ has a dependency set $\mathcal{D}_3 = \{\emptyset\}$. After application of the $\sqcup$-Rule, a concept $\geq 2R.(A \sqcap B)$ is added to the label with a dependency set $\mathcal{D}_4 = \{\mathcal{D}_3 \cup 2\}$.

When the algorithm tries to create 2 R-successors of x with A and B in their label, it identifies a clash with $\leq 1R.\top$ that has the dependency set $\mathcal{D}_2$. Therefore, the resulting `ClashSet` is the union of the dependency sets of participating concepts, i.e., $\mathcal{D}_l = \{\mathcal{D}_2 \cup \mathcal{D}_4\} = \{1, 2\}$. CH at first finds out if there are other choices to explore at dependency 2. If CH finds other options, it asks RE to jump back directly to that decision. Otherwise, CH updates `ClashSet` by removing dependency

2 and sets $\mathcal{D}_l = \{1\}$. CH then checks other options at dependency 1. The process terminates when either `ClashSet` is empty or it finds some other option to explore.

ILP-branching

ILP-branching manifests for two primary reasons:

1. **Clash with Concepts Propagated by Inverse Roles:** If a concept, propagated back in the label of a node due to the inverse roles, clashes with a concept added by the ILP solution, the Clash Handler (CH) identifies this conflict. CH then relays this information to the Rule Engine (RE). RE, in turn, backtracks to that specific point, integrates new information, and communicates with the Arithmetic Module to obtain an updated solution.

2. **Clash Triggered by ch-Rule Application:** Given that the ch-Rule is applied within the Algebraic Module, a clash may occur due to this specific branching. CH detects this clash using dependencies and notifies RE. Subsequently, RE includes information about the clash, prompting a reevaluation of numerical restrictions by calling the Algebraic Module. The Algebraic Module then takes into account the new clash-related information for an updated solution.

Merge branching

If a concept that triggered a clash is added by merging nodes, the Clash Handler (CH) takes corrective action. CH reverses the merging process and restores the `CGraph` to its previous state. Following this, CH prompts the Rule Engine (RE) to backtrack and explore alternative options if they are available. This undoing of merging ensures that the clash is appropriately addressed, and the system can consider different choices to proceed.

5.2.5 Solution Processor

To manage numerical restrictions, the Rule Engine (RE) invokes the Algebraic Module, providing all related information required to solve these numerical restrictions. The Algebraic Module subsequently returns the solution after addressing these numerical restrictions. The Solution Processor (SP) is then engaged to process this solution.

SP determines the number of proxy nodes that need to be created and how many can be reused. It refines and processes the solution before forwarding it to RE. The processed solution is then utilized by RE for the application of expansion rules and further processing. This collaborative process ensures the effective integration of solutions obtained from the Algebraic Module into the Tableau Module.

5.3 Algebraic Module (AM)

The Algebraic Module (AM) is responsible for handling all numerical restrictions using Integer Linear Programming (ILP). It formulates inequalities and addresses them through the branch-and-price technique (see Section 4.3.1 for details). This module comprises two subcomponents:

1. Inequality Generator

2. Inequality Solver

The sequence diagram shown in Figure 5.9 depicts the process of formulating inequalities and generating solutions.

5.3.1 Inequality Generator

The Inequality Generator (IG) implements Algorithm 4.1 and 4.2 in order to generate the Restricted Master Problem (RMP) and Pricing Problem (PP). IG initiates the process by initializing

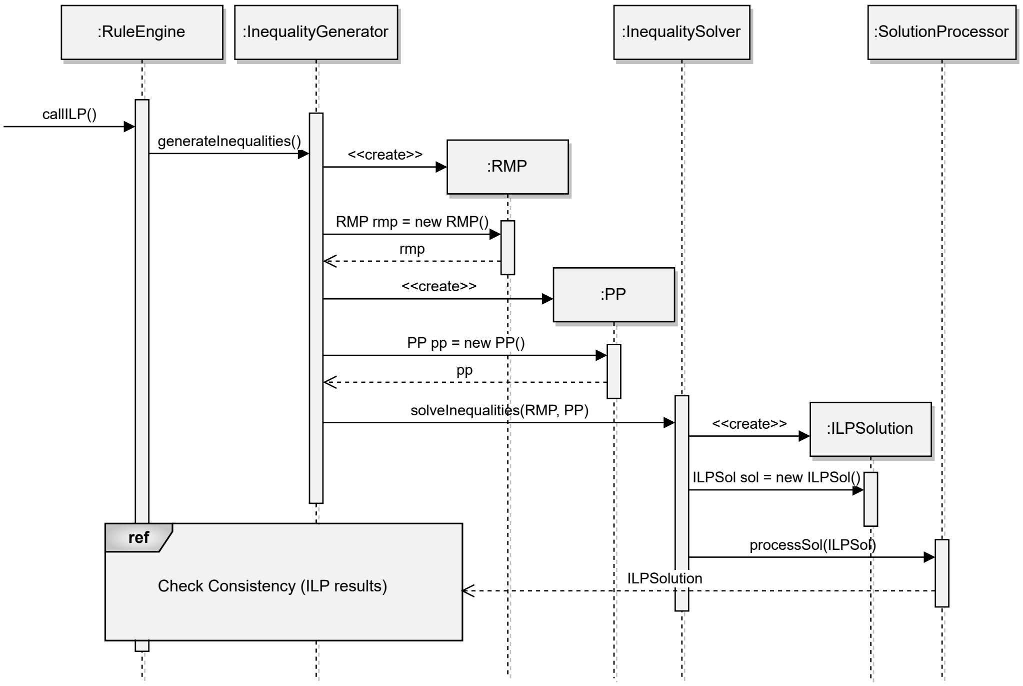


Figure 5.9: Sequence diagram for generating inequalities and ILP solution

the RMP model, as outlined in Algorithm 5.11. IG gets all numerical restrictions, existential restrictions and related nominals and generates RMP. Information related to subsumptions, disjointness, universal restrictions and role hierarchies are used to generate PP. IG implements Algorithm 5.12 for initializing the PP model.

IG uses the atomic decomposition technique to encode numerical restrictions on concepts and role fillers into inequalities. It represents the decomposition sets $Q_{\geq}$, $Q_{\leq}$ and Q_o as an array (`IloRange[] Constraint`) in RMP. On the other hand, $Q_{\forall}$ is represented in PP using arrays (`IloNumVar[] r`) and (`IloNumVar[] b`). These representations and encodings ensure that the numerical restrictions are appropriately translated into the ILP formulation.

IG sends these RMP and PP models to the Inequality Solver by invoking the $solveInequalities()$ method, which is responsible for solving the formulated ILP models.

By calling $solveInequalities()$, IG initiates the process of solving the RMP and PP, and the Inequality Solver takes on the task of finding solutions based on the encoded constraints and objectives within these ILP models.

Algorithm 5.11 initializeRMPModel($totalVar$)

```
Input: Total number of variables $totalVar$
Output: An instance of RMPModel $rmpModel$
$rmpModel \leftarrow$ create an instance of RMPModel
$rmpModel.rmpCplex \leftarrow$ initialize the CPLEX model for RMP
$rmpModel.Objective \leftarrow$ a minimization objective to the CPLEX model
$rmpModel.Constraint \leftarrow$ initialize the array of inequalities for numerical restrictions using $totalVar$
$rmpModel.H \leftarrow$ initialize the starting artificial variables
$rmpModel.X \leftarrow$ initialize the new variables for generated columns
```

Algorithm 5.12 initializePPModel($totalVar, totalQualifiers$)

```
Input: Total number of variables $totalVar$, Total number of qualifiers $totalQualifiers$
Output: An instance of PPModel $ppModel$
$ppModel \leftarrow$ create an instance of PPModel
$ppModel.ppCplex \leftarrow$ initialize the CPLEX model for PP
$rmpModel.reducedCost \leftarrow$ objective from the CPLEX model for PP
$rmpModel.R \leftarrow$ initialize the array of binary variables to ensure description logic semantics using $totalVar$
$rmpModel.B \leftarrow$ initialize the array of binary variables to ensure description logic semantics using $totalQualifiers$
```

5.3.2 Inequality Solver

The Inequality Solver (IS) plays a crucial role in solving the generated Restricted Master Problem (RMP) and Pricing Problem (PP) models. It implements Algorithm 4.3, which outlines the steps and procedures for solving ILP models.

IS is responsible for executing the necessary computations and optimizations to find solutions that satisfy the constraints and objectives encoded in the RMP and PP. It uses CPLEX, an optimization software tool, to achieve this. It ensures that the ILP models are effectively solved to provide an optimal integer solution.

In the case of a non-integer solution, IS takes corrective action by invoking the $applyBranchAndPrice()$ method, as specified in Algorithm 4.4. This method implements the branch-and-price technique to further refine the solution and attempt to find an optimal integer solution. As discussed in the previous chapter, the branch-and-price technique involves a combination of branching, where the solution space is divided into subproblems, and pricing, where new

Algorithm 5.13 processSolution($ILPSolutionSet$)

```
Input: $ILPSolutionSet$
Output: $SolutionSet$
for all $sol \in ILPSolutionSet$ do
    $solution \leftarrow$ an instance of Solution
    $solution.edges \leftarrow sol.roles$
    $solution.fillers \leftarrow sol.concepts$
    $solution.cardinality \leftarrow sol.cardinality$
    $solution.nodeSet \leftarrow sol.nodeSet$
    $SolutionSet \leftarrow SolutionSet \cup solution$
end for all
```

variables are introduced to the model to improve the solution.

ILP Solution

After the termination of the branch-and-price process, the Inequality Solver (IS) generates a solution derived from the feasible inequalities. The result is encapsulated in an object of the `ILPSoultion` class, representing a solution set σ (as presented in the previous chapter). A set of tuples in the form $\langle \mathsf{R}, \mathsf{C}, n, \mathsf{V} \rangle$ is represented as a set of objects of the `Solution` class. Algorithm 5.13 outlines this process.

The `Solution` class serves as a representation for the tuple $\langle \mathsf{R}, \mathsf{C}, n, \mathsf{V} \rangle$, and each object of this class encapsulates attributes corresponding to the components of the tuple. Therefore, the attributes within each `Solution` object are:

- `Set` $<$ `OWLObjectPropertyExpression` $>$ `edges` which represents a set of roles $\mathsf{R} \subseteq N_R$
- `Set` $<$ `OWLClassExpression` $>$ `fillers` which represents a set of concepts $\mathsf{C} \subseteq N$
- `int cardinality` which represents cardinality n
- `Set` $<$ `Integer` $>$ `nodeSet` which represents a set of nodes $\mathsf{V} \subseteq V$ by using their ids.

Chapter 6

Performance Evaluation

This chapter endeavours to assess the real-world application of the algebraic calculus introduced in Chapter 4. The practical performance of Cicada, the prototype reasoner presented in Chapter 5, is evaluated. Cicada implements the hybrid algebraic tableau calculus outlined in this thesis. We compared Cicada with major OWL reasoners such as FaCT++ (1.6.5), HermiT (1.3.8), JFact (1.2.3), and Konclude (0.6.2). Section 6.1 provides an overview of the evaluation methodology, detailing the benchmarking process. In Section 6.2, the test cases employed for evaluation are outlined, accompanied by the presentation of evaluation results. Section 6.3 offers a general analysis of Cicada's performance.

6.1 Evaluation Methodology

The algebraic reasoning algorithm was proposed to address the inefficiencies observed in dealing with highly expressive DL constructs, notably QCRs, nominals, and inverse roles. Therefore, Cicada has been designed and implemented to show its proficiency in handling nominals, inverse roles, and QCRs more effectively than existing reasoners lacking algebraic reasoning. To demonstrate its enhanced capabilities, Cicada undergoes an evaluation through Tbox consistency tests, utilizing ontologies containing these expressive DL constructs.

The effectiveness of the algebraic method has previously been highlighted in handling QCRs, as demonstrated in [37, 26]. Additionally, its applicability extends to addressing both QCRs and nominals, as presented in [23, 22]. It is noteworthy that these previous works did not incorporate the column generation technique.

Therefore, as part of comprehensive testing, Cicada is assessed against test cases involving the use of QCRs and nominals, without necessarily incorporating inverse roles. This approach offers a thorough evaluation of Cicada's expertise in handling a diverse range of expressive DL constructs.

6.1.1 Benchmarking

In addition to ILP and branch-and-price, Cicada only implements a few standard optimization techniques such as lazy unfolding [4, 38], concept absorption [43], role absorption [73], nominal absorption [67], dependency directed backtracking [43], and a ToDo list architecture [74], strategically controlling the application of expansion rules.

However, it is worth noting that, unlike many state-of-the-art reasoners that are equipped with a wide range of optimization techniques to ensure reasonable runtime behaviour across real-world problems, Cicada's optimization techniques are more focused. This specialization could potentially result in suboptimal performance, particularly for $\mathcal{SHOIQ}$ ontologies that require additional optimization techniques.

Furthermore, existing reasoning approaches often face challenges in effectively handling QCRs, particularly when these restrictions involve high numerical values. Consequently, many real-world ontologies are structured in a way that either excludes QCRs completely or uses very small numerical values in QCRs. Therefore, such ontologies may not be suitable as benchmarks for evaluating Cicada's performance. To address this potential limitation, we have used two distinct categories of benchmarks:

1. **A set of synthetic test cases:** For a comprehensive empirical evaluation of Cicada, we constructed a set of synthetic test cases. Among these test cases, some were adapted from existing sources, including those documented in [26, 21]. This diverse set of synthetic scenarios allows us to rigorously assess Cicada's performance across a spectrum of reasoning

challenges.

2. **A set of ontologies from OWL Reasoner Evaluation (ORE 2014) Dataset:** To evaluate Cicada's performance with large-sized ontologies, we derived a set of ontologies from ORE 2014 [7].

6.1.2 Comparative Analysis with Prominent OWL Reasoners

In our investigation, Cicada underwent a comprehensive comparison with leading state-of-the-art OWL reasoners, including FaCT++ (1.6.5) [74], HermiT (1.3.8) [66], JFact (1.2.3), and Konclude (0.6.2) [72]. These reasoners employ distinct reasoning algorithms. Specifically, FaCT++ employs tableau-based DL reasoning, HermiT utilizes hyper-tableau reasoning, and Konclude, while primarily based on tableau calculus, incorporates consequence-based reasoning as well.

In addition to incorporating various state-of-the-art optimization techniques, each reasoner is equipped with numerous optimizations, some tailored to address specific complexities. For instance, FaCT++ employs a ToDo list architecture to control the application of expansion rules. HermiT, on the other hand, implements core blocking to handle ontologies with large cyclic TBoxes effectively. Konclude utilizes absorption-based handling of nominal schemas, pool-based merging, and a known/possible set classification and realization approach.

Therefore, it is crucial to highlight that the performance of a particular system may vary across specific test cases due to various factors that make reasoning service more challenging for the employed reasoning algorithm. Consequently, attributing a degradation or improvement in reasoning performance directly to the reasoning algorithm adopted can be a difficult task.

6.1.3 Evaluation Platform

The benchmark set utilized for evaluation comprises .owl files representing OWL ontologies in the OWL functional format. The evaluation tests were conducted on a robust HP DL580 Scientific Linux SMP server equipped with four 15-core processors (Gen8 Intel Xeon E7-4890v2 2.8 GHz) and a substantial total RAM of 1TB. Each processor boasts 256GB of shared RAM, and the 15 cores are enhanced with hyper-threading support.

6.2 Test Cases

This section outlines the employed test cases and presents the run-times required for Cicada to determine TBox consistency, drawing comparisons with the performance of other reasoners. We focus our evaluation on concept expressions only containing QCRs or nominals.

We have organized three primary sets of synthetic test cases, with an incremental enhancement of DL expressivity in each set. The first set exclusively involves QCRs (DL $\mathcal{SHQ}$), the second set introduces nominals (DL $\mathcal{SHOQ}$), and the third set incorporates inverse roles (DL $\mathcal{SHOIQ}$). Each of these sets is further categorized into two subsets: consistent and inconsistent. While we adapted the synthetic test cases from [26, 21], for the sake of completeness and to ensure this thesis is self-contained, we reiterate the descriptions of those test cases as we report on their performance. Each test case comprises a TBox consistency assessment featuring TBox $\mathcal{T}$, which includes the description of a concept C to evaluate its satisfiability. Additionally, a TBox axiom $\top \sqsubseteq \neg\{a\} \sqcup \mathsf{C}$ is incorporated, where a is a freshly introduced nominal. Furthermore, we derived a set of ontologies from ORE 2014.

In the subsequent sections, we describe the test cases formulated to evaluate the algebraic reasoning approach. These test cases encompass a comprehensive examination of various parameters, including:

1. **The Size of Numbers:** Exploring the impact of varying numerical values within qualified cardinality restrictions.

2. **Number of Nominals:** Assessing the effect of the number of nominals incorporated.

3. **Impact of Inverse Roles:** Examining the consequences of inverse roles in the presence of qualified cardinality restrictions and nominals.

Table 6.1: Evaluation results while the value of i is increased linearly (TO=timeout, Cic=Cicada, FaC=FaCT++, Her=HermiT, JFa=JFact, Kon=Konclude)

i	Test Ontologies - Consistent					Test Ontologies - InConsistent				
	Cic	**FaC**	**Her**	**JFa**	**Kon**	**Cic**	**FaC**	**Her**	**JFac**	**Kon**
1	10.65	1.42	3.15	3.11	0.03	9.53	2.69	4.08	2.27	0.03
2	9.67	1.48	2.71	2.66	0.02	11.89	1.38	3.19	2.19	0.04
3	9.32	1.77	3.99	2.75	0.03	11.35	2.09	4.72	3.55	0.1
4	10.74	1.66	3.56	1.99	0.05	9.61	1.52	5.96	7.93	1.1
5	10.36	2.22	12.9	3.59	0.04	10.19	6.99	171.41	23.03	16.16
6	11.73	7.26	TO	2.7	0.07	12.52	96.07	TO	216.31	230.93
7	9.96	34.29	TO	4.3	0.08	10.47	TO	TO	TO	TO
8	11.1	364.93	TO	5.94	0.1	12.02	TO	TO	TO	TO
9	11.47	TO	TO	7.45	0.12	11.38	TO	TO	TO	TO
10	10.38	TO	TO	17.18	0.17	10.49	TO	TO	TO	TO

4. **Satisfiability vs. Unsatisfiability:** Distinguishing the performance concerning the satisfiability and unsatisfiability of the given concept expression.

These parameters collectively contribute to a thorough understanding of the algebraic reasoning approach's efficacy in handling numerical restrictions across various scenarios. We show runtimes of our results in seconds and set the timeout limit to 1000 seconds.

6.2.1 Test cases for $\mathcal{SHQ}$

This test examines the impact of varying numerical values within Qualified Cardinality Restrictions (QCRs). A concept C is defined as:

$$\mathsf{C} \quad \sqsubseteq \; \geq 2iRS.(A \sqcup B) \sqcap \leq iS.A \sqcap \leq iR.B \sqcap \leq (i-1)T.\neg A \sqcup \leq jT.\neg B$$

where $\{R, S, RS, T\} \subseteq N_R, \{R \sqsubseteq T, S \sqsubseteq T, RS \sqsubseteq R, RS \sqsubseteq S\} \subseteq \mathcal{R}$.

Given $\top \sqsubseteq \neg\{a\} \sqcup \mathsf{C}_{\mathsf{SAT}}$, which indicates that a is a member of C and has a label $\mathcal{L}(a) = \{\geq 2iRS.(A \sqcup B), \leq iS.A, \leq iR.B, (\leq (i-1)T.\neg A \sqcup \leq jT.\neg B)\}$. Considering that C is satisfiable if a satisfies the following numerical restrictions:

1. $\geq 2iRS.(A \sqcup B)$: a must have at least $2i$ RS-fillers, each satisfying $(A \sqcup B)$

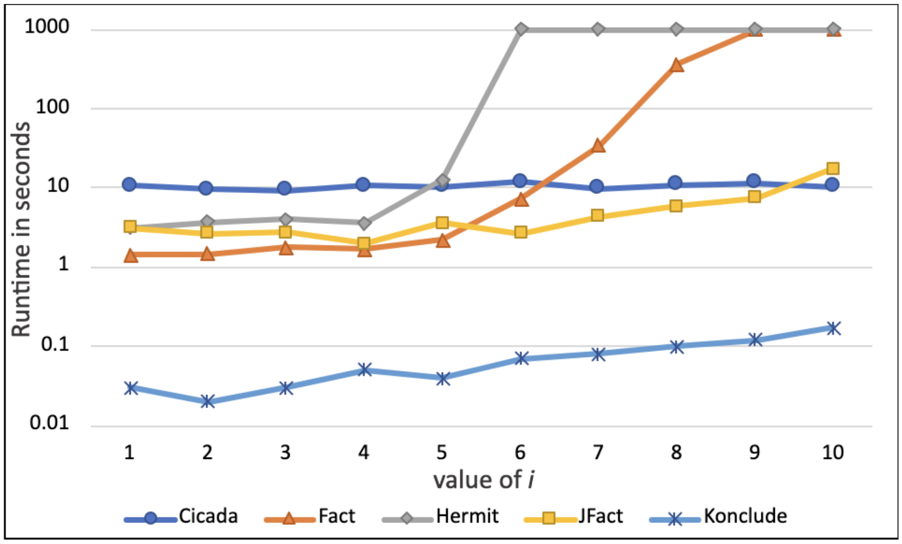


Figure 6.1: Effects of linearly increasing the numbers used in QCRs in $\mathsf{C}_{\mathsf{SAT}}$

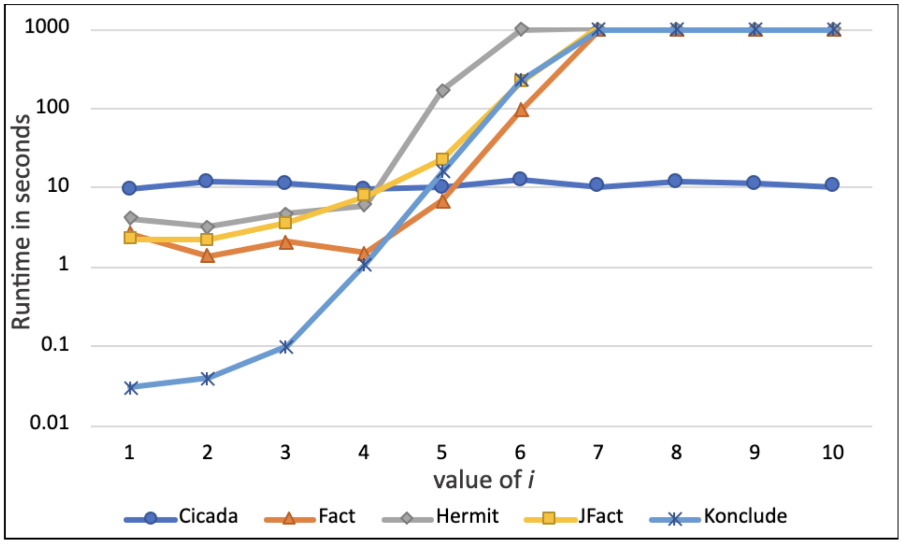


Figure 6.2: Effects of linearly increasing the numbers used in QCRs in $\mathsf{C}_{\mathsf{UnSAT}}$

2. $\leq iS.A$: At most i S-fillers of a can be members of A.

3. $\leq iR.B$: At most i R-fillers of a can be members of B.

4. $(\leq (i-1)T.\neg A \sqcup \leq jT.\neg B)$: There can be at most $(i-1)$ T-fillers of a that are members of $\neg A$, or there can be at most j T-fillers of a that are members of $\neg B$.

The satisfaction of a with (1), (2), and (3) holds for all $i > 0$. However, to satisfy (4), a must also satisfy $\leq jT.\neg B$ because $\leq (i-1)T.\neg A$ cannot be satisfied. Consequently, the satisfiability of C depends on the value of j; when $j \geq i$, C becomes satisfiable; otherwise, it becomes unsatisfiable. Therefore, this set of ontologies is divided into two subsets. In the first set, a concept C is satisfiable where $j = i$, while in the second one, C is unsatisfiable where $j = (i-1)$.

$$\begin{aligned}
\mathsf{C}_{\mathsf{SAT}} &\sqsubseteq \geq 2iRS.(A \sqcup B) \sqcap \leq iS.A \sqcap \leq iR.B \sqcap \leq (i-1)T.\neg A \sqcup \leq iT.\neg B \\
\mathsf{C}_{\mathsf{UnSAT}} &\sqsubseteq \geq 2iRS.(A \sqcup B) \sqcap \leq iS.A \sqcap \leq iR.B \sqcap \leq (i-1)T.\neg A \sqcup \leq (i-1)T.\neg B
\end{aligned}$$

We increment the value of i to observe its impact on reasoning performance. Initially, the numbers are increased linearly with i ranging from 1 to 10. Table 6.1 presents the evaluation results for both satisfiable and unsatisfiable cases. These findings reveal that Cicada's performance remains unaffected by the linear increase in numbers. For satisfiable cases, most reasoners exhibit improved performance with a linear increase in i. In contrast, in unsatisfiable cases runtime for other reasoners escalates significantly even with a linear increase in i and relatively small values.

Figure 6.1 and Figure 6.2 visually illustrate the effects of linearly increasing numbers in satisfiable and unsatisfiable cases, respectively.

In the second test, numbers are increased exponentially using $i = 10^k$, with k ranging from 1 to 6. Table 6.2 provides the evaluation results for both satisfiable and unsatisfiable cases. Once again, these results demonstrate that Cicada's performance remains robust against the increase in numbers. However, as the value of i grows exponentially, most of the reasoners are not able to process these ontologies within the time limit whenever $i \geq 7$.

Table 6.2: Evaluation results while the value of i is increased exponentially (TO=timeout, ERR=wrong result, Cic=Cicada, FaC=FaCT++, Her=HermiT, JFa=JFact, Kon=Konclude)

i	Test Ontologies - Consistent					Test Ontologies - InConsistent				
	Cic	**FaC**	**Her**	**JFa**	**Kon**	**Cic**	**FaC**	**Her**	**JFac**	**Kon**
10^1	12.9	4.02	TO	TO	TO	9.53	TO	TO	TO	TO
10^2	12.53	TO	TO	TO	TO	11.89	TO	TO	TO	TO
10^3	12.18	TO	ERR	TO	TO	9.61	TO	ERR	TO	TO
10^4	11.13	TO	TO	TO	TO	10.19	TO	TO	TO	TO
10^5	10.12	ERR	TO	TO	TO	12.52	ERR	TO	TO	TO
10^6	12.42	TO	TO	TO	TO	11.35	TO	TO	TO	TO

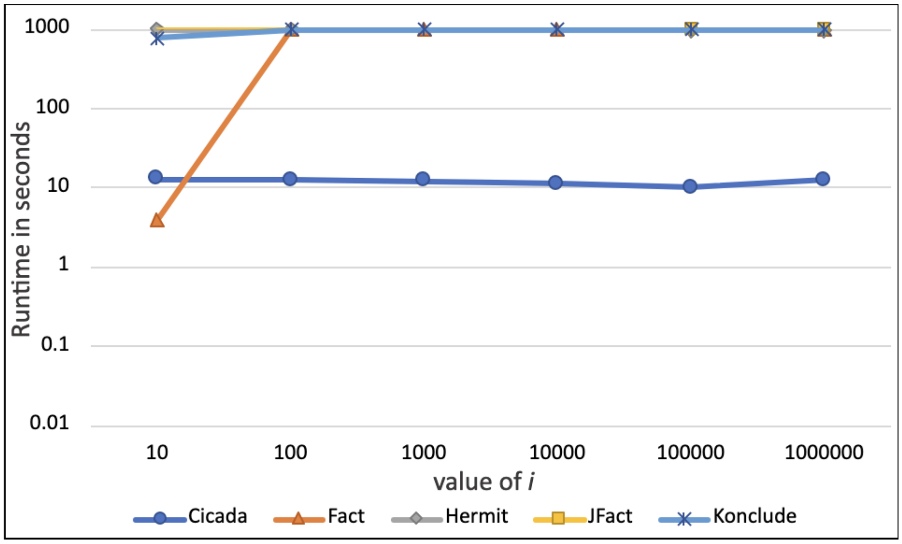


Figure 6.3: Effects of exponentially increasing the numbers used in QCRs in $\mathsf{C}_{\mathsf{SAT}}$

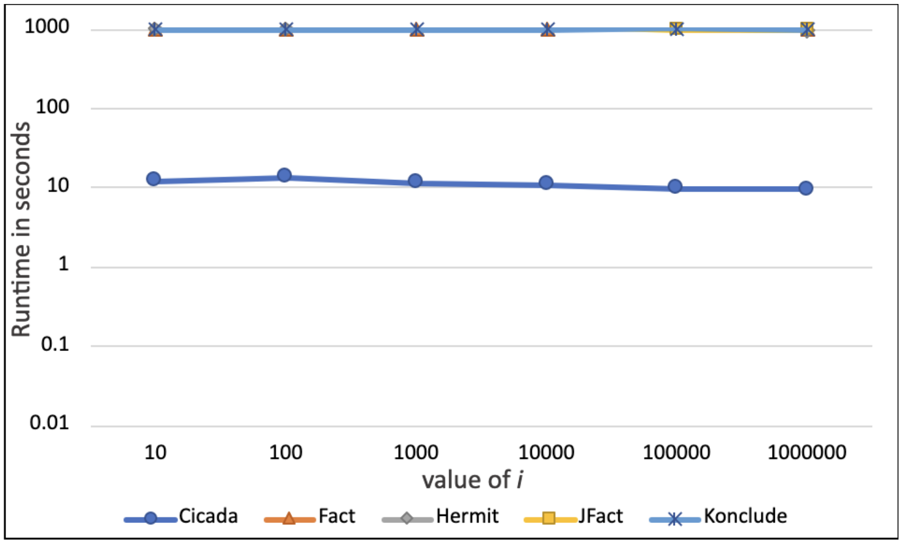


Figure 6.4: Effects of exponentially increasing the numbers used in QCRs in $\mathsf{C}_{\mathsf{UnSAT}}$

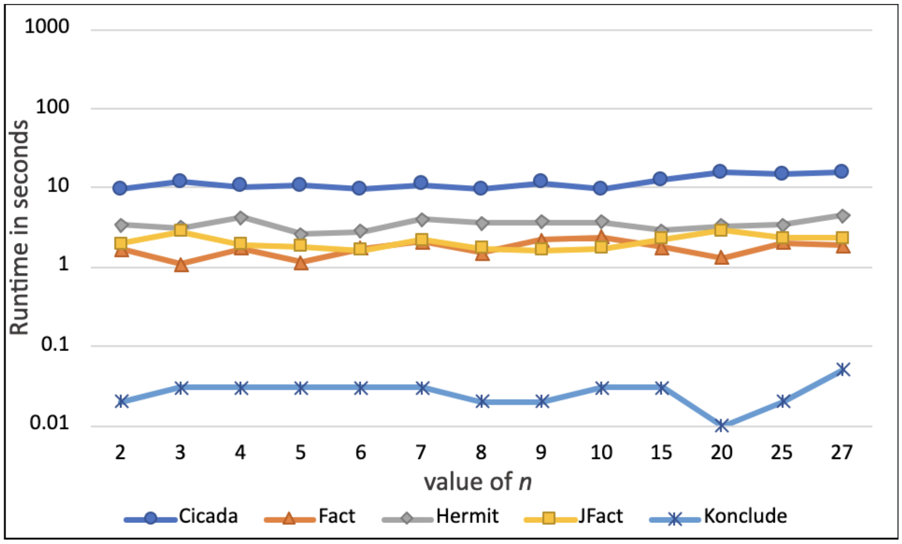


Figure 6.5: Effects of increasing the number of nominal and the numbers used in QCRs in $\mathsf{EU}_{\mathsf{SAT}}$

Figures 6.3 and 6.4 depict the effects of exponentially increasing numbers in satisfiable and unsatisfiable cases, respectively.

The consistent and effective performance of Cicada in solving these test cases underscores the advantage of employing algebraic reasoning for handling QCRs compared to alternative reasoning approaches. It is noteworthy that, due to the utilization of proxy nodes (refer to Definition 31), the same completion graph remains valid for cases where numbers are increased linearly and exponentially. In the former scenario, the proxy nodes represent i elements, while in the latter case, they represent 10^k elements ($1 \leq k \leq 6$). This adaptability highlights the flexibility and stability of Cicada's approach across varying numerical values within QCRs.

6.2.2 Test cases for $\mathcal{SHOQ}$

This test specifically investigates the influence of the number of nominals and varying numerical values within QCRs. The European Union (EU) example, adapted from [21], serves as the basis for creating test cases that incorporate both nominals and QCRs. In this example, the member states are represented as an enumeration of 27 distinct nominals, with each nominal corresponding to a specific member state. This representation can be expressed as

Table 6.3: Evaluation results of consistency test of $\mathsf{EU}_{\mathsf{SAT}}$ and $\mathsf{EU}_{\mathsf{UnSAT}}$ (TO=timeout, Cic=Cicada, FaC=FaCT++, Her=HermiT, JFa=JFact, Kon=Konclude)

n	Test Ontologies - Consistent					Test Ontologies - InConsistent				
	Cic	**FaC**	**Her**	**JFa**	**Kon**	**Cic**	**FaC**	**Her**	**JFac**	**Kon**
2	9.5	1.69	3.37	1.97	0.02	8.76	1.83	3.89	2.7	0.03
3	11.84	1.09	3.13	2.78	0.03	8.91	1.63	3.15	2	0.03
4	10.43	1.75	4.29	1.9	0.03	14.53	2.37	3.94	2.72	0.04
5	10.62	1.14	2.58	1.81	0.03	10.36	2.63	3.92	3.3	0.07
6	9.62	1.72	2.8	1.64	0.03	8.75	1.65	3.69	7.15	0.29
7	10.95	2.11	3.99	2.18	0.03	8.4	1.73	5.4	10.47	1.99
8	9.53	1.51	3.58	1.72	0.02	9.28	2.94	11.39	26.96	17.41
9	11.72	2.21	3.74	1.63	0.02	10.36	7.63	80.64	161.54	176.11
10	9.47	2.37	3.69	1.73	0.03	12.94	76.18	TO	TO	TO
15	12.6	1.77	2.89	2.25	0.03	11	TO	TO	TO	TO
20	15.7	1.3	3.33	2.9	0.01	11.71	TO	TO	TO	TO
25	14.79	2.03	3.44	2.34	0.02	15.02	TO	TO	TO	TO
27	15.75	1.88	4.41	2.32	0.05	17.44	TO	TO	TO	TO

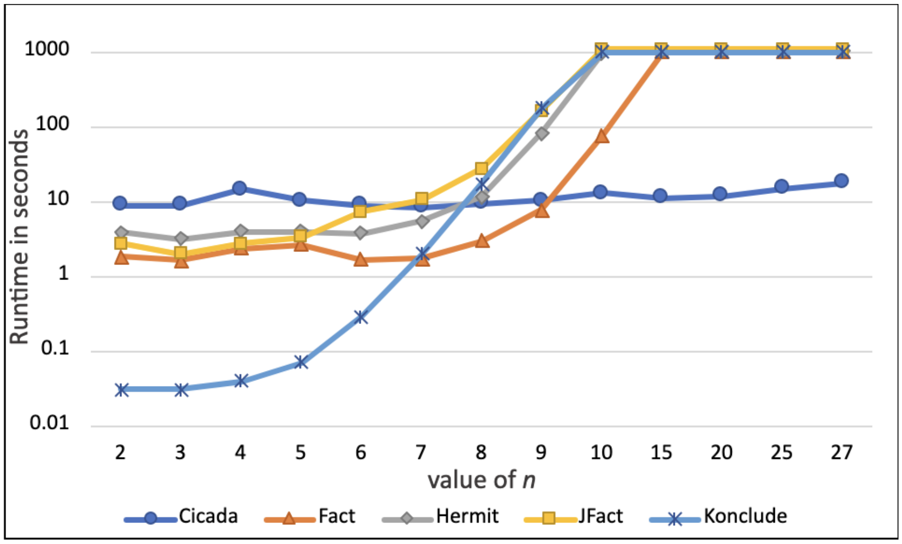


Figure 6.6: Effects of increasing the number of nominal and the numbers used in QCRs in $\mathsf{EU}_{\mathsf{UnSAT}}$

$$\mathsf{EU\_MemberState} \equiv \{\mathsf{Austria}, ..., \mathsf{UK}\}$$

Here, a TBox consistency test is conducted by assessing the satisfiability of a concept EU through the inclusion of a TBox axiom $\top \sqsubseteq \neg\{a\} \sqcup \mathsf{EU}$ within a TBox $\mathcal{T}$. Similar to the previous sets, this test is further categorized into two subsets. In the first subset, the concept EU is satisfiable, whereas, in the second subset, EU is unsatisfiable. A concept EU is defined as:

$$\begin{aligned} \mathsf{EU}_{\mathsf{SAT}} &\sqsubseteq \geq n\, \mathsf{memberOf.EU\_MemberState} \\ \mathsf{EU}_{\mathsf{UnSAT}} &\sqsubseteq \geq (n+1)\mathsf{memberOf.EU\_MemberState} \end{aligned}$$

Furthermore, the number of nominals used to define a concept $\mathsf{EU\_MemberState}$ also varies depending on the value of n in a concept EU. Therefore, for testing purposes, we define $\mathsf{EU\_MemberState}$ as:

$$\mathsf{EU\_MemberState} \equiv \{o_1, ..., o_n\}$$

In order to satisfy the at-least restriction $\geq n\, \mathsf{memberOf.EU\_MemberState}$, a standard tableau reasoner generates n distinct $memberOf$-fillers as instances of $\mathsf{EU\_MemberState}$ and then attempts to merge them with n nominals enumerated in the definition of EU_MemberState. Therefore, the complexity of these test cases arises from this nondeterministic merging process.

Table 6.3 provides a comprehensive overview of the evaluation results for both consistent and inconsistent ontologies. In satisfiable cases, most reasoners exhibit efficient performance. However, in the case of $\mathsf{EU}_{\mathsf{UnSAT}}$, these reasoners generate $n+1$ distinct $memberOf$-filler and attempt to merge them with n nominals. Consequently, their performance deteriorates as the value of n increases in unsatisfiable cases. Utilizing algebraic reasoning for handling nominals and QCRs, Cicada swiftly recognizes that $n+1$ elements cannot be merged with n nominals, ensuring its performance remains unaffected by an increase in the value of n. Figures 6.5 and 6.6 visually illustrate how the reasoning performance is affected by increasing the number of nominal and the numbers used in QCRs.

6.2.3 Test cases for $\mathcal{SHOIQ}$

As discussed earlier, the interaction of nominals with inverse roles and QCRs adds complexity to reasoning services. In this section, we investigate Cicada's performance on $\mathcal{SHOIQ}$ ontologies where nominals, QCRs and inverse roles interact. We employ three sets of benchmarks for this evaluation:

1. The first set consists of small synthetic test ontologies utilizing a variable n to represent the number of nominals. This allows us to assess the impact of increased nominals and the numbers used in QCRs in the presence of inverse roles.

2. The second benchmark includes a set of 221 consistent and a set of 181 inconsistent $\mathcal{SHOIQ}$ ontologies.

3. The third benchmark is derived from ORE 2014 and encompasses 2882 $\mathcal{SHOIQ}$ ontologies. For testing purposes, Aboxes have been removed because Cicada implements no Abox optimization techniques.

In the first benchmark, we explore the effect of increasing the number of nominals and the numbers used in QCRs in the presence of inverse roles. For testing purposes, concepts C, A and B are defined as

$$C \sqsubseteq \exists R^{-}.A$$

$$A \sqsubseteq \leq n\, R.\top \sqcap \geq n\, R.B \sqcap \geq m_1\, R.D \sqcap \geq m_2\, R.E$$

$$B \sqsubseteq \{o_1, ..., o_n\}$$

$$C \sqcap E \sqsubseteq \bot, C \sqcap D \sqsubseteq \bot, E \sqcap D \sqsubseteq \bot$$

Nominals $o_1, ..., o_n$ are declared as pairwise disjoint. The first set comprises consistent ontologies where $m_1 + m_2 = n - 1$. The second set consists of inconsistent ontologies where $m_1 + m_2 = n$. Table 6.4 presents the evaluation results for both consistent and inconsistent cases. Similar to previous tests, these results demonstrate improved performance for consistent ontologies. However, only Cicada can process all inconsistent ontologies within the time limit.

Table 6.4: Evaluation results of satisfiability test of $\mathsf{EU}_{\mathsf{SAT}}$ and $\mathsf{EU}_{\mathsf{UnSAT}}$
(TO=timeout, Cic=Cicada, FaC=FaCT++, Her=HermiT, JFa=JFact, Kon=Konclude)

n	**Test Ontologies - Consistent**					**Test Ontologies - InConsistent**				
	Cic	**FaC**	**Her**	**JFa**	**Kon**	**Cic**	**FaC**	**Her**	**JFac**	**Kon**
5	12.44	1.95	3.02	0.74	0.04	14.3	1.64	24.59	24.59	0.04
7	14.26	2.36	12.68	0.77	0.03	14.49	12.78	TO	TO	0.23
10	11.88	2.34	TO	0.92	0.05	11.71	TO	TO	TO	40.63
20	19.42	2.35	TO	0.87	0.06	16.2	TO	TO	TO	TO
40	34.49	1.81	TO	0.93	0.2	19.7	2.46	TO	TO	TO

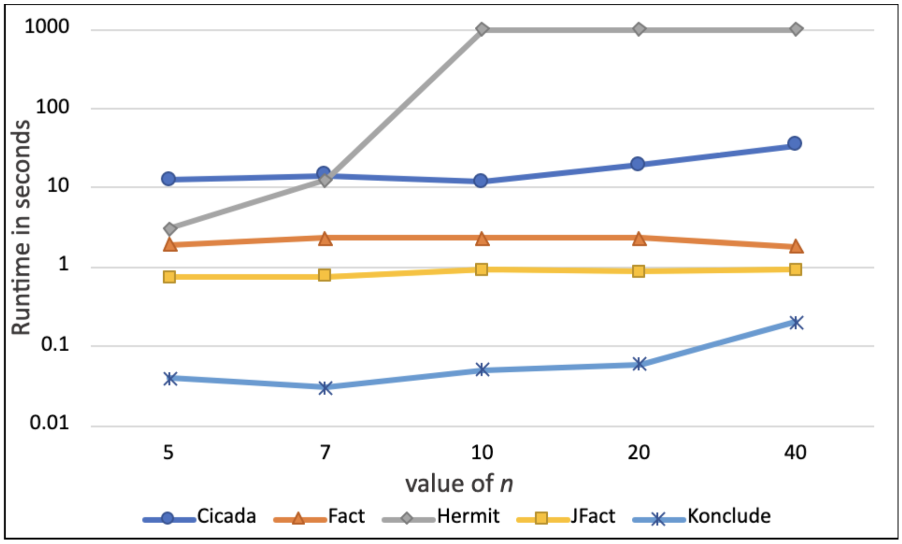


Figure 6.7: Effects of increasing the number of nominals and the numbers used in QCRs in the presence of inverse role in consistent ontologies

Figures 6.7 and 6.8 visually depict the performance of the reasoners in both consistent and inconsistent cases, considering the presence of nominals, cardinality restrictions, and inverse roles.

The second benchmark contains a set of 221 consistent and a set of 181 inconsistent ontologies. These small ontologies exhibit entailments depending on combinations of cardinality restrictions, nominals, and inverse roles. A timeout of 1000 seconds of CPU time was used. The runtimes for test ontologies are ordered for each reasoner in increasing runtime values. The resulting graphs demonstrate the superior performance of Cicada (see Figures 6.9 and 6.10).

The third benchmark is derived from the OWL Reasoner Evaluation (ORE 2014 [7]) dataset. This dataset encompasses multiple extensive collections of real-life OWL ontologies acquired from

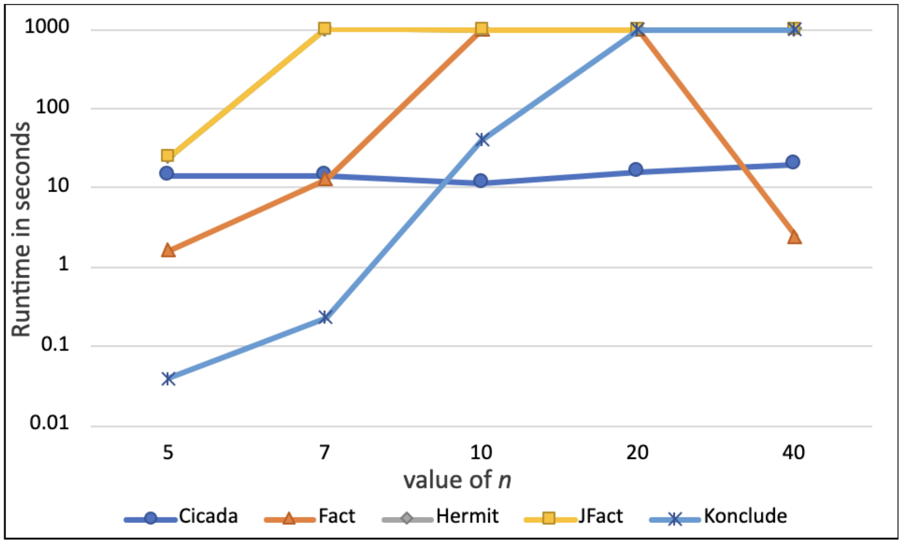


Figure 6.8: Effects of increasing the number of nominals and the numbers used in QCRs in the presence of inverse role in consistent ontologies

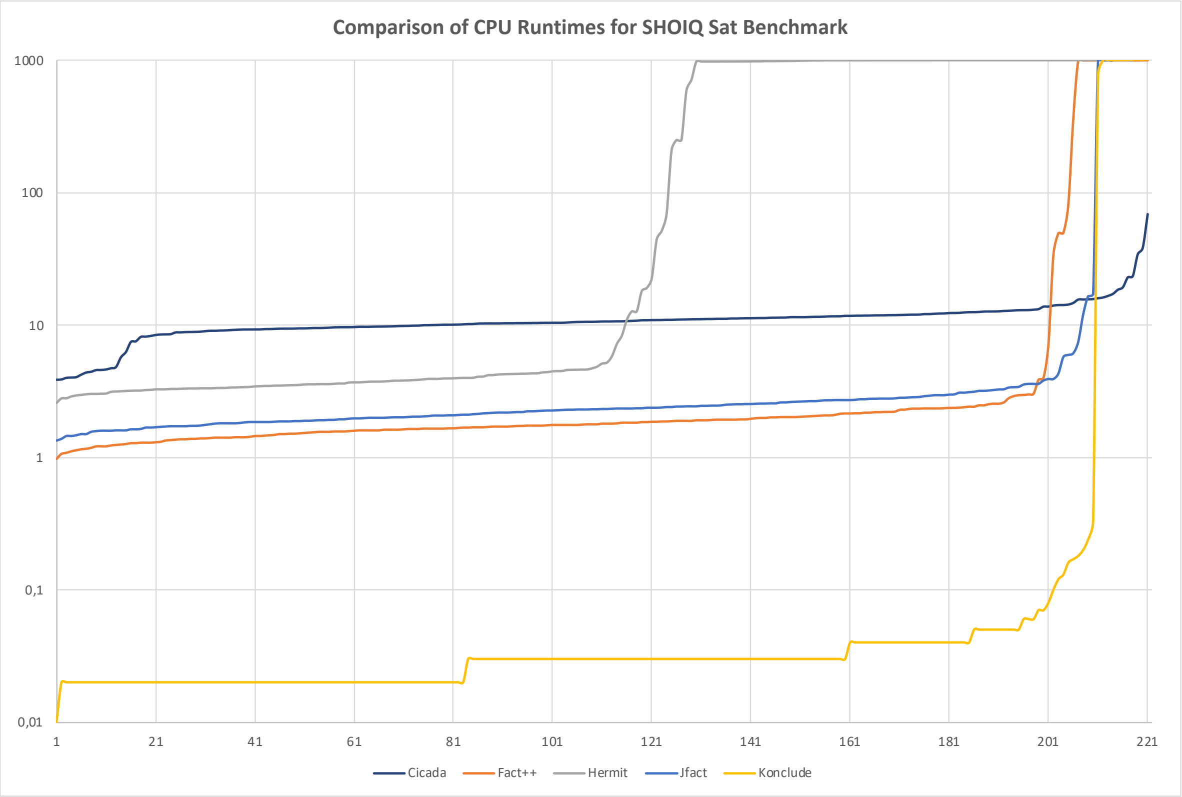


Figure 6.9: Results of 221 Consistent Ontologies

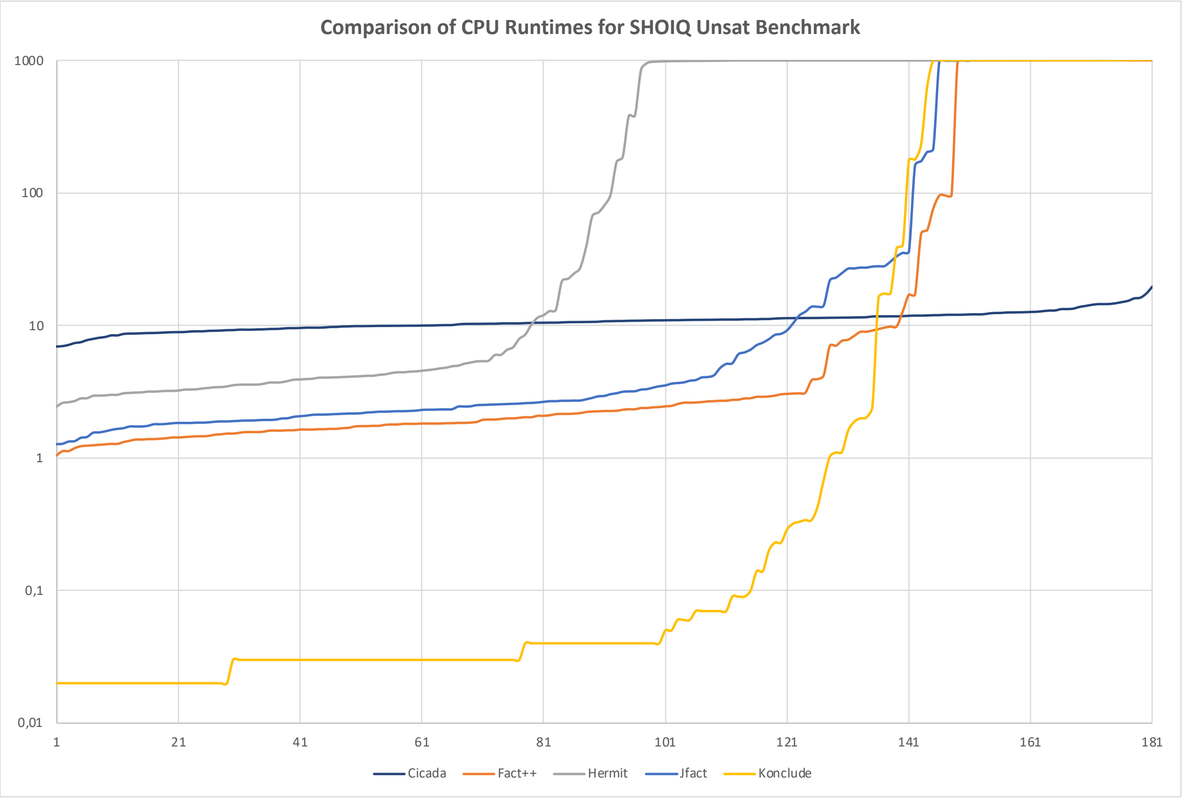


Figure 6.10: Results of 181 Inconsistent Ontologies

the web. Additionally, it includes user-submitted ontologies that were identified as particularly challenging for reasoners. It contains 2882 $\mathcal{SHOIQ}$ ontologies. The set includes:

- The MOWLCorp (Manchester OWL Corpus)[1]
- The Oxford Ontology Library[2]
- The NCBO BioPortal[3] Snapshot (June 2014)
- User-submitted ontologies, including:
  - The Data Mining OPtimization Ontology (DMOP) [47], a complex $\mathcal{SROIQ}$ ontology with around 3,000 logical axioms

[1] http://mowlrepo.cs.manchester.ac.uk/datasets/mowlcorp/
[2] http://www.cs.ox.ac.uk/isg/ontologies/
[3] https://bioportal.bioontology.org/

  - Genomic CDS [63], an $\mathcal{ALCQ}$ ontology containing approximately 4,000 logical axioms, involving a high number of qualified number restrictions of the type ’exactly 2’.
  - Bio KB 101 [12], a set of OWL approximations of the first-order logic representation of a biology textbook, consisting of 432 $\mathcal{SHOIQ}$ ontologies.
  - FMA-FNL, a variant of the FMA (Foundational Model of Anatomy) ontology [62]. It is a large and highly cyclic $\mathcal{ALCOI}\,(\mathcal{D})$ ontology with over 120,000 logical axioms.
  - GALEN-FNL, a highly cyclic $\mathcal{ALCHOI}\,(\mathcal{D})$ variant of the well-known Galen ontology [61], containing around 37,000 logical axioms and 951 object properties.
  - GALEN-Heart: a highly cyclic $\mathcal{ALCHOI}\,(\mathcal{D})$ ontology containing a module extracted from the Galen ontology with over 10,000 logical axioms.
  - Functional Therapeutic Chemical Classification System (FTC)[4], a large ontology with nearly 300,000 logical axioms

- The National Cancer Institute (NCI) Thesaurus (NCIt) (May 2013 version) [33]
- The Systematized Nomenclature of Medicine (SNOMED) Clinical Terms (SNOMED CT) (January 2011 version) [65]

A timeout of 1000 seconds of CPU time was set. The runtimes for test ontologies are organized for each reasoner in increasing runtime values, as illustrated in Figure 6.11, showcasing the evaluation results of the ORE 2014 dataset. Notably, Cicada employs a limited set of optimization techniques that do not detect cases where entailments cannot be derived. Comparatively, the other four reasoners use more extensive optimization techniques. Consequently, Cicada’s performance falls within the mid-range for this benchmark.

Despite this, Cicada outperformed HermiT in over 2700 ontologies and exhibited superior performance to JFact in 339 ontologies. It’s worth noting that neither HermiT nor JFact could process

[4] https://www.ebi.ac.uk/chembl/ftc/

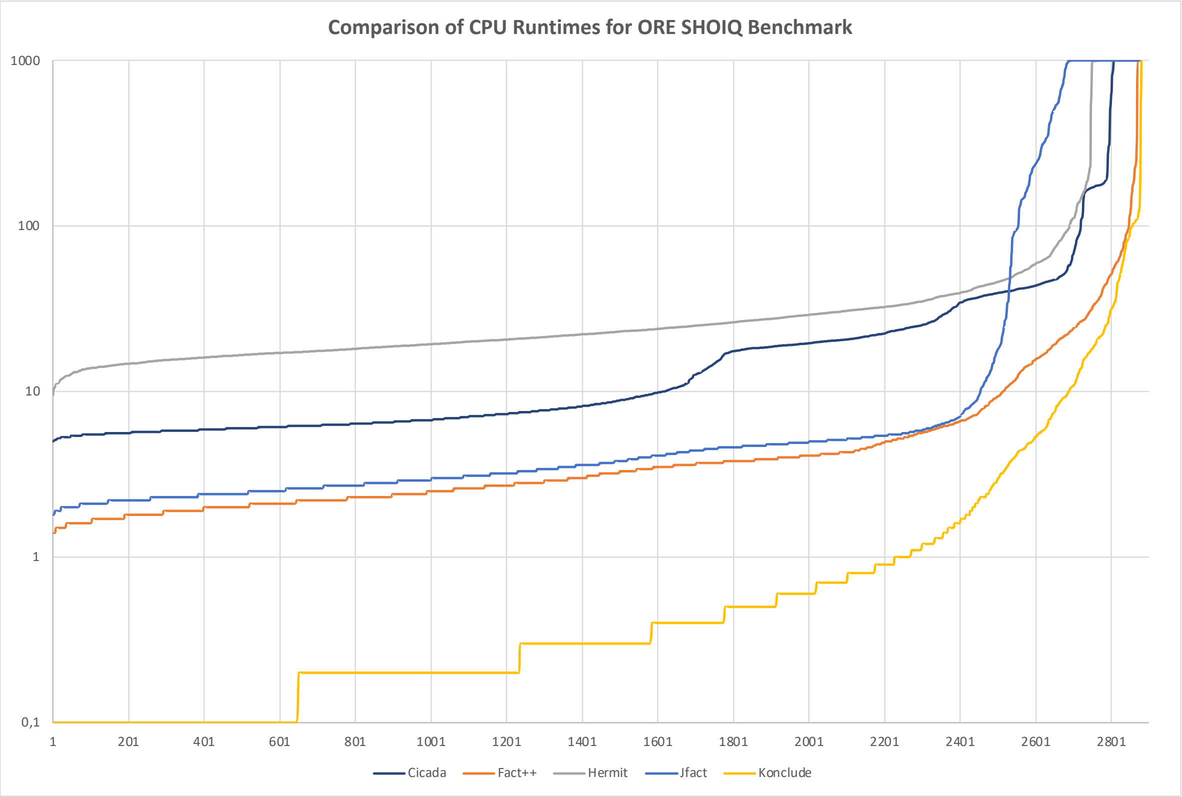


Figure 6.11: Results of ORE 2014 $\mathcal{SHOIQ}$ consistency benchmark (2882 ontologies)

most of the BioKB ontologies within the specified time limit. In contrast, Cicada successfully processed all of them in less than 4 minutes.

6.3 Discussion

Algebraic reasoning allows Cicada to handle nominals, inverse roles, and QCRs more effectively than existing reasoners. The evaluation of Cicada's performance highlights its effectiveness compared to existing reasoners. Traditional reasoning approaches often encounter challenges when dealing with QCRs, especially those involving high numerical values. Real-world ontologies often circumvent these challenges by either excluding QCRs or using small numerical values. Consequently, such ontologies might not serve as ideal benchmarks for evaluating Cicada's capabilities. To overcome this limitation, we used a set of synthetic test cases along with the ORE 2014 dataset.

The evaluation involved 32 ontologies focused solely on QCRs, examining the impact of increasing numerical values in these restrictions. In 26 additional ontologies, nominals were introduced alongside QCRs to assess their combined impact on the reasoning process. A total of 412 small $\mathcal{SHOIQ}$ ontologies were also tested. The results demonstrated Cicada's robust performance against increasing numerical values. However, existing reasoners faced challenges with high numerical values, struggling to process these ontologies within the specified time limit. Konclude, leveraging new optimization techniques, performed better overall, yet it also encountered difficulties with ontologies featuring large numerical values.

Furthermore, we derived 2882 ontologies from the ORE 2014 dataset. It's noteworthy that Cicada's optimization techniques are more focused, primarily employing column generation and branch-and-price techniques. While Cicada's performance is suboptimal, especially for ontologies requiring additional optimization techniques, it still outperformed other reasoners in some cases.

Chapter 7

Conclusions and Future Work

Description Logics (DLs) have gained significant traction in Knowledge Representation and modelling, prompting numerous research endeavours to enhance their expressivity and reasoning efficiency. It has become imperative for a DL not only to accommodate expressivity for modelling all elements within an application's domain but also to facilitate efficient reasoning when employing the full extent of expressivity. As previously mentioned, conventional tableau-based reasoning procedures lack arithmetic awareness and exhibit a blind approach. As a consequence, DL reasoners employing these procedures face inefficiencies when handling the expressivity associated with nominals and qualified cardinality restrictions (QCRs) during inference processes.

Nominals, capturing the essence of uniqueness and identity, play a crucial role in representing concepts with a single instance in various real-world domains such as "Sun," "Blue," or "Canada." On the other hand, QCRs empower the language with the capability to articulate numerical constraints concerning relationships. Despite the significance of nominals and QCRs, the limited ability of existing DL reasoners to efficiently handle these constructs has hindered the development of real-world ontologies extensively utilizing them. Additionally, inverse roles, while enhancing expressiveness by representing bidirectional connections, introduce higher computational complexity in reasoning tasks.

The primary goal of this thesis, as outlined in Section 1.3, was to devise an efficient reasoning algorithm capable of handling the expressive features of DLs, including QCRs, nominals, and inverse roles. The subsequent sections of this chapter delineate the achieved objectives and highlight the key contributions made by this thesis.

7.1 Research Methodology

The research methodology adopted involves the following key steps:

1. **Problem Identification:** The initial step involved identifying the challenges and limitations in existing reasoning algorithms for Description Logics, especially when dealing with expressive features such as QCRs, nominals, and inverse roles.

2. **Objective Formulation:** Once the challenges were identified, the research objectives were formulated. The primary goal was to design a reasoning algorithm that efficiently handles the expressivity of DLs incorporating QCRs, nominals, and inverse roles.

3. **Algorithm Design:** A novel reasoning algorithm was designed, building on the hybrid approach that combined tableau-based reasoning with algebraic reasoning. Theoretical foundations, including soundness, completeness, and termination proofs, were established for the proposed reasoning algorithm. This ensured the reliability and correctness of the developed approach.

4. **Implementation of Prototype System:** A prototype system, named Cicada, was implemented to validate the practical applicability of the proposed reasoning algorithm.

5. **Empirical Evaluation:** An empirical evaluation was conducted to assess the performance of Cicada against existing state-of-the-art reasoners. Datasets and metrics were defined to measure and compare the efficiency and effectiveness of the developed reasoning system.

6. **Documentation and Analysis:** The entire research process was documented, and the results obtained from the theoretical framework, algorithm design, and empirical evaluation were

analyzed to draw meaningful conclusions.

7.2 Research Contributions

The main contributions of the thesis align with the previously outlined objectives (see Section 1.3) and can be categorized as follows:

Algorithmic and Theoretical Contribution:

1. **Calculus Design for $\mathcal{SHOI}$ and $\mathcal{SHOIQ}$:**

   (a) The design of a decidable calculus for the DL $\mathcal{SHOI}$ was achieved through a hybrid algebraic approach, as documented in [24, 25]. This included defining a tableau for $\mathcal{SHOI}$, creating a hybrid reasoning procedure that utilized algebraic reasoning for satisfiability decisions involving nominals and inverse roles, and establishing proofs for soundness, completeness, and termination.

   (b) Extending the calculus to handle the more expressive DL $\mathcal{SHOIQ}$ was another main contribution. This involved adapting the tableau reasoning algorithm and integrating an algebraic component while ensuring the preservation of soundness, completeness, and termination. The extension incorporated the handling of QCRs, nominals, and inverse roles. The proofs for soundness, completeness, and termination were extended to cover the augmented expressive DL. The calculus was presented in Chapter 4.

2. **Algorithmic Enhancements:** The reasoning algorithm has undergone substantial enhancements, constituting a pivotal aspect of our contributions. These enhancements included:

   (a) Encoding numerical restrictions imposed by nominals and QCRs into inequalities.

   (b) Utilizing Integer Linear Programming (ILP) to decide the feasibility of numerical restrictions and enhancing computational capabilities of the reasoning approach. The branch-and-price method has been used which is a combination of column generation and branch-and-bound technique. The feasibility test for the linear inequalities can be computed in polynomial time, as shown in [51].

(c) Embedding additional knowledge about axioms such as universal restrictions, role hierarchy, subsumption, and disjointness to provide a more informed mapping of QCR satisfiability to feasibility.

Practical Contribution

1. **Prototype Reasoner Design and Implementation:** A running prototype reasoner has been designed and implemented. The system served as a proof of concept and demonstrated the feasibility of the hybrid approach in real-world scenarios. The architecture of this prototype reasoner was detailed in Chapter 5.

2. **Empirical Evaluation:** The thesis contributed to the field by conducting an empirical evaluation of the performance of the proposed reasoning approach. The evaluation, presented in Chapter 6, provides insights into the practical efficiency and effectiveness of the hybrid approach. The evaluation, encompassing synthetic test cases and the ORE 2014 dataset, showed that algebraic reasoning enhanced Cicada's performance in handling nominals, inverse roles, and QCRs. Moreover, the evaluation revealed Cicada's robust performance against increasing numerical values. To assess the efficiency of Cicada, we compared its total runtime performance with other reasoners. In the $\mathcal{SHOIQ}$ inconsistent dataset, Cicada demonstrated a notable speedup factor, surpassing Fact++ by 17.29, Konclude by 19.57, JFact by 20.50, and HermiT by an impressive 44.56. Additionally, in the $\mathcal{SHOIQ}$ consistent dataset, Cicada exhibited a substantial speedup factor, outperforming Konclude by 4.35, JFact by 4.89, Fact++ by 6.41, and HermiT by 38.08. This comparison revealed that Cicada outperformed other reasoner by a significant margin.

These contributions collectively advanced the understanding and application of reasoning techniques for expressive DLs involving nominals, QCRs, and inverse roles, emphasizing both theoretical foundations and practical implementations.

7.3 Future Work

Future work in this research domain could explore several avenues to further enhance and extend the contributions made by the current study. Here are some potential directions for future research:

1. **Optimization Techniques:** The proposed approach primarily emphasizes managing large cardinality restrictions and nominals in the presence of inverse roles; however, Cicada currently lacks certain optimization techniques. To further refine the reasoning process, reduce computational complexity, and enhance overall efficiency, there is a need to explore and develop additional optimization techniques. It's worth noting that Cicada does not incorporate any Abox optimization techniques, which could be considered for inclusion in future work.

2. **Support for Additional DL Features:** The proposed calculus effectively handles the most expressive DL constructs, yet there are certain DL features that it does not currently support. For instance, complex role inclusion axioms are not included. Additionally, reflexive, irreflexive asymmetric, and disjoint roles are not addressed within the scope of this calculus. Future enhancements to the calculus could incorporate support for these additional features.

3. **Open Source Collaboration:** Cicada can be released as an open-source project to encourage collaboration and contributions from the broader research and developer communities. This can lead to continuous improvement and adoption by a wider user base. Furthermore, other reasoners can benefit from algebraic reasoning in handling large numbers.

4. **Developing an Independent Algebraic Module:** Consider creating a standalone algebraic module, allowing other reasoners to adopt and leverage algebraic reasoning for efficient

handling of large numbers. This modular approach enhances the flexibility and applicability of algebraic reasoning across various reasoning systems.

By exploring these avenues, future research can contribute to the ongoing development of efficient and effective reasoning algorithms for Description Logics, addressing both theoretical and practical aspects of knowledge representation and reasoning.

Bibliography

[1] Franz Baader. *The description logic handbook: Theory, implementation and applications*. Cambridge university press, 2003.

[2] Franz Baader, Sebastian Brandt, and Carsten Lutz. Pushing the EL envelope. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI'05)*, volume 5, pages 364–369, 2005.

[3] Franz Baader, Sebastian Brandt, and Carsten Lutz. Pushing the EL envelope further. In *Proceedings of the 5th OWL: Experiences and Directions (OWLED'08)*, October 2008.

[4] Franz Baader, Bernhard Hollunder, Bernhard Nebel, Hans-Jürgen Profitlich, and Enrico Franconi. Am empirical analysis of optimization techniques for terminological representation systems. *Applied Intelligence*, 4(2):109–132, 1994.

[5] Franz Baader, Ian Horrocks, and Ulrike Sattler. Description logics. *Foundations of Artificial Intelligence*, 3:135–179, 2008.

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

[7] Samantha Bail, Birte Glimm, Ernesto Jiménez-Ruiz, Nicolas Matentzoglu, Bijan Parsia, and Andreas Steigmiller, editors. *Informal Proceedings of the 3rd International Workshop on OWL Reasoner Evaluation (ORE 2014) co-located with the Vienna Summer of Logic (VSL*

2014), Vienna, Austria, July 13, 2014, volume 1207 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2014.

[8] Cynthia Barnhart, Ellis L Johnson, George L Nemhauser, Martin WP Savelsbergh, and Pamela H Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations research*, 46(3):316–329, 1998.

[9] Andrew Bate, Boris Motik, Bernardo Cuenca Grau, David Tena Cucala, František Simančík, and Ian Horrocks. Consequence-based reasoning for description logics with disjunctions and number restrictions. *Journal of Artificial Intelligence Research*, 63:625–690, 2018.

[10] Andrew Bate, Boris Motik, Bernardo Cuenca Grau, Frantisek Simancik, and Ian Horrocks. Extending consequence-based reasoning to SRIQ. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR'16)*, pages 187–196, 2016.

[11] Sebastian Brandt. Polynomial time reasoning in a description logic with existential restrictions, gci axioms, and-what else? In *Proceedings of the European Conference on Artificial Intelligence (ECAI'04)*, volume 16, page 298, 2004.

[12] Vinay K Chaudhri, Michael A Wessel, and Stijn Heymans. Kb_bio_101: A challenge for owl reasoners. In *Proceedings of the 2nd OWL Reasoner Evaluation Workshop (ORE'13)*, pages 114–120, 2013.

[13] Vasek Chvatal. *Linear programming*. Macmillan, 1983.

[14] David Tena Cucala, B Cuenca Grau, and Ian Horrocks. Consequence-based reasoning for description logics with disjunction, inverse roles, and nominals. In *Proceedings of the 30th International Workshop on Description Logics (DL'17)*, July 2017.

[15] David Tena Cucala, Bernardo Cuenca Grau, and Ian Horrocks. Consequence-based reasoning for description logics with disjunction, inverse roles, number restrictions, and nominals. *arXiv preprint arXiv:1805.01396*, 2018.

[16] David Tena Cucala, Bernardo Cuenca Grau, and Ian Horrocks. Sequoia: A consequence based reasoner for sroiq. *Description Logics*, 2373, 2019.

[17] George B Dantzig, Alex Orden, Philip Wolfe, et al. The generalized simplex method for minimizing a linear form under linear inequality restraints. *Pacific Journal of Mathematics*, 5(2):183–195, 1955.

[18] George B Dantzig and Philip Wolfe. Decomposition principle for linear programs. *Operations research*, 8(1):101–111, 1960.

[19] Jacques Desrosiers, François Soumis, and Martin Desrochers. Routing with time windows by column generation. *Networks*, 14(4):545–565, 1984.

[20] Yu Ding and Volker Haarslev. Tableau caching for description logics with inverse and transitive roles. In *Proceedings of the International Workshop on Description Logics (DL'06)*, pages 143–149, 2006.

[21] Jocelyne Faddoul. *Reasoning algebraically with description logics*. PhD thesis, Concordia University, 2011.

[22] Jocelyne Faddoul and Volker Haarslev. Algebraic tableau reasoning for the description logic SHOQ. *Journal of Applied Logic*, 8(4):334–355, 2010.

[23] Jocelyne Faddoul and Volker Haarslev. Optimizing algebraic tableau reasoning for SHOQ: First experimental results. In *Proceedings of the 23rd International Workshop on Description Logics (DL'10)*, page 161, 2010.

[24] Humaira Farid and Volker Haarslev. Handling nominals and inverse roles using algebraic reasoning. In *Proceedings of the 31st International Workshop on Description Logics (DL'18)*, oct 2018.

[25] Humaira Farid and Volker Haarslev. Handling nominals and inverse roles using algebraic reasoning. *arXiv preprint arXiv:1810.00916*, 2018.

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

[27] Lester Randolph Ford Jr and Delbert R Fulkerson. A suggested computation for maximal multi-commodity network flows. *Management Science*, 5(1):97–101, 1958.

[28] Jon William Freeman. *Improvements to propositional satisfiability search algorithms*. University of Pennsylvania, 1995.

[29] Paul C Gilmore and Ralph E Gomory. A linear programming approach to the cutting-stock problem. *Operations research*, 9(6):849–859, 1961.

[30] Paul C Gilmore and Ralph E Gomory. A linear programming approach to the cutting stock problem - part ii. *Operations research*, 11(6):863–888, 1963.

[31] Birte Glimm, Ian Horrocks, Boris Motik, Rob Shearer, and Giorgos Stoilos. A novel approach to ontology classification. *Web Semantics: Science, Services and Agents on the World Wide Web*, 14:84–101, 2012.

[32] Birte Glimm, Ian Horrocks, Boris Motik, Giorgos Stoilos, and Zhe Wang. Hermit: an owl 2 reasoner. *Journal of automated reasoning*, 53:245–269, 2014.

[33] Jennifer Golbeck, Gilberto Fragoso, Frank Hartel, Jim Hendler, Jim Oberthaler, and Bijan Parsia. The national cancer institute's thesaurus and ontology. *Journal of Web Semantics*, 1(1), 2003.

[34] Rajeev Goré and Linh Anh Nguyen. Exptime tableaux with global caching for description logics with transitive roles, inverse roles and role hierarchies. In *Proceedings of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX'07)*, pages 133–148. Springer, 2007.

[35] Volker Haarslev and Ralf Möller. Optimizing reasoning in description logics with qualified number restrictions. *Description Logics*, 49:142–151, 2001.

[36] Volker Haarslev and Ralf Möller. Racer system description. In *Proceedings of the International Joint Conference on Automated Reasoning (IJCAR'01)*, pages 701–705. Springer, 2001.

[37] 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'01)*, 2001.

[38] Ian Horrocks. Using an expressive description logic: FaCT or fiction? In *Proceedings of the 6th International Conference on Principles of Knowledge Representation and Reasoning (KR'98)*, volume 98, pages 636–645, 1998.

[39] Ian Horrocks and Ulrike Sattler. A description logic with transitive and inverse roles and role hierarchies. *Journal of logic and computation*, 9(3):385–410, 1999.

[40] Ian Horrocks and Ulrike Sattler. Optimised reasoning for shiq. In *Proceedings of the 15th European Conference on Artificial Intelligence (ECAI'02)*, pages 277–281, 2002.

[41] Ian Horrocks and Ulrike Sattler. A tableau decision procedure for SHOIQ. *Journal of automated reasoning*, 39(3):249–276, 2007.

[42] Ian Horrocks, Ulrike Sattler, and Stephan Tobies. Practical reasoning for expressive description logics. In *Proceedings of the International Conference on Logic for Programming Artificial Intelligence and Reasoning (LPAR'99)*, pages 161–180. Springer, 1999.

[43] Ian R Horrocks. *Optimising tableaux decision procedures for description logics*. The University of Manchester (United Kingdom), 1997.

[44] Narendra Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the 16h annual ACM symposium on Theory of computing (STOC 1984)*, pages 302–311. ACM, 1984.

[45] Yevgeny Kazakov et al. Consequence-driven reasoning for Horn SHIQ ontologies. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'09)*, volume 9, pages 2040–2045, 2009.

[46] Yevgeny Kazakov, Markus Krötzsch, and Frantisek Simancik. Practical reasoning with nominals in the EL family of description logics. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR'12)*, 2012.

[47] C Maria Keet, Agnieszka Lawrynowicz, Claudia d'Amato, and Melanie Hilario. Modeling issues & choices in the data mining optimization ontology. 2013.

[48] Leonid G Khachiyan. Polynomial algorithms in linear programming. *USSR Computational Mathematics and Mathematical Physics*, 20(1):53–72, 1980.

[49] Markus Krötzsch, Frederick Maier, Adila Krisnadhi, and Pascal Hitzler. A better uncle for OWL: Nominal schemas for integrating rules and ontologies. In *Proceedings of the 20th international conference on World Wide Web (WWW'11)*, pages 645–654. ACM, 2011.

[50] Carsten Lutz, Ulrike Sattler, and Lidia Tendera. The complexity of finite model reasoning in description logics. *Information and Computation*, 199(1-2):132–171, 2005.

[51] Nimrod Megiddo. *On the complexity of linear programming*. IBM Thomas J. Watson Research Division, 1986.

[52] Boris Motik, Rob Shearer, and Ian Horrocks. Optimizing the nominal introduction rule in (hyper) tableau calculi. In *Proceedings of the 21st International Workshop on Description Logics (DL'08)*, 2008.

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

[54] Hans Jürgen Ohlbach and Jana Koehler. Reasoning about sets via atomic decomposition. Technical report, International Computer Science Institute (ICSI), 1996.

[55] Hans Jürgen Ohlbach and Jana Koehler. Modal logics, description logics and arithmetic reasoning. *Artificial Intelligence*, 109(1-2):1–31, 1999.

[56] Magdalena Ortiz, Sebastian Rudolph, and Mantas Simkus. Worst-case optimal reasoning for the Horn-DL fragments of OWL 1 and 2. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR'10)*, 2010.

[57] Bijan Parsia and Evren Sirin. Pellet: An OWL DL reasoner. In *Proceedings of the 3rd International Semantic Web Conference (ISWC'04)*, volume 18, page 2. Publishing, 2004.

[58] Laleh Roosta Pour and Volker Haarslev. Algebraic reasoning for SHIQ. In *Proceedings of the International Workshop on Description Logics (DL'12)*, pages 530–540. Citeseer, 2012.

[59] Ian Pratt-Hartmann. Complexity of the two-variable fragment with counting quantifiers. *Journal of Logic, Language, and Information*, pages 369–395, 2005.

[60] Ian Pratt-Hartmann. On the computational complexity of the numerically definite syllogistic and related logics. *Bulletin of Symbolic Logic*, 14(1):1–28, 2008.

[61] Alan L Rector, JE Rogers, Pieter E Zanstra, and Egbert Van Der Haring. Opengalen: open source medical terminology and tools. In *Proceedings of the American Medical Informatics Association Annual Symposium (AMIA'03)*, volume 2003, page 982. American Medical Informatics Association, 2003.

[62] Cornelius Rosse and José LV Mejino Jr. A reference ontology for biomedical informatics: the foundational model of anatomy. *Journal of biomedical informatics*, 36(6):478–500, 2003.

[63] Matthias Samwald. Genomic cds: an example of a complex ontology for pharmacogenetics and clinical decision support. In *Proceedings of the 2nd OWL Reasoner Evaluation Workshop (ORE'13)*, pages 128–133. Citeseer, 2013.

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

[65] Stefan Schulz, Ronald Cornet, and Kent Spackman. Consolidating snomed ct's ontological commitment. *Applied ontology*, 6(1):1–11, 2011.

[66] Rob Shearer, Boris Motik, and Ian Horrocks. HermiT: A highly-efficient OWL reasoner. In *Proceedings of the OWL: Experiences and Directions (OWLED'08)*, volume 432, page 91, 2008.

[67] Evren Sirin, Bernardo Cuenca Grau, and Bijan Parsia. From wine to water: Optimizing description logic reasoning for nominals. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR'06)*, pages 90–99, 2006.

[68] Andreas Steigmiller, Birte Glimm, and Thorsten Liebig. Nominal schema absorption. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI'13)*, pages 1104–1110, 2013.

[69] Andreas Steigmiller, Birte Glimm, and Thorsten Liebig. Coupling tableau algorithms for expressive description logics with completion-based saturation procedures. In *Proceedings of the 7th International Joint Conference on Automated Reasoning (IJCAR'14)*, pages 449–463. Springer, 2014.

[70] Andreas Steigmiller, Thorsten Liebig, and Birte Glimm. Extended caching, backjumping and merging for expressive description logics. In *Proceedings of the International Joint Conference on Automated Reasoning (IJCAR'12)*, pages 514–529. Springer, 2012.

[71] Andreas Steigmiller, Thorsten Liebig, and Birte Glimm. Extended caching, backjumping and merging for expressive description logics. In *Proceedings of the 6th International Joint Conference on Automated Reasoning (IJCAR'12)*, pages 514–529, Berlin, Heidelberg, 2012. Springer.

[72] Andreas Steigmiller, Thorsten Liebig, and Birte Glimm. Konclude: system description. *Web Semantics: Science, Services and Agents on the World Wide Web*, 27:78–85, 2014.

[73] Dmitry Tsarkov and Ian Horrocks. Efficient reasoning with range and domain constraints. In *Proceedings of the International Workshop on Description Logics (DL'04)*, volume 104, pages 41–50, 2004.

[74] Dmitry Tsarkov and Ian Horrocks. FaCT++ description logic reasoner: System description. In *Proceedings of the International Joint Conference on Automated Reasoning (IJCAR'06)*, pages 292–297. Springer, 2006.

[75] Jelena Vlasenko, Maryam Daryalal, Volker Haarslev, and Brigitte Jaumard. A saturation-based algebraic reasoner for ELQ. In *Proceedings of the 5th Workshop on Practical Aspects of Automated Reasoning (PAAR'16), Affiliated with the 8th International Joint Conference on Automated*, page 110, July 2016.

[76] Jelena Vlasenko, Volker Haarslev, and Brigitte Jaumard. Pushing the boundaries of reasoning about qualified cardinality restrictions. In *Proceedings of the International Symposium on Frontiers of Combining Systems (FroCoS'17)*, pages 95–112. Springer, 2017.

[77] Nikoo Zolfaghar Karahroodi and Volker Haarslev. A consequence-based algebraic calculus for SHOQ. In *Proceedings of the International Workshop on Description Logics (DL'17)*, 2017.